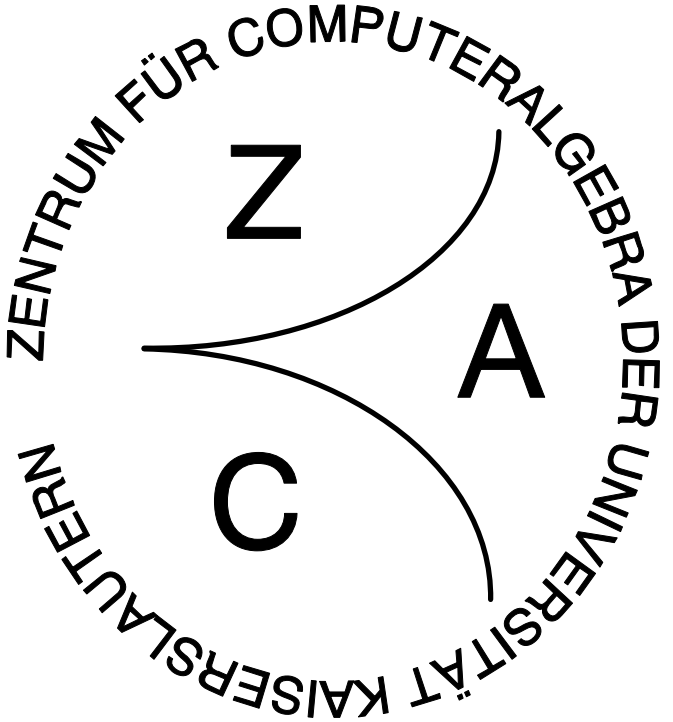
UNIVERSITÄT KAISERSLAUTERN
Zentrum für Computeralgebra

REPORTS ON COMPUTER ALGEBRA
NO. 11



Effective simplification of cr expressions

by

O. Bachmann

January 1997

The Zentrum für Computeralgebra (Centre for Computer Algebra) at the University of Kaiserslautern was founded in June 1993 by the Ministerium für Wissenschaft und Weiterbildung in Rheinland-Pfalz (Ministry of Science and Education of the state of Rheinland-Pfalz). The centre is a scientific institution of the departments of **Mathematics, Computer Science, and Electrical Engineering** at the University of Kaiserslautern.
The goals of the centre are to advance and to support the use of Computer Algebra in industry, research, and teaching. More concrete goals of the centre include

- the development, integration, and use of software for Computer Algebra
- the development of curricula in Computer Algebra under special consideration of interdisciplinary aspects
- the realisation of seminars about Computer Algebra
- the cooperation with other centres and institutions which have similar goals

The present coordinator of the Reports on Computer Algebra is:
Olaf Bachmann (email: `obachman@mathematik.uni-kl.de`)

Zentrum für Computeralgebra
c/o Prof. Dr. G.-M. Greuel, FB Mathematik
Erwin-Schrödinger-Strasse
D-67663 Kaiserslautern; Germany
Phone: 49 - 631/205-2850 Fax: 49 - 631/205-5052
email: `greuel@mathematik.uni-kl.de`
URL: `http://www.mathematik.uni-kl.de/~zca/`

Effective Simplification of CR expressions*

Olaf Bachmann
Centre for Computer Algebra
Department of Mathematics
University of Kaiserslautern
Kaiserslautern, Germany
obachman@mathematik.uni-kl.de

Abstract

Chains of Recurrences (CRs) are a tool for expediting the evaluation of elementary expressions over regular grids. CR based evaluations of elementary expressions consist of 3 major stages: CR construction, simplification, and evaluation. This paper addresses CR simplifications. The goal of CR simplifications is to manipulate a CR such that the resulting expression is more efficiently to evaluate. We develop CR simplification strategies which take the computational context of CR evaluations into account. Realizing that it is infeasible to always optimally simplify a CR expression, we give heuristic strategies which, in most cases, result in a optimal, or close-to-optimal expressions. The motivations behind our proposed strategies are discussed and the results are illustrated by various examples.

1 Introduction

Chains of Recurrence (CRs) address the problem of fast evaluations of elementary expressions over regular grids. Informally speaking, elementary expressions are expressions built from constants, variables, and elementary function symbols (e.g., rational or transcendental expressions) and a regular grid is a set of regularly spaced points (e.g., linearly spaced points on a line, points on the intersection of lines which are parallel to the coordinate axes, evenly distributed points on circles, ellipses, spheres, etc).

The main idea behind the CR method is: Instead of computing from scratch, an elementary expression can be evaluated at the next point much faster by using its values at previous points. That is, based on the given elementary expression and the relation of the evaluation points, we construct an equivalent CR expression, whose evaluation recursively connects consecutive evaluation values. CR expressions are similar to elementary expressions, except that Chains of Recurrences play the role of variables.

A CR based evaluation of an elementary expression F consists of three stages:

1. The construction of a CR expression Ψ which is equivalent to F.
2. The simplification of Ψ.
3. The evaluation of Ψ.

*Work reported herein has been part of the authors Ph.D. work at Kent State University (Ohio, USA) and has been supported in part by the National Science Foundation under Grant CCR-9503650.

Before proceeding further, let us illustrate these stages by an example. Suppose we wish to evaluate the polynomial $p(x) = 2x^3+x^2+x-3$ at the $1,001$ points 0,.01,. . .,9.99,10.0:

1. The CR expression $\Psi = 2\{0, +, .01\}^3 + \{0, +, .01\}^2 + \{0, +, .01\} - 3$ is obtained from the CR construction. Notice that Ψ is similar to p, except that each occurrence of x in p is replaced by the Chain of Recurrence $\{0, +, .01\}$.
2. Ψ is simplified:

$$\begin{aligned} & 2\{0, +, .01\}^3 + \{0, +, .01\}^2 + \{0, +, .01\} - 3 \\ & \Rightarrow 2\{0, +, .000001, +, .000006, +, .000006\} \\ & \quad +\{0, +, .0001, +, .0002\} + \{0, +, .01\} - 3 \\ & \Rightarrow \{0, +, .000002, +, .000012, +, .000012\} \\ & \quad +\{0, +, .0001, +, .0002\} + \{0, +, .01\} - 3 \\ & \Rightarrow \{-3, +, .010102, +, .000112, +, .000012\} \end{aligned}$$

   where all simplification steps are based on general simplification properties of CR expressions (see below).
3. The CR $\{-3, +, .010102, +, .000112, +, .000012\}$ is evaluated over $1,000$ points by a procedure which is of the following form:

```
φ0 := −3; φ1 := .010102; φ2 := .000112; φ3 := .000012;
for i := 0 to 1,000 do
    Ψ[i] := φ0;
    φ0 := φ0 + φ1; φ1 := φ1 + φ2; φ2 := φ2 + φ3;
od
```

   where $\Psi[i]$ contains the values of p at the points 0,.01,. . .,9.99,10.0.

We should notice that the evaluation of $\Phi[i]$ requires 3 additions per evaluation point. Therefore, compared with a Horner evaluation of p, we saved a total of $3,000$ multiplications. Due to this saving and due to judicious implementation techniques, a CR based evaluation of p can be significantly faster than a "normal" Horner evaluation (for this example, a speedup of 200 is reported in [4]).

In [2] algorithms for the construction and evaluation of multi - dimensional CRs are given, which assure that a CR based evaluation without simplification is at least as efficient as an evaluation of the given elementary expression. However, the true power, the "salt in the soup", of the CR method stems from the fact that certain classes of CR expressions (and hence, elementary expressions) can be simplified in such a way that evaluations of the resulting CR expression require significantly less arithmetic operations than

the original expression, enabling a significant increase in the evaluation efficiency.

However, as with other simplifications, there is a major problem with CR simplifications, namely that of the evasive and context-dependent concept of "simplicity": As outlined in [6], simpler might mean "closer to a canonical representation", "shorter", "needs less memory for a computer representation", "numerically more stable", etc; some concepts of simplicity might be undecidable (e.g., that of the canonical simplification of transcendental expressions, see [6]) or computationally very expensive to realize (e.g., critical pair simplification algorithms); simplicity w.r.t. one concept might be diametric w.r.t. another concept (e.g. compare $(x+1)^{100}$ and its expanded canonical representation w.r.t. shortness); and so on.

Clearly, in our context "simpler" means "can be evaluated more efficiently". However, already this qualification is problematic, since it does not reflect some other, and often important, evaluation criteria, such as numeric stability, memory consumption, and the time spent for the simplification itself. But even the sole use of "evaluation efficiency" as a measure for simplicity is problematic: The evaluation efficiency of a given CR expression depends on many, possibly varying, factors, such as the complexity and location of the evaluation region (i.e., the regular grid), the underlying computational domain (e.g., floating point or arbitrary precision numbers), the used hard- and software platform, etc. In this paper we develop new simplification strategies which take these varying factors into account, thereby closing important gaps of previous works about CRs ([3, 5, 8, 9]):

- The previously suggested unconditional application of CR simplification rules leads to often very unsatisfiable results for more complex expressions (like those containing trigonometric functions, or high degree polynomials).
- The problem of the variable ordering of multi - dimensional CR expressions was not addressed.

The results given here are based on the partial solutions of this problem for the two-dimensional case, as described in [4].

In the ideal case, we would like to find a simplification procedure that is optimal in the sense that it produces the CR expression that is the most efficient to evaluate, under consideration of all factors of the computational context. However, this is probably not realizable, given the complexity of such a task. Realistically, we can only hope for simplification results which are optimal or very close to optimal in *most* cases.

To achieve this goal, we introduce in Section 2 the concept of the Cost Index (CI) as an approximate measure of the efficiency of CR evaluations. This is followed an analysis of the effect of CR simplification rules on the CI of the transformed CR expression (Section 3). Based on this analysis, we develop CR simplification strategies in Section 4 and conclude this by considering an extended set of examples in Section 5.

2 CRs and their Cost Index

Before returning to the problem of simplifying CR expressions, let us briefly define the needed CR concepts. Our following definition of CR expressions follows the concepts introduced in [2] which are generalizations of the respective concepts used in earlier work on CRs (see, e.g., [5] or [3]).

Definition 1 *Let $\mathcal{R}$ be a ring with identity. Let $\mathcal{R}_{\mathcal{F}}$ be a set of symbols for allowable $\mathcal{R}^m \to \mathcal{R}$ functions and $\mathcal{X}$ be a set of allowable variables $\{x_1, x_2, \ldots, x_n\}$. The set $\mathcal{CR}$ of* CR expressions *over $(\mathcal{R}_{\mathcal{F}}, \mathcal{X})$ is the minimal set of terms such that*

(i) $\mathcal{R} \subset \mathcal{CR}$,

(ii) *for any $x_k \in \mathcal{X}; \Phi_0, \Phi_1, \ldots, \Phi_l \in \mathcal{CR}; \odot_1, \ldots, \odot_l \in \{+, *\}$ the term $\{\Phi_0, \odot_1, \Phi_1, \odot_2, \ldots, \odot_l, \Phi_l\}_{x_k} \in \mathcal{CR}$, and*

(iii) *for any $f \in \mathcal{R}_{\mathcal{F}}; \Psi_1, \Psi_2, \ldots, \Psi_m \in \mathcal{CR}$ the term $f\Psi_1\Psi_2\ldots\Psi_m \in \mathcal{CR}$*

CR expressions of the form $\{\Phi_0, \odot_1, \Phi_1, \odot_2, \ldots, \odot_l, \Phi_l\}_{x_k}$ are also called Chains of Recurrences (CRs).

Moreover, if $\Phi \in \mathcal{CR}$ then we define the value *of Φ, denoted by $V(\Phi)$, to be a function $V(\Phi) : \mathbf{N}^n \to \mathcal{R}$ which is recursively defined by*

$V(\Phi)(\mathbf{i})$	*Condition*
r	$\Phi = r \in \mathcal{R}$
$V(\Phi_0)(\mathbf{i})$	$\Phi = \{\Phi_0\}_{x_k}$
$V(\Phi_0)(\mathbf{i}) + \sum_{j=0}^{i_k-1} V(\{\Phi_1, \odot_2, \ldots, \odot_l, \Phi_l\}_{x_k})(\mathbf{i})$	$\Phi = \{\Phi_0, +, \Phi_1, \odot_2, \ldots, \odot_l, \Phi_l\}_{x_k}$
$V(\Phi_0)(\mathbf{i}) \cdot \prod_{j=0}^{i_k-1} V(\{\Phi_1, \odot_2, \ldots, \odot_l, \Phi_l\}_{x_k})(\mathbf{i})$	$\Phi = \{\Phi_0, *, \Phi_1, \odot_2, \ldots, \odot_l, \Phi_l\}_{x_k}$
$f(V(\Psi_1)(\mathbf{i}), \ldots, V(\Psi_m)(\mathbf{i}))$	$\Phi = f\Psi_1\Psi_2\ldots\Psi_m$

Since $\mathcal{R}$ is a ring with identity, the summation and product of elements of $\mathcal{R}$ is always well defined. If it is clear from the context, we will often simply write Φ instead of $V(\Phi)$ and $\mathcal{CR}$ instead of $\mathcal{CR}(\mathcal{R}_{\mathcal{F}}, \mathcal{X}_n)$. Furthermore, we define the set $X(\Phi)$ of variables contained in Φ by

$$X(\Phi) = \begin{cases} x_k \cup \bigcup_{j=0}^{l} X(\Psi_j) & \text{if } \Phi = \{\Phi_0, \odot_1, \ldots, \odot_l, \Phi_l\}_{x_k} \\ X(\Psi_1) \cup \ldots \cup X(\Psi_m) & \text{if } \Phi = f\Psi_1\ldots\Psi_m \end{cases}$$

and the dimension $D(\Phi)$ of Φ by the cardinality of $X(\Phi)$, i.e., $D(\Phi) = |X(\Phi)|$.
If $\Phi = \{\Phi_0, \odot_1\Phi_1, \odot_2, \ldots, \odot_l, \Phi_l\}_{x_k}$ is a CR, then we call

- $\Phi_0, \ldots, \Phi_l$ the *coefficients* of Φ
- $l = L(\Phi)$ the *length* of Φ
- Φ *regular*, if $x_k \notin X(\Phi_0) \cup X(\Phi_1) \cup \ldots \cup X(\Phi_{l-1})$
- Φ *simple*, if Φ is regular and $x_k \notin X(\Phi_l)$
- Φ *polynomial*, if Φ is regular and $\odot_1 = \cdots = \odot_l = +$
- Φ *exponential*, if Φ is regular and $\odot_1 = \cdots = \odot_l = *$

and will frequently use the following abbreviations

- Φ for $\Phi(\mathbf{i})$
- indexed lower-case Greek letters (like φ_j) for CR coefficients for which $x_k \notin X(\Phi_k)$.

- φ_j for $\varphi_k(\mathbf{i})$
- $\{\Phi_0, \odot_1, \ldots, \odot_l, \Phi_l\}$ and $\Phi(i)$ if Φ is a one - dimensional CR

In addition, we extend the concepts of regular, simple, and polynomial CRs to CR expressions, by calling a CR expressions Φ regular/simple/polynomial if all the CRs contained in Φ are regular/simple/polynomial and, for polynomial CR expressions, all the (function) symbols f_k contained in Φ are constants, or contained in $\{+, -, \cdot\}$.

The following definition of the Cost Index (CI) extends the previous definition given [5]: Instead of defining the CI as an operation count of the evaluation of one - dimensional CR expressions, we use a weighted operation count of the evaluation efficiency of multi - dimensional CR expressions.

Definition 2 *Let Φ be a CR expression over $\mathcal{CR}(\mathcal{R}_\mathcal{F}, \mathcal{X}_n)$ and let $W : \mathcal{R}_\mathcal{F} \to \mathbf{R}$ be a map which assigns each $f \in \mathcal{R}_\mathcal{F}$ a real number (which is called the* weight *of f) such that $W(f) = 0$ if f is a constant and $W(+) = 1$ (where $+$ is the symbol for addition). Furthermore, let $\mathbf{m} \in \mathbf{N}^n$ and $m = \prod_{i=1}^{n} m_i$. Then we recursively define the* Cost Index (CI) *of Φ by $CI(\Phi) =$*

$$\begin{cases} 0 & \text{if } \Phi \in \mathcal{CR} \\ \frac{m_k}{m}\sum_{i=1}^{l} W(\odot_l) + \sum_{i=0}^{l} CI(\Phi_i) & \text{if } \Phi = \{\Phi_0, \odot_1, \ldots, \odot_l, \Phi_l\}_{x_k} \\ \frac{W(f_k)}{m} \prod_{i:x_i \in X(\Phi)} m_i + \sum_{i=1}^{n_k} CI(\Psi_i) & \text{if } \Phi = f\ \Psi_1 \ldots \Psi_m \end{cases}$$

In other words, we may consider the CI of a CR expression as an approximate measure of the average time (w.r.t. one addition) spent in the inner-most loop of a CR evaluation.

If we assume that the execution times of the basic arithmetic operations (i.e., the operations computing the value of $f(r_1, \ldots, r_m)$) were largely independent of their arguments, then we can assign a weight to each arithmetic operation such that it approximates the execution time of the operation relative to the speed of one addition. For example, for `double` floating point operations, we measured the following relative and averaged execution times[1]. As the table indicates, the relative weights may vary greatly from platform to platform.

If we furthermore assume that the execution time of the evaluation algorithm considered is directly proportional to the weighted operation count of the basic arithmetic operations, then we can use the Cost Index of a CR expression as a relative measure of its evaluation efficiency.

However, supposing that the execution times of the basic arithmetic operations are largely independent of their arguments is a strong assumption. First, it is definitely not true for arbitrary precision arithmetic since the execution times of arbitrary precision operations clearly depend on the size of the operands. Secondly, even in fixed precision arithmetic the execution times of most basic arithmetic operations are not truly independent of their arguments. For example, consider the operation `pow`(x, y). If y is an integer, then we do not need to use a numeric approximation to obtain the value of x^y, but can use a repeated squaring procedure (see [7], Section 3.4) which requires at most $2\lfloor \log y \rfloor$ multiplications and one division.

Unfortunately, there seems to be no realistic alternative to this assumption, since we can not exactly predict the operands of the basic arithmetic operations at simplification time. Therefore, we can only try to approximate the real execution times as much as possible, by

1. estimating the size of the operands for arbitrary precision arithmetics
2. using an average of the execution times for numeric approximations
3. approximating the execution time of special cases (like x^n, where it is known at simplification time that n is an integer) by a separate estimate

3 Cost Index and CR simplifications

In this section we examine the major CR simplification rules w.r.t. their effect on the CI of the transformed CR expression.

Table 1 lists the major rules for simplifying multi - dimensional CR expressions. Most of these are generalizations of the rules given in [5, 3, 9] from one- or two-dimensional CR expressions to multi - dimensional CR expressions. For a more detailed description and a proofs of these rules, see [2].

The left hand sides (LHS) of the simplification rules are given in column 2 and the respective right hand sides (RHS) are given in column 3. For reasons of clarity, the trivial simplification rules for general CR expressions based on the ring axioms are not shown.

In general, the difference $CI(LHS) - CI(RHS)$ is dependent on the complexity of the regular grid, on the weights of the operations involved, and on a chosen variable ordering. Column 4 marks the cases where $CI(LHS) - CI(RHS) > 0$ independently of all of these factors (the symbol $\wr$ is used to mark the rules where $CI(LHS) - CI(RHS) > 0$ can not be decided a priori). Column 5 shows $CI(LHS) - CI(RHS)$ of one - dimensional CR expressions for which this difference depends at most on the weights of the operations involved.

For one - dimensional CRs, the application of a simplification rule is always unique in the sense that the same rule can not be applied to the same CR expression and yield different simplification results. However, this is, in general, not the case for multi - dimensional CRs. Let us illustrate this by an example: Let $\Phi = \{\varphi_0, +, \varphi_1\}_x$ and $\Psi = \{\psi_0, +, \psi_1\}_y$ and consider $\Phi \cdot \Psi$. Then by (2), i.e., by

$$\{\varphi_0, +, \ldots, +, \varphi_l\}_{x_k} \cdot \psi = \{\varphi_0 \cdot \psi, +, \ldots, +, \varphi_l \cdot \psi\}_{x_k}$$

with $x_k \notin X(\psi)$, we can simplify $\Phi \cdot \Psi$ to

$$\Lambda = \{\{\varphi_0\psi_0, +, \varphi_0\psi_1\}_y, +, \{\varphi_1\psi_0, +, \varphi_1\psi_1\}_y\}_x,$$

and alternatively to

$$\Lambda' = \{\{\varphi_0\psi_0, +, \varphi_1\psi_0\}_x, +, \{\varphi_0\psi_1, +, \varphi_1\psi_1\}_x\}_y,$$

where

$$CI(\Lambda) = 1 + \frac{2}{m_x} \qquad CI(\Lambda') = 1 + \frac{2}{m_y}.$$

[1] The measurements were done using 1,000,000 randomly generated `double` numbers and are relative w.r.t. the addition (i.e., the time for the addition was taken to be the unit time of the measurements)

SparcStationII

add	mult	div	sqrt	exp	log	pow	sin	tan	asin	atan	sinh	tanh
1.0	1.1	3.1	22.3	26.5	20.3	77.5	21.3	27.1	32.7	38.8	25.8	33.1

HP 9000/735

add	mult	div	sqrt	exp	log	pow	sin	tan	asin	atan	sinh	tanh
1.0	1.0	3.3	5.9	9.3	25.1	124.0	9.9	26.3	30.6	34.1	33.14	38.1

Table 1: Measured average relative execution times of `double` operations on a SparcStationII and HP 9000/735

Hence, if $m_x = 1000$ and $m_y = 10$ then an evaluation of Λ requires 1980 more additions than an evaluation of Λ'.

More generally, the freedom of choosing a variable ordering implies an additional degree of freedom in applying certain simplification rules which consequently has an impact on the CI of the simplified CR expression. Column 6 of table 1 marks all simplification rules to which this applies with the symbol $\surd$.

4 CR simplification strategies

Evaluating the results of table 1 we see that for most rules we can not decide a priori whether they are CI reducing. Therefore, the previously suggested CR simplification algorithms ([5], [3], [8], [9]) which recursively traverse the parse tree of a CR expression from the bottom to the top and unconditionally apply the simplification rules, do not, in general, result in a CR expression with a minimal CI.

An obvious algorithm which solves this problem is based on the observation that there are only finitely many CR expressions Φ' into which a given CR expression Φ can be transformed by applications of the transformation rules of table 1. Hence, for a first, straight-forward approach, we may use an exhaustive search algorithm which, informally speaking,

- works on lists of CR expressions and recursively obtains *all* CR expressions into which each CR expression of the current list can be transformed
- if a CR simplification rule can be applied to the top node of the CR expression currently under consideration, then return a list of the expressions obtained by not applying the rule, by applying the rule, and, possibly, by applying the rule with a different variable ordering

and, consequently, returns a list of *all* CR expressions Φ' into which the given input CR expression Φ can be transformed. By computing the CI of each CR expression obtained we can then easily find the one with the minimal CI.

Unfortunately, as might be expected, this algorithm has a (worst case) exponential time and space complexity w.r.t. the number of nodes of the given CR expression. Therefore its usage is limited to applications in which the CR simplification time does not contribute to the overall evaluation time (e.g., to source code generation and optimization problems) and to computational contexts with ample computing resources (especially memory).

However, a majority of applications require "on-the-fly" evaluations of elementary expressions (e.g., visualization of mathematical objects, see [1]). Hence, if we use the CR method for such computations, then the CR simplification time counts towards the overall evaluation time. Consequently, it is necessary to develop an alternative to the exhaustive search simplification algorithm. The goal of such an alternative algorithm is to balance the time spent during simplifications with the time saved in the following evaluations. In other words, we may not expect the algorithm to always find the CR expression with the minimum CI. Instead, we may expect that the algorithm simplifies a CR expression in a reasonable time (i.e., in at most polynomial time) such that the CI of the resulting CR expression is *close* to the minimum. To accomplish this goal, we need to make some ad-hoc decisions which are based on experience and observations, instead of strict reasoning. We refer to the following simplification as a simplification heuristic.

The first heuristic decision is based on the observation that exponential CRs are obtained by transformations of polynomial CRs and that there is only one rule which transforms an exponential CR back into a polynomial CR (rule (7)). Therefore, we suggest arranging CR simplification in two independent passes: The first pass accomplishes the simplification of all polynomial CR (sub-) expressions, and the second pass considers the simplification of exponential CRs.

4.1 Heuristic simplification of polynomial CR expressions

Given CR a expression Φ, we first isolate all polynomial CR sub-expressions of Φ and consider the simplification of each polynomial CR sub-expression independently. Since a CR simplification follows the CR construction, we may assume that the expressions considered are polynomial CR expressions whose leaves are simple, one - dimensional CRs.

Secondly, we determine the dimension of the polynomial CR expression and handle the simplification of one- and multi - dimensional CR expressions separately.

For both cases, the suggested simplification heuristic makes use of the degree of a polynomial CR expression, which is defined as follows:

$Deg(\Phi, x_k)$	Condition
0	$x_k \notin X(\Phi)$
l	$\Phi = \{\varphi_0, +, \ldots, +, \varphi_l\}_{x_k}$
$\max\{Deg(\varphi_0, x_k), \ldots, Deg(\varphi_l, x_k)\}$	$\Phi = \{\varphi_0, +, \ldots, +, \varphi_l\}_{x_j}$ $x_j \neq x_k$
$\max\{Deg(\Phi_1, x_k), \ldots, Deg(\Phi_k, x_k)\}$	$\Phi = \Phi_1 + \cdots + \Phi_k$
$\sum_{j=1}^{k} Deg(\Phi_j, x_k)$	$\Phi = \Phi_1 \cdot \cdots \cdot \Phi_k$
$n \cdot Deg(\Phi_1, x_k)$	$\Phi = \Phi_1^n$

Based on this definition, we suggest a simplification algorithm for the one - dimensional case which is based on the observation that a one - dimensional polynomial CR expression Φ can be transformed into a polynomial CR of length $Deg(\Phi, x)$.

Algorithm CRPolySimp1D(Φ)

	LHS	RHS	$CI(LHS)-CI(RHS)$ >0	1 - dim	order-dependent
(1)	$\{\varphi_0, +, \ldots, +, \varphi_l\}_{x_k} + \psi$	$\{\varphi_0 + \psi, +, \ldots, +, \varphi_l\}_{x_k}$	$\surd$	1	$\surd$
(2)	$\{\varphi_0, +, \ldots, +, \varphi_l\}_{x_k} \cdot \psi$	$\{\varphi_0 \cdot \psi, +, \ldots, +, \varphi_l \cdot \psi\}_{x_k}$	$\wr$	w_*	$\surd$
(3)	$\Phi + \{\psi_0, +, \ldots, +, \psi_m\}_{x_k}$	$\{\beta_0, +, \ldots, +, \beta_l\}_{x_k}$	$\surd$	$m+1$	/
(4)	$\Phi \cdot \{\psi_0, +, \ldots, +, \psi_m\}_{x_k}$	$\{\gamma_0, +, \ldots, +, \gamma_{l+m}\}_{x_k}$	$\wr$	1	/
(5)	$\{\varphi_0, +, \ldots, +, \varphi_l\}_{x_k}^m$	$\{\delta_0, +, \ldots, +, \delta_{l \cdot m}\}_{x_k}$	$\wr$	$2w_* \lfloor \log m \rfloor - n(l-1)$	/
(6)	$\psi^{\{\varphi_0, +, \ldots, +, \varphi_l\}_{x_k}}$	$\{\psi^{\varphi_0}, *, \ldots, +, \psi^{\varphi_l}\}_{x_k}$	$\wr$	$w_{\texttt{pow}} - l(w_*-1)$	/
(7)	$\log_\psi \{\varphi_0, *, \ldots, *, \varphi_l\}_{x_k}$	$\{\log_\psi \varphi_0, +, \ldots, +, \log_\psi \varphi_l\}_{x_k}$	$\wr$	$w_{\texttt{log}} + l(w_*-1)$	/
(8)	$\psi \cdot \{\varphi_0, *, \ldots, *, \varphi_l\}_{x_k}$	$\{\psi \cdot \varphi_0, *, \ldots, *, \varphi_l\}_{x_k}$	$\surd$	w_*	$\surd$
(9)	$\{\varphi_0, *, \ldots, *, \varphi_l\}_{x_k}^\psi$	$\{\varphi_0^\psi, *, \ldots, *, \varphi_l^\psi\}_{x_k}$	$\wr$	$w_{\texttt{pow}}$	/
(10)	$\{\varphi_0, *, \ldots, *, \varphi_l\}_{x_k} \cdot \Psi$	$\{\beta_0, *, \ldots, *, \beta_l\}_{x_k}$	$\surd$	$(m+1)w_*$	/
(11)	$\Psi^{\{\varphi_0, +, \ldots, +, \varphi_l\}_{x_k}}$	$\{\gamma_0, *, \ldots, *, \gamma_{m+l}\}_{x_k}$	$\wr$	$w_{\texttt{pow}} - l(w_*-1)$	/
(12)	$\sin\{\varphi_0, +, \ldots, +, \varphi_l\}_{x_k}$	$\Im\{e^{\imath\varphi_0}, *, \ldots, *, e^{\imath\varphi_l}\}_{x_k}$	$\wr$	$w_{\texttt{sin}} - l(4w_*+1)$	/
(13)	$\cos\{\varphi_0, +, \ldots, +, \varphi_l\}_{x_k}$	$\Re\{e^{\imath\varphi_0}, *, \ldots, *, e^{\imath\varphi_l}\}_{x_k}$	$\wr$	$w_{\texttt{cos}} - l(4w_*+1)$	/
(14)	$\tan\{\varphi_0, +, \ldots, +, \varphi_l\}_{x_k}$	$\frac{\Im}{\Re}\{e^{\imath\varphi_0}, *, \ldots, *, e^{\imath\varphi_l}\}_{x_k}$	$\wr$	$w_{\texttt{tan}} - w_{/} - l(4w_*+1)$	/
(15)	$\cot\{\varphi_0, +, \ldots, +, \varphi_l\}_{x_k}$	$\frac{\Re}{\Im}\{e^{\imath\varphi_0}, *, \ldots, *, e^{\imath\varphi_l}\}_{x_k}$	$\wr$	$w_{\texttt{cot}} - w_{/} - l(4w_*+1)$	/

$\varphi_0, \ldots, \varphi_l, \psi_0, \ldots, \psi_m, \psi \in \mathcal{CR}(\mathcal{R}_\mathcal{F}, \mathcal{X}_n)$, $x_k \notin X(\varphi_0) \cup \cdots \cup X(\varphi_l) \cup X(\psi_0) \cup \cdots \cup X(\psi_m)$, $\Phi = \{\varphi_0, +, \ldots, +, \varphi_l\}_{x_k}$, $\Psi = \{\psi_0, *, \ldots, *, \psi_m\}_{x_k}$, $l \geq m$, $\imath = \sqrt{-1}$, $\Re(a + \imath b) = a$ $\Im(a + \imath b) = b$

Figure 1: Properties of CR simplification rules

Input: Polynomial CR expression Φ with $X(\Phi) = \{x\}$

Output: Simplified polynomial CR expression Φ' with $CI(\Phi) \geq CI(\Phi')$

(A) If Φ is a constant or a CR then return Φ;

(B) If $Deg(\Phi, x) < CI(\Phi)$ then

1. Recursively simplify Φ by unconditionally applying the simplification rules (1) – (6) and return the resulting polynomial CR of length c_s;

(C) else if $\Phi = \Phi_1 + \cdots + \Phi_k$ then

1. Set Φ_j' =CRPolySimp1D(Φ_j) for $1 \leq j \leq k$;
2. Transform all constant or polynomial CRs among the returned Φ_j' into one polynomial CR (or constant) by applying the rules (1) and (3) and return the resulting CR expression.

(D) else if $\Phi = \Phi_1 \cdot \cdots \cdot \Phi_k$ then

1. Set Φ_j' =CRPolySimp1D(Φ_j) for $1 \leq j \leq k$;
2. Transform all constant or polynomial CRs among the returned Φ_j' into one constant or polynomial CR by applying the rules (2) and (4) and return the resulting CR expression.

(E) else /* Now $\Phi = \Phi_1^n$ */

Set Φ =CRPolySimp1D(Φ_1) and return Φ^n

Notice that (B) is the crucial step of this algorithm, since it determines whether or not the CR expression is fully expanded into a polynomial CR. This is also the only step where we might lose the optimality of the CR simplification. For example, if we consider the CR expression $\Phi = \{x_0, +, h\}^{63} + \sum_{j=1}^{12} j \cdot \{x_0, +, h\}^j$ and assume that $W(*) = 1$ then $CI(\Phi) = 2\lfloor \log_2 63 \rfloor + \sum_{j=1}^{12}(2 + 1\lfloor \log_2 j \rfloor) = 84$, $Deg(\Phi) = 63$, and hence, Φ is simplified into a polynomial CR of length 63, i.e., $CI(\Phi') = 63$. However, a simplification into $\Phi' = \{x_0, +, h\}^{63} + \{\varphi_1, +, \ldots, +, \varphi_{12}\}$ would yield $CI(\Phi') = 33$, which is better. However, such cases could only be remedied by backtracking algorithms which, by their very nature, have a (worst-case) exponential time complexity.

The heuristic simplification of n-dimensional polynomial CR expressions is similar to the one - dimensional case. However, in addition to determining whether or not to expand the expression into a polynomial CR, we also have to determine a variable ordering for the expansion. Furthermore, we have to take the complexity of the evaluation grid into consideration, since the CI of multi - dimensional CR expressions depends on the number of evaluation points for each variable.

Algorithm CRPolySimpND($\Phi, \mathbf{m}$)

Input: Polynomial CR expression Φ with $X(\Phi) \subseteq \{x_1, \ldots, x_n\}$, $\mathbf{m} \in \mathbf{N}^n$

Output: Simplified polynomial CR expression Φ' with $CI(\Phi) \geq CI(\Phi')$

(A) If Φ is a constant or a CR then return Φ;

(B) If $X(\Phi) = x_k$ then return CRPolySimp1D(Φ);

(C) Let $J = (j_1, \ldots, j_s)$ be a tuple of s integers, such that $\{x_{j_1}, \ldots, x_{j_s}\} = X(\Phi)$ and for all $1 \leq k < s$: $d_{j_k} < d_{j_{k+1}}$ or $d_{j_k} = d_{j_{k+1}}$ and $m_{j_k} \geq m_{j_{k+1}}$, where $d_{j_k} = Deg(\Phi, x_{j_k})$;

(D) If $d_{j_1} + \frac{1}{m_{j_1}}(d_{j_2} + \frac{1}{m_{j_2}}(d_{j_3} + \ldots)) \leq CI(\Phi, \mathbf{m})$ then

1. Recursively simplify Φ by unconditionally applying the simplification rules (1) – (6) with the variable ordering $x_{j_1} > x_{j_2} > \cdots > x_{j_n}$ and return the result.

(C) else if $\Phi = \Phi_1 + \cdots + \Phi_k$ then

1. Set Φ'_j =CRPolySimpND(Φ_j) for $1 \leq j \leq k$;
2. Merge constant or polynomial CRs among the returned Φ'_j as much as possible by applying the rules (1) and (3) and return the resulting CR expression.

(D) else if $\Phi = \Phi_1 \cdot \cdots \cdot \Phi_k$ then

1. Set Φ'_j =CRPolySimpND(Φ_j) for $1 \leq j \leq k$;
2. Merge constants or one - dimensional polynomial CRs among the returned Φ'_j as much as possible by applying the rules (2) and (4) and return the resulting CR expression.

(E) else /* Now $\Phi = \Phi_1^n$ */

Set Φ =CRPolySimpND(Φ_1) and return Φ^n

Notice that the variable ordering is determined in step (C). Our heuristic for determining this ordering is based on the observation that for any variable ordering $x_{i_1} > x_{i_2} > \cdots > x_{i_s}$, $CI(\Phi') \leq d_{i_1} + \frac{1}{m_{i_1}}(d_{i_2} + \frac{1}{m_{i_2}}(d_{i_3} + \ldots))$. Hence, by choosing the variable ordering $x_{j_1} > x_{i_2} > \cdots > x_{j_s}$ as done in step (C), we minimize the upper bound for the CI of the respective fully expanded polynomial CR.

Various small and tedious to describe, but more or less obvious improvements to the algorithm above should be added. For example, in step C.1 (resp. D.1), we should collect all $\Phi_{k_1}, \ldots, \Phi_{k_j}$ for which $X(\Phi_{k_1}) = \ldots = X(\Phi_{k_j})$ and call the algorithm recursively with the expressions $\Phi_{k_1} + \cdots \Phi_{k_j}$ as arguments (provided they are different from Φ) instead of calling the algorithm recursively with each single Φ_k as argument. This way, we enable a finer grained simplification of CR expressions which contains the same variables.

Let us look at some examples by considering the simplification of two-dimensional polynomial CR expressions over the variables x and y: To simplify our notation, let us write x for the CR $\{x_0, +, h_x\}_x$, y for the CR $\{y_0, +, h_y\}_y$, and x_l (resp. y_l) for a CR of the form $\{\varphi_0, +, \varphi_1, +, \ldots, +, \varphi_l\}_x$ for some constants $\varphi_0, \ldots, \varphi_l$. Furthermore, let us suppose that $\mathbf{m} = (100, 100)$ and that $W(*) = 1$. Then each entry in table 2 shows a two-dimensional CR expression with its CI. Column 1 shows the unsimplified CR expression, column 2 and 3 show the fully-expanded polynomial CRs with the variable ordering $x > y$ and $y > x$, respectively. Column 4 shows the CR expression returned by the algorithm CRPolySimpND and column 5 shows the respective CR expression with the minimal CI. Obviously, the larger the degree of the CR expressions, the larger the difference between the various simplification methods. However, the examples in table 2 already illustrate that a simplification algorithm which would fully expand a given CR expression (using any variable ordering) is inferior to CRPolySimpND, and that CRPolySimpND is not as good as the exhaustive search simplification algorithm.

4.2 Heuristic simplification of exponential CR expressions

The second pass of the heuristic simplification procedure is concerned with exponential CRs. Similar to the simplification of polynomial CRs, we first isolate all subexpressions of a CR expression which either already are, or could potentially be, transformed into an exponential CR and then deal with each of these expressions separately. Therefore, we may assume that the expressions which are to be simplified are CR expressions whose leaves are either already simplified polynomial CRs or are simple, one - dimensional exponential CRs.

Secondly, we once again treat the one - dimensional case differently from the n-dimensional case.

The simplification of one - dimensional exponential CR expressions is very similar to the simplification of one - dimensional polynomial CR expressions, except that we can not use the degree of an expression to predetermine its CI. Instead, we symbolically determine the structure of the exponential CR expression which is obtained by unconditionally applying all simplification rules and can then easily read its CI. Hence, the algorithm for simplifying one - dimensional exponential CR expressions proceeds as follows:

Algorithm CRExpSimp1D(Φ)

Input: CR expression Φ with $X(\Phi) = \{x\}$

Output: Simplified CR expression Φ' with $CI(\Phi) \geq CI(\Phi')$

(A) If Φ is a constant or a CR then return Φ;

(B) Let c_a be the CI of the expression which would be obtained from Φ by unconditionally applying the simplification rules (6) – (15).

(C) If $c_a < CI(\Phi)$ then
Unconditionally apply the simplification rules (6) – (15) to Φ and return the result.

(D) else /* Now $\Phi = f\ \Psi_1 \ldots \Psi_m$ */
Return f CRExpSimp1D(Ψ_1)...CRExpSimp1D(Ψ_m).

For example, if we are given the expression $\Phi = \sin\{2, +, 1, +, -1\} \cdot 2^{\{0, +, 1\}}$, then to realize step (B) we symbolically apply all simplification rules and obtain the symbolic CR expression $\Re\{\psi_0, *, \psi_1, *, \psi_2\}$ which has a CI of $c_s = 8w_* + 4w_+$. Consequently, if $c_s < CI(\Phi) = 3w_+ + w_{\texttt{sin}} + w_{\texttt{pow}}$ then we actually apply all simplification rules to Φ and return the results. Otherwise, we call the algorithm recursively on $\sin\{2, +, 1, +, -1\}$ and $2^{\{0, +, 1\}}$ before returning the result.

We apply an even simpler strategy for n-dimensional exponential CR expressions by basing the decision whether or not to apply a simplification rule on local criteria, only. The reason for not taking properties of the entire expression into account is that the multi - dimensional polynomial CRs which are the leaves of the expression already predetermine the variable orderings, and furthermore greatly complicate the construction of symbolic expressions:

Algorithm CRExpSimpND(Φ)

Input: Multi - dimensional, exponential CR expression Φ.

Output: Simplified CR expression Φ' with $CI(\Phi) \geq CI(\Phi')$

(A) If Φ is a constant or a CR then return Φ;

(B) If Φ is a one - dimensional CR expression return CRExpSimp1D(Φ);

(D) else /* Now $\Phi = f\ \Psi_1 \ldots \Psi_m$ */

1. Set Ψ'_j =CRExpSimpND(Φ_j) for $1 \leq j \leq n_k$;

	unsimplified	$x > y$ simplified	$y > x$ simplified	CRPolySimpND	optimal
1	$xy^2 + 1$	$\{y_2, y_2\}_x$	$\{x_1, x_1, x_1\}_y$	$\{y_2, y_2\}_x$	$\{y_2, y_2\}_x$
CI	2.03	1.04	2.03	1.04	1.04
2	$xy^2 + x^3 + y$	$\{y_2, y_2, \varphi, \psi\}_x$	$\{x_3, x_1, x_1\}_y$	$\{x_3, x_1, x_1\}_y$	$\{x_3, x_1, x_1\}_y$
CI	3.06	3.04	2.05	2.05	2.05
3	$(xy^2 + 3x)^2$	$\{y_4, y_4, y_4\}_x$	$\{x_2, x_2, x_1, x_1, x_1\}_y$	$\{y_4, y_4, y_4\}_x$	$\{y_2, y_2\}_x^2$
CI	3.05	2.12	4.04	2.12	2.04
4	$(4y + 3x)^3$	$\{y_3, y_2, y_1, \varphi\}_x$	$\{x_3, x_2, x_1, \psi\}_y$	$\{y_1, y_1\}^3$	$\{y_1, y_1\}^3$
CI	3.04	3.06	3.06	3.02	3.02

Table 2: Examples for simplifications of two-dimensional polynomial CR expressions

2. If one of the simplification rules (6) – (15) applies to $f\ \Psi'_1 \dots \Psi'_m$ such that the CI is reduced then apply the rule (if a choice of a variable ordering is involved, choose the one which results in the smaller CI) and recursively apply the algorithm to the coefficients of the obtained CR;
3. else return $f\ \Psi'_1 \dots \Psi'_m$;

5 Examples

We conclude this section by illustrating the suggested heuristic simplification procedures. In Table 3 we give examples which are very similar to those used in [4] to measure and demosntrate the evaluation speed of our implementation of the CR method. Column 1 shows the elementary expressions we used as input and column 2 shows their respective CIs (by extending the definition of the CI for CR expressions in the obvious way to elementary expressions). Column 3 shows the CR expression obtained from the CR construction and heuristic simplification, and column 4 shows their respective CIs. We assumed that we work over a regular grid of $10,000$ or 100×100 evaluation points and that we used the weights obtained for a Sun-SparcStationII of table 2.

We furthermore assumed that expressions of the form x^n were always evaluated using the `pow` operation for elementary expressions and using a repeated squaring procedure for CR expressions.

For row 2 we used the expanded form of $(x-1)^{10}+1$ as input, i.e. $x^{10} - 10x^9 + \binom{10}{2}x^8 \cdots - 10x$ and for row 3 we used the Horner representation of $(x-1)^{10}+1$ as input, i.e. $(\dots((x-10)x + \binom{10}{2})x \dots - 10)x$. Likewise, for rows 5 and 6, we used the expanded input $y^3x^4 - 4y^3x^3 + 3y^2x^4 + \cdots + 6x^2 + 3y - 4x + 2$ and the Horner representation $2 + (-4 + (6 + (-4 + x)x)x)x + (\dots + (\dots + (\dots + (\dots + \dots x)y)y)y)y$.

All CR expressions in column 3 are optimal, i.e., the heuristic simplification algorithm returned the same results as the exhaustive search algorithms. This illustrates that especially for relatively simple examples (like the ones above), the results of the heuristic simplification algorithm are usually as good as those from the exhaustive search algorithm (only much more efficient).

Notice also that the CR method does not result in a decrease of the CI for the example in row 9, i.e., the input expression does not allow CR simplifications.

6 Summary and Discussion

The simplification strategies developed in this paper, considered, for the first time, CR simplifications not independently of the computational context and firstly addressed the problem of variable orderings for multi - dimensional CR expressions. Our simplification strategies were based on the Cost Index (CI) of a CR expression, which we introduced in Section 2 as a system-independent measure for the evaluation cost, and on the analysis of the influence of CR simplifications on the CI of the transformed CR expressions (Section 3).

Our first CR simplification algorithm, given in Section 4, employs an exhaustive search technique to find the CR expression with a minimal CI. While this strategy has the advantage of resulting in a CR expression whose evaluation cost is minimal, the algorithm itself has a (worst case) exponential time and space complexity and is therefore of limited applicability. As an alternative, we furthermore developed a heuristic simplification algorithm, which balances the time spent during simplifications with the time saved in the following evaluations. The suggested heuristics are based on observations about the structure of CR simplification rules and on the ratio of the dimensions of the evaluation domain (for determining the variable ordering of multi - dimensional CRs). The simplifications are accomplish in a reasonable time (i.e., in at most polynomial time) such that the CI of the resulting CR expression is close to the minimum.

The MPCR server – our implementation of the CR method described in [4, 2] – successfully implements the simplification algorithms described here, and the reported timings are a further illustration that our proposed strategies work.

Numerous, often tediously to describe, but more or less obvious refinements of these simplification strategies are possible, especially for the multi - dimensional heuristic simplification algorithms. Furthermore, the simplification strategies should be extended to include rational and factorial simplifications of CR expressions based on the principles given in [9].

7 Acknowledgments

Large portions of this paper are based on Chapter four of the Author's Ph.D. dissertation [2], written under at Kent State University under the direction of Paul S. Wang. The author would like to thank all members of his Ph.D. committee, and in particular Paul S. Wang, for their support of the author's Ph.D. work.

	EE	$CI(EE)$	CR	$CI(CR)$
1	$(x-1)^{10}+1$	$2+w_{\texttt{pow}}$	$\{\varphi_0, +, \varphi_1\}^{10}+1$	$2+5w_*$
2	expand (1)	$10+10w_*+9w_{\texttt{pow}}$	$\{\varphi_0, +, \ldots, +, \varphi_{10}\}$	10
3	Horner (2)	$10+10w_*$	$\{\varphi_0, +, \ldots, +, \varphi_{10}\}$	10
4	$(x-1)^4(y+1)^3+1$	$3+w_*+2w_{\texttt{pow}}$	$\{\varphi_0, +, \varphi_1\}_x^4\{\psi_0, +, \psi_1\}_y^3+3$	$1.02+1.05w_*$
5	expand (3)	$19+28w_*+23w_{\texttt{pow}}$	$\{x_4, x_4, x_4, x_4\}_y$	3.16
6	Horner (3)	$19+19w_*$	$\{x_4, x_4, x_4, x_4\}_y$	3.16
7	$\frac{u^2\cos 2v}{2}$	$w_/ + w_* + w_{\texttt{pow}} + w_{\texttt{cos}}$	$\{\varphi_0, +, \varphi_1, +, \varphi_2\}_u\Re\{\psi_0, *, \psi_1\}$	$.04+1.04w_*$
8	$1.3^{1.2x-1}\sin(1.5x)$	$1+5w_*+w_{\texttt{pow}}+w_{\texttt{sin}}$	$\Re\{\varphi_0, *, \varphi_1\}$	$2+4w_*$
9	$\log(x)+\sqrt{x}$	$3+w_{\texttt{log}}+w_{\texttt{sqrt}}$	$\log\{\varphi_0, +, \varphi_1\}_x+\sqrt{\{\psi_0, *, \psi_1\}_y}$	$3+w_{\texttt{log}}+w_{\texttt{sqrt}}$

Table 3: Examples of simplifications of CR expressions

References

[1] Avitzur, R., Bachmann, O., and Kajler, N. From Honest to Intelligent Plotting. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation - ISSAC'95* (Montreal, Canada, July 1995), ACM Press, pp. 32–41.

[2] Bachmann, O. *Chains of Recurrences.* PhD thesis, Department of Mathematics and Computer Science, Kent State University, 1996.

[3] Bachmann, O. Chains of Recurrences for Functions of Two Variables and their Application to Surface Plotting. In *Human Interaction for Symbolic Computation*, N. Kajler, Ed. Springer-Verlag, 1996. To appear.

[4] Bachmann, O. MPCR: An Efficient and Flexible Chains of Recurrences Server. *ACM SIGSAM Bulletin*, 3 (December 1996).

[5] Bachmann, O., Wang, P. S., and Zima, E. V. Chains of Recurrences - a method to expedite the evaluation of closed-form functions. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation - ISSAC'94* (Oxford, England, United Kingdom, July 1994), ACM Press, pp. 242–249.

[6] Buchberger, B., and Loos, R. Algebraic simplification. In *Computer Algebra Symbolic and Algebraic Computation*, B. Buchberger, G. Collins, R. Loos, and R. Albrecht, Eds., 2nd ed. Springer-Verlag, 1982.

[7] Knuth, D. E. *Seminumerical Algorithms*, vol. 2 of *The Art of Computer Programming.* Reading Mass.:Addison-Wesley, 1981.

[8] Zima, E. V. Automatic construction of system of recurrence relations. *Journal of Computational Mathematics and Mathematical Physics 24*, 6 (1984), 193–197.

[9] Zima, E. V. Simplifications and Optimization Transformations of Chains of Recurrences. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation - ISSAC'95* (Montreal, Canada, July 1995), ACM Press, pp. 42–50.

List of papers published in the Reports on Computer Algebra series

[1] H. Grassmann, G.-M. Greuel, B. Martin, W. Neumann, G. Pfister, W. Pohl, H. Schönemann, and T. Siebert. Standard bases, syzygies and their implementation in singular. 1996.

[2] H. Schönemann. Algorithms in singular. 1996.

[3] R. Stobbe. FACTORY: a C++ class library for multivariate polynomial arithmetic. 1996.

[4] O. Bachmann and H. Schönemann. A Manual for the MPP Dictionary and MPP Library. 1996.

[5] O. Bachmann, S. Gray, and H. Schönemann. MPP: A Framework for Distributed Polynomial Computations. 1996.

[6] G.M. Greuel and G. Pfister. Advances and improvements in the theory of standard bases and szyzygies. 1996.

[7] G.M. Greuel. Description of SINGULAR: A computer algebra system for singularity theory, algebraic geometry, and commutative algebra. 1996.

[8] T. Siebert. On strategies and implementations for computations of free resolutions. September 1996.

[9] B. Reinert. Introducing reduction to polycyclic group rings – a comparison of methods. October 1996.

[10] O. Bachmann, S. Gray, and H. Schönemann. A proposal for syntactic data integration for math protocols. January 1997.

[11] O. Bachmann. Effective simplification of cr expressions. January 1997.

[12] O. Bachmann, S. Gray, and H. Schönemann. MP Prototype Specification. January 1997.

[13] O. Bachmann. MPT – a library for parsing and Manipulating MP Trees. January 1997.

[14] K. Madlener and B. Reinert. Relating rewriting techniques on monoids and rings: Congruences on monoids and ideals in monoid rings. September 1997.

[15] B. Martin and T. Siebert. Splitting Algorithm for vector bundles. September 1997.

[16] K. Madlener and B. Reinert. String Rewriting and Gröbner Bases – A General Approach to Monoid and Group Rings. October 1997.

[17] Thomas Siebert. An algorithm for constructing isomorphisms of modules. January 1998.

[18] O. Bachmann and H. Schönemann. Monomial Representations for Gröbner Bases Computations. January 1998.

[19] B. Reinert, K. Madlener, and T. Mora. A note on nielsen reduction and coset enumeration. February 1998.