On Computational Properties of Chains of Recurrences

Eugene V. Zima *
Department of Computer Science
University of Waterloo
Waterloo, Canada
ezima@scg.uwaterloo.ca

ABSTRACT

Backward and mixed chains of recurrences are introduced. A complete set of chains of recurrences manipulation tools is described. Applications of these tools, related to the safety and numeric stability of chained computations are given.

1. INTRODUCTION

The chains of recurrences (CR) technique to expedite function evaluation over regular intervals was introduced in [12]. Algorithms to construct and interpret linear, two-dimensional and multi-dimensional chains of recurrences have been considered in [2, 3, 4, 9, 12, 14] together with implementations within different computer algebra systems (CAS) and as standalone C and Java libraries. It was shown in [14] that the CR-based representation of expressions is a canonical representation of polynomials and rational functions.

We briefly recall the main idea behind the CR-technique in the univariate case along with an introduction of backward chains of recurrences. Let $F(i)$ be a closed form function which we need to compute for $i = 0, 1, \ldots, n$ (assume that $F(i)$ is defined for all these values of the argument i). The CR-method is based on the conversion of $F(i)$ into a faster scheme $\Phi(i)$, $i = 0, 1, \ldots, n$, which involves chains of recurrences of the form

$$f_j(i) = \begin{cases} \varphi_j, & \text{if } i = 0, \\ f_j(i-1) \odot_{j+1} f_{j+1}(i-1), & \text{if } i > 0, \end{cases} \quad (1)$$

$$j = 0, 1, \ldots, k-1,$$

where $\varphi_0, \ldots, \varphi_{k-1}$ are constant expressions, $\odot_j \in \{+, *\}$, $j = 1, \ldots, k$ and $f_k(i)$ is a closed form function which can either be a constant expression or be defined via other chains. For example,

$$F(i) = \frac{i!(n-i)!}{n!}, i = 0, 1, \ldots, n,$$

can be represented by a CR

$$f_0(i) = \begin{cases} 1, & \text{if } i = 0, \\ f_0(i-1) * f_1(i-1), & \text{if } i > 0, \end{cases}$$

where $f_1(i)$ is an expression with two other CRs as operands (CR-expression) $f_1(i) = g(i)/h(i)$:

$$g(i) = \begin{cases} 1, & \text{if } i = 0, \\ g(i-1) + 1, & \text{if } i > 0, \end{cases}$$

$$h(i) = \begin{cases} n, & \text{if } i = 0, \\ h(i-1) + (-1), & \text{if } i > 0, \end{cases}$$

For chains (1) we use the linear notation

$$f_0(i) = \Phi(i) = \{\varphi_0, \odot_1, \varphi_1, \odot_2, \varphi_2, \ldots \odot_k, f_k\}(i) \quad (2)$$

and recursively write operands of $f_k(i)$ (if it is not constant) in linear form. Therefore, CR for $F(i)$ above can be written as

$$\Phi(i) = \{1, *, \frac{\{1, +, 1\}}{\{n, +, -1\}}\}(i), \quad (3)$$

and $F(i) = \Phi(i), i = 0, 1, \ldots, n$.

It is clear that problems of the form "compute $F(x)$ for $x = a, a + h, a + 2h, \ldots$" (typical for plotting or numeric integration) can be easily reformulated in the form "compute $\tilde{F}(i)$ for $i = 0, 1, 2, \ldots$" after substituting $a + ih$ for x in F. That is why we will keep the simplest formulation of the problem in this paper. We also do not specify (unless it is necessary) the computational domain ($\mathbb{Z}$, $\mathbb{R}$, $\mathbb{C}$, $\mathbb{Z}_p$,...), because all the results hold without loss of generality for any commutative ring.

CR-based evaluation yields algebraically the same result as a straightforward evaluation of the original formula. However, depending on the arithmetic in use, natural concerns about results of evaluation arise.

Exact arithmetic: although $\Phi(i) = F(i), i = 0, \ldots, n$ (and therefore is defined for all values of i), some of the subchains of the CR Φ (i.e. some $f_j(i)$ from (1)) could be undefined at the last several points of evaluation. This can happen because CR (1) is based on forward differences (quotients), which involves at the last evaluation points values of $F(i)$ for $i > n$, and those values (or their quotients) are not guaranteed to be defined in advance.

Floating point arithmetic: since the CR technique is based on the use of previously computed values to compute the next value, any computational error in the previous step will be passed on to the new value in addition to any error

*Supported in part by the Natural Sciences and Engineering Research Council of Canada.

ISSAC 2001, 7/01, Ontario, Canada
©2001 ACM 1-58113-417-7/ 01/ 0007 $5.00

in the current step, i.e. the iterative computations using the CR technique possess a cumulative error effect.

The goal of this paper is to extend CR-technique by chains of recurrences based on backward differences (quotients), also by mixed (forward/backward) chains and to define a complete toolkit of CR-manipulations. Another goal is to demonstrate that the CR-representation is self-contained: since both CR-construction and CR-interpretation (as it will be shown) are based on the same algebraic tools, one does not need to change this representation in order to analyze, for example, the numeric stability of CR-computations. It is enough to use CR simplification properties in order to obtain the result of such analysis.

The section "Tabulating polynomial values" in [10] gives a caution on the accumulation of rounding errors when a table of finite differences (CR) is used for the evaluation of polynomial values. However it does not indicate the accumulative error effect quantitatively. On the other hand, very pessimistic bounds for the accumulated error of CR-based polynomial evaluation are given in [4] and repeated in [6]. These bounds involve the degree of the evaluated polynomial (because of mistakes in reasoning). We will show that the accumulated error does not depend on the degree, which means that the numeric stability of CR-based polynomial evaluation is much better than it was previously thought.

The rest of the paper is organized as follows. In section 2 we introduce backward and mixed chains of recurrences and study their properties related to the safety of computations. Section 3 is devoted to the analysis of numeric stability of chains of recurrences. Section 4 contains concluding remarks.

2. FORWARD, BACKWARD AND MIXED CHAINS AND THEIR PROPERTIES

For chains of recurrences (CR) of the form (1) we use the linear notation (2) and call by *CR-expression* an expression with CRs of the form (1) as elementary operands. The CR $\Phi_r = \{\varphi_r, \odot_{r+1}, \varphi_{r+1}, \ldots \odot_k, f_k\}$, $(0 \le r \le k)$ is called an *r-order subchain* of the CR Φ. The CR (1) of *length* k is called *simple* if f_k is a constant and *pure-sum* or *polynomial* (respectively *pure-product* or *exponential*) if $\odot_j = +, j = 1, \ldots, k$ $(\odot_j = *, j = 1, \ldots, k)$. For example, the function $F(i) = i^3, i = 0, 1, \ldots$ is defined by the pure-sum simple CR $\Phi(i) = \{0, +, 1, +, 6, +, 6\}$ of length 3. When it is convenient we use the notation $\Phi = \{\varphi_0, \odot_1, \Phi_1\}$ instead of (2), dropping also (i) if it does not lead to misunderstandings.

The definition of *backward chains of recurrences (BCRs)* almost exactly repeats the definition of CRs [14]. Given a constant φ_0, a function f_1 defined over $\mathbf{N} \cup \{0\}$, and an operator $\odot$ equal to either $+$ or $*$, we can consider a first-order recurrence

$$f_0(i) = \begin{cases} \varphi_0, & \text{if } i = 0, \\ f_0(i-1) \odot f_1(i), & \text{if } i > 0, \end{cases} \qquad (4)$$

which is called a *Base Backward Recurrence* (BBR) and is denoted by $f_0 = \langle \varphi_0, \odot, f_1 \rangle$ in linear form.

Given constants $\varphi_0, \ldots, \varphi_{k-1}$, a function f_k defined over $\mathbf{N} \cup \{0\}$, and operators $\odot_1, \ldots, \odot_k$ equal to either $+$ or $*$, we recursively define a *Backward Chain of Recurrences (BCR)* as the set of functions $f_0, f_1, \ldots, f_{k-1}, f_k$ connected in such a way, that for $0 \le j < k$

$$f_j(i) = \begin{cases} \varphi_j, & \text{if } i = 0, \\ f_j(i-1) \odot_{j+1} f_{j+1}(i), & \text{if } i > 0. \end{cases} \qquad (5)$$

Further, to denote the BCR (5), we will use the shorthand notation $f_0(i) = \Phi(i) = \langle \varphi_0, \odot_1, \varphi_1, \odot_2, \varphi_2, \ldots \odot_k, f_k \rangle$.
Such terms as *length* of the BCR, *simple* BCR, *pure-sum* (or *pure-product*) BCR, *r-order subchain*, *BCR-expression* and so on, are similar to those for CRs. For example, the function $F(i) = i^3, i = 0, 1, \ldots$ can be defined by the pure-sum simple BCR $\Phi(i) = \langle 0, +, 1, +, -6, +, 6 \rangle$ of length 3. Note, that simple BCR of length 1 defines the same function as simple CR of length 1 when their elements are equal: $\langle \varphi_0, \odot, \varphi_1 \rangle(i) \equiv \{\varphi_0, \odot, \varphi_1\}(i)$.

2.1 Chains manipulation toolkit

The operators $\mathbf{V}$ (*value*) and E (*shift* with respect to i: $E(g(i)) = g(i+1)$) form the basic toolkit for interpretation of CR(BCR)-expressions:

$$\mathbf{V}(\Phi) = \begin{cases} c, & \text{if } \Phi \text{ is a constant expression } c; \\ \varphi_0, & \text{if } \Phi = \{\varphi_0, \odot_1, \Phi_1\} \text{ or } \Phi = \langle \varphi_0, \odot_1, \Phi_1 \rangle; \\ P(\mathbf{V}(\Phi^{(1)}), \ldots, \mathbf{V}(\Phi^{(m)})), \\ \quad \text{if } \Phi = P(\Phi^{(1)}, \ldots, \Phi^{(m)}); \end{cases}$$

$$E(\Phi) = \begin{cases} c, & \text{if } \Phi \text{ is a constant expression } c; \\ \{\varphi_0 \odot_1 \mathbf{V}(\Phi_1), \odot_1, E(\Phi_1)\}, \\ \quad \text{if } \Phi = \{\varphi_0, \odot_1, \Phi_1\}; \\ \langle \varphi_0 \odot_1 \mathbf{V}(E(\Phi_1)), \odot_1, E(\Phi_1) \rangle, \\ \quad \text{if } \Phi = \langle \varphi_0, \odot_1, \Phi_1 \rangle; \\ P(E(\Phi^{(1)}), \ldots, E(\Phi^{(m)})), \\ \quad \text{if } \Phi = P(\Phi^{(1)}, \ldots, \Phi^{(m)}). \end{cases}$$

Note the difference in the shifting of a CR and a BCR: for a BCR Φ the subchain Φ_1 has to be shifted before its value is used to produce the result of $E(\Phi)$. This difference in the shifting is the reason of differences in features of computational schemes based on CRs and BCRs.

Given a closed form function $F(i), i = 0, 1, \ldots, n$, after obtaining its CR(BCR)-representation $\Phi(i)$ the values of $F(i)$ are generated by $\mathbf{V}(E^i(\Phi)), i = 0, 1, \ldots, n$. This leads to the following standard interpretation scheme:

```
Initialize(Φ);
V(Φ) ;
for i:= 1 to n do
  Φ:=E(Φ);
  V(Φ)
od;
```

Here `Initialize(Φ)` initializes the components of Φ. The function $\mathbf{V}(\Phi)$ returns the value of $F(i)$, and $\Phi{:=}E(\Phi)$ updates the CR-expression for the next evaluation point. Observe that the above loop has n iterations instead of $n+1$. This is the nature of CR-computations: the value of $F(0) = \Phi(0)$ is available as the result of CR-construction.

The number of operations to be performed at each step of the above loop is called the *Cost Index (CI)* of a CR-expression Φ and is defined as:

$$CI(\Phi) = \begin{cases} 0, & \text{if } \Phi \text{ is a constant;} \\ k + CI(f_k), & \text{if } \Phi = \{\varphi_0, \odot_1, \varphi_1, \ldots, \odot_k, f_k\} \\ & \text{or } \Phi = \langle \varphi_0, \odot_1, \varphi_1, \ldots, \odot_k, f_k \rangle; \\ q + \sum_{j=1}^{m} CI(\Phi_j), & \text{if } \Phi = P(\Phi_1, \Phi_2, \ldots, \Phi_m). \end{cases}$$

Here q is the number of operations in the expression P. The cost index of a CR-expression Φ gives an indication of its evaluation cost (it counts the number of operations needed to compute $\mathbf{V}(E(\Phi))$). For example, for a simple CR Φ the cost index is equal to the length of the CR. For the CR-expression (3) $CI(\Phi) = 4$.

The application of the *inverse shift* operator E^{-1} ($E^{-1}g(i) = g(i-1)$) to CR-expressions is defined as follows:

$$E^{-1}(\Phi) = \begin{cases} c, & \text{if } \Phi \text{ is a constant expression } c; \\ \{\varphi_0 \odot_1^{-1} \mathbf{V}(E^{-1}(\Phi_1)), \odot_1, E^{-1}(\Phi_1)\}, & \\ \quad \text{if } \Phi = \{\varphi_0, \odot_1, \Phi_1\}; & \\ \langle \varphi_0 \odot_1^{-1} \mathbf{V}(\Phi_1), \odot_1, E^{-1}(\Phi_1)\rangle, & \\ \quad \text{if } \Phi = \langle \varphi_0, \odot_1, \Phi_1 \rangle; & \\ P(E^{-1}(\Phi^{(1)}), \ldots, E^{-1}(\Phi^{(m)})), & \\ \quad \text{if } \Phi = P(\Phi^{(1)}, \ldots, \Phi^{(m)}). \end{cases}$$

Here $+^{-1}$ denotes subtraction and $*^{-1}$ denotes division. It is easy to find a symmetry in the definition of E and E^{-1} for CR and BCR-expressions: an inverse shift acts on a BCR in the same manner as a shift acts on a CR, and vice versa.

2.2 Chains simplification and interpretation

Elementary subexpressions of $F(i)$ are constants or simple pure-sum CRs for variable i: $\{0, +, 1\}$. Conversion of $F(i)$ into CR-expression $\Phi(i)$ is based on algebraic operations (simplifications) defined for CRs. We list some simplifications which will be used further (here c is a constant):

$$S_1 : c + \{\varphi_0, +, \Phi_1\} = \{c + \varphi_0, +, \Phi_1\}$$
$$S_2 : c\{\varphi_0, +, \Phi_1\} = \{c\varphi_0, +, c\Phi_1\}$$
$$S_3 : c\{\varphi_0, *, \Phi_1\} = \{c\varphi_0, *, \Phi_1\}$$
$$S_4 : c^{\{\varphi_0, +, \Phi_1\}} = \{c^{\varphi_0}, *, c^{\Phi_1}\}$$
$$S_5 : \{\varphi_0, +, \Phi_1\} \pm \{\psi_0, +, \Psi_1\} = \{\varphi_0 \pm \psi, +, \Phi_1 \pm \Psi_1\}$$
$$S_6 : \{\varphi_0, *, \Phi_1\} * / \{\psi_0, *, \Psi_1\} = \{\varphi_0 * / \psi_0, *, \Phi_1 * / \Psi_1\}$$
$$S_7 : \{\varphi_0, +, \Phi_1\} * \{\psi_0, +, \Psi_1\} = \{\varphi_0 * \psi_0, +, \Phi\Psi_1 + \Phi_1 E(\Psi)\}$$

Most of the CR-simplification rules from [14] (or from $S_1 - S_7$) turn into BCR-simplification rules by substituting $\langle$ and $\rangle$ in place of $\{$ and $\}$. For example,
$c * \langle \varphi_0, +, \Phi_1 \rangle = \langle c * \varphi_0, +, c * \Phi_1 \rangle$,
$\langle \varphi_0, +, \Phi_1 \rangle \pm \langle \psi_0, +, \Psi_1 \rangle = \langle \varphi_0 \pm \psi_0, +, \Phi_1 \pm \Psi_1 \rangle$,
$c^{\langle \varphi_0, +, \Phi_1 \rangle} = \langle c^{\varphi_0}, *, c^{\Phi_1} \rangle$,
and so on. There are only two rules where CR and BCR construction are different. Only in these rules forward (resp. backward) nature of the CRs (resp. BCR) is apparent. Let $\Phi(i) = \{\varphi_0, +, \Phi_1(i)\}$ and $\Psi(i) = \{\psi_0, +, \Psi_1(i)\}$ be two CRs. Then

$$\Phi(i) * \Psi(i) = \{\varphi_0\psi_0, +, \Phi(i)\Psi_1(i) + \Phi_1(i)E(\Psi(i))\}$$

where E is the shift operator. Now let $\Phi(i) = \langle \varphi_0, +, \Phi_1(i) \rangle$ and $\Psi(i) = \langle \psi_0, +, \Psi_1(i) \rangle$ be two BCRs. Then

$$\Phi(i) * \Psi(i) = \langle \varphi_0\psi_0, +, \Phi(i)\Psi_1(i) + \Phi_1(i)E^{-1}(\Psi(i)) \rangle$$

where E^{-1} is the inverse shift operator. The rules to construct the CR (BCR) for $\{\varphi_0, *, \Phi_1(i)\}^{\{\psi_0, +, \Psi_1(i)\}}$ (resp. for $\langle \varphi_0, *, \Phi_1(i) \rangle^{\langle \psi_0, +, \Psi_1(i) \rangle}$) differ in the same way.

Remark. Observe that when we construct BCRs by means of the rules above we will not encounter problems with applications of E^{-1}, because this operator will be applied only to pure-sum BCRs.

In other parts the construction of BCRs looks exactly as the algorithm `CRmake` described in [14]. Given the closed form function $F(i)$, which we need to compute for $i = 0, 1, \ldots$, this algorithm replaces occurrences of i by BCR $\langle 0, +, 1 \rangle$, applies BCR-simplification rules recursively and returns the BCR-expression $\Phi(i)$ of the same Cost Index as `CRmake` does.

The tight relationship between CR-simplifications and CR-interpretation is best seen for a simple pure-sum CR

$$\Phi(i) = \{\varphi_0, +, \varphi_1, +, \ldots, \varphi_{k-1}, +, \varphi_k\},$$

which defines a polynomial of degree k with the set φ_0, φ_1, $\ldots$, φ_k of finite differences at $i = 0$. Using the definition of E and CR-simplifications we can write

$$\begin{aligned} & E(\Phi(i)) = \\ & = \{\varphi_0 + \varphi_1, +, \varphi_1 + \varphi_2, +, \ldots, \varphi_{k-1} + \varphi_k, +, \varphi_k\} = \\ & = \{\varphi_0, +, \varphi_1, +, \ldots, \varphi_{k-1}, +, \varphi_k\} + \\ & \quad + \{\varphi_1, +, \ldots, \varphi_{k-1}, +, \varphi_k\} = \\ & = \Phi(i) + \Phi_1(i), \end{aligned} \tag{6}$$

where $\Phi_1(i)$ is first-order subchain of $\Phi(i)$. Similarly for a simple pure-sum BCR

$$\Phi(i) = \langle \varphi_0, +, \varphi_1, +, \ldots, \varphi_{k-1}, +, \varphi_k \rangle,$$

which defines a polynomial of degree k with the set φ_0, φ_1, $\ldots$, φ_k of backward finite differences at $i = 0$, using the definition of E and BCR-simplifications we can write

$$\begin{aligned} & E(\Phi(i)) = \\ & = \langle \varphi_0 + \varphi_1 + \ldots + \varphi_k, +, \varphi_1 + \varphi_2 + \ldots + \varphi_k, +, \ldots \\ & \quad \ldots, \varphi_{k-1} + \varphi_k, +, \varphi_k \rangle = \\ & = \langle \varphi_0, +, \varphi_1, +, \ldots, \varphi_{k-1}, +, \varphi_k \rangle + \\ & \quad + \langle \varphi_1, +, \ldots, \varphi_{k-1}, +, \varphi_k \rangle + \ldots \\ & \quad \ldots + \langle \varphi_{k-1}, +, \varphi_k \rangle + \langle \varphi_k \rangle = \\ & = \Phi(i) + \Phi_1(i) + \Phi_2(i) + \ldots + \Phi_{k-1}(i) + \Phi_k(i), \end{aligned} \tag{7}$$

where $\Phi_j(i)$ is jth-order subchain of $\Phi(i)$. This gives a connection between chains interpretation and simplification which will be used in section 3.

2.3 Safety of chained computations

Here we will give an example which shows why the BCR-representation might be preferred to the CR-representation. We first introduce another useful measure of a CR-expression which is called the *effective length* and is defined recursively as follows:

$$\mathtt{el}(\Phi) = \begin{cases} 0, & \text{if } \Phi \text{ is a constant expression } c; \\ 1 + \mathtt{el}(\Phi_1), & \text{if } \Phi = \{\varphi_0, \odot_1, \Phi_1\}; \\ \max(\mathtt{el}(\Phi^{(1)}), \ldots, \mathtt{el}(\Phi^{(m)})), & \\ \quad \text{if } \Phi = P(\Phi^{(1)}, \ldots, \Phi^{(m)}). \end{cases}$$

Informally, $\mathbf{el}(\Phi)$ is the length of the longest path (counted in the number of chain operations $\odot_j$) from the root of CR-expression Φ to the last component of any subchain found in Φ. This value gives two important characteristics of computational scheme based on CR-expression Φ: the size of the potentially unsafe region at the end of computations, and the size of the longest "error-accumulation" chain in the computation of $\Phi(i)$ which will be used for estimating a priori error-bounds in section 3.

Suppose we are to compute values of $F(i) = \sum_{k=0}^{i} f(k)$ for $i = 0, 1, \ldots, n$, where $f(k) = \frac{k!(n-k)!}{n!}$. The values $F(i)$ are defined by a CR:

$$\Phi = \{1, +, \frac{1}{n}, *, \frac{\{2, +, 1\}}{\{n-1, +, -1\}}\}(i).$$

The standard scheme of interpretation will compute a sequence of values $\mathbf{V}(\Phi)$, $\mathbf{V}(E(\Phi))$, $\mathbf{V}(E^2(\Phi)), \ldots$ Note that

$$E^{n-1}(\Phi) = \{\alpha, +, \beta, *, \frac{\{n+1, +, 1\}}{\{0, +, -1\}}\}(i)$$

for some rational constants α and β. Application of the shift operator to $E^{n-1}(\Phi)$ will fail because of division by 0:

$$E^{n}(\Phi) = \{\alpha+\beta, +, \beta \cdot \frac{n+1}{0}, *, \frac{\{n+2, +, 1\}}{\{-1, +, -1\}}\}(i),$$

although $(\alpha + \beta)$ is correct value of $F(n)$.

As we mentioned before, although $F(i)$ is defined for all values of i, some subchains or subexpressions of Φ can be undefined at several last evaluation points. The region $i = 0, 1, \ldots, n - \mathbf{el}(\Phi)$ is absolutely safe for computing the values $F(i)$ using the general CR-interpretation scheme. Computations at points $i = n - \mathbf{el}(\Phi) + 1, \ldots, n$ should be organized in such a way which avoids the evaluation of unnecessary (thus potentially undefined) values. This can be achieved at the stage of program generation simply by splitting computations into two loops: for $i = 0, 1, \ldots, n - \mathbf{el}(\Phi)$ and for $i = n - \mathbf{el}(\Phi) + 1, \ldots, n$, unfolding the second one and getting rid of unnecessary computations. If the CR-representation is being interpreted, the standard interpretation scheme has to undergo similar transformation (it has to consist of two loops, where the second loop implements "careful" interpretation of a CR-expression: dropping all subchains of the order $n - i$ after i-th iteration).

The number of iterations to split can be reduced if instead of $\mathbf{el}(\Phi)$ we use the longest path from the root of CR-expression to potentially "dangerous" operation in CR-representation (such as division, tan, cot or log function calls and so far). In our example it would be sufficient to split just one iteration, although $\mathbf{el}(\Phi) = 3$. In such case we still have some unnecessary computations at the end of the loop, but they do not lead to division by zero.

Now, let us consider BCR-based evaluation of the same values. Values $F(i)$ can be defined by BCR

$$\Psi = \langle 1, +, 1, *, \frac{\langle 0, +, 1\rangle}{\langle n+1, +, -1\rangle}\rangle(i).$$

Standard scheme of interpretation will compute sequence of values $\mathbf{V}(\Psi)$, $\mathbf{V}(E(\Psi))$, $\mathbf{V}(E^2(\Psi)), \ldots$ flawlessly. The explanation of this fact is very simple. Because of symmetries between CRs and BCRs, some subchains or subexpressions of Ψ can be undefined at several first evaluation points. To cure this we would need to split several first iterations from the standard interpretation loop. In this particular example it suffices to split one iteration, but this splitting (of the first iterations) is always done due to the nature of chained computations.

Generally, BCRs allow us not to worry about the end of the evaluation interval: attempts to compute undefined values can be made only at the starting points $i = 0, 1, ..., \mathbf{el}(\Psi) - 1$ of the loop when computing the values of BCR-expression $\Psi(i)$. This is important, because an attempt to compute undefined values will happen now during the BCR-construction (if construction is done numerically). The BCR-construction procedure can trap such event and dynamically split the first step of the interpretation loop generating the loop for $E(\Psi)$ (or $E^2(\Psi)$) instead [1]. This means that at the end of the BCR-construction we will have a safe computational scheme and the BCR-interpreter would not need to take special care about the last evaluation points, which generally makes interpreter simpler and faster.

2.4 Properties of polynomial chains

As it was shown in [3], there exists simple relationship between simple pure-sum CRs and falling factorial powers ([7]) $i^{\underline{j}} = i(i-1)\ldots(i-j+1)$. For BCRs there exists simple relationship between simple pure-sum BCRs and rising factorial powers ([7]) $i^{\overline{j}} = i(i+1)\ldots(i+j-1)$. The following equalities hold

$$i^{\underline{k}} = \{\underbrace{0, +, 0, +, \ldots, 0, +}_{k \text{ times}}, k!\},$$

$$i^{\overline{k}} = \langle\underbrace{0, +, 0, +, \ldots, 0, +}_{k \text{ times}}, k!\rangle.$$

From which we derive using rules S_1-S_7

$$\{\varphi_0, +, \varphi_1, \ldots, +, \varphi_k\} = \varphi_0 + \varphi_1 i^{\underline{1}} + \ldots + \frac{\varphi_k}{k!} i^{\underline{k}} = \\ = \varphi_0 + \varphi_1 \binom{i}{1} + \varphi_2 \binom{i}{2} + \ldots + \varphi_k \binom{i}{k} \quad (8)$$

and

$$\langle\psi_0, +, \psi_1, \ldots, +, \psi_k\rangle = \psi_0 + \psi_1 i^{\overline{1}} + \ldots + \frac{\psi_k}{k!} i^{\overline{k}} = \\ = \psi_0 + \psi_1 \binom{i}{1} + \psi_2 \binom{i+1}{2} + \ldots + \psi_k \binom{i+k-1}{k}. \quad (9)$$

The last equation means that a simple pure-sum BCR of length k is a polynomial of degree k in i.

From (8) and $\sum_{j=0}^{i-1} j^{\underline{k}} = \frac{1}{k+1} i^{\underline{k+1}}$ we obtain

$$\sum_{j=0}^{i-1} \{\varphi_0, +, \varphi_1, \ldots, +, \varphi_k\}(j) = \\ = \{0, +, \varphi_0, +, \varphi_1, \ldots, +, \varphi_k\}(i) \quad (10)$$

or $\sum_{j=0}^{i-1} \Phi(j) = \{0, +, \Phi\}(i)$ for simple pure-sum CR Φ.

[1] more precisely, for $E^m(\Psi)$, where $m < \mathbf{el}(\Psi)$ is the distance in the number of chained operations from the root of BCR expression to the subexpression which caused the event.

Similarly, from (9) and $\sum_{j=0}^{i-1} j^{\overline{k}} = \frac{1}{k+1} E^{-1}(i^{\overline{k+1}})$ we obtain

$$\sum_{j=0}^{i-1} \langle \varphi_0, +, \varphi_1, \ldots, +, \varphi_k \rangle(j) = \\ = E^{-1}(\langle 0, +, \varphi_0, +, \varphi_1, \ldots, +, \varphi_k \rangle(i)) \tag{11}$$

or $\sum_{j=0}^{i-1} \Phi(j) = E^{-1}(\langle 0, +, \Phi \rangle(i))$ for simple pure-sum BCR Φ. These formulae are respectively CR and BCR analogs of summation formula for polynomials.

Remark. The relationship between pure-sum CRs and BCRs is even tighter. Consider, for example, the function i^k which is defined by a CR $\{\varphi_0, +, \varphi_1, \ldots, \varphi_{k-1}, +, \varphi_k\}$. Writing the same function as a BCR gives

$$\langle (-1)^k \varphi_0, +, (-1)^{k-1} \varphi_1, \ldots, (-1)^1 \varphi_{k-1}, +, \varphi_k \rangle.$$

This follows from the above equations and from the following conversion formulae between powers (see [7]):

$$i^k = \sum_{j=0}^{k} \left\{ {k \atop j} \right\} i^{\underline{j}} = \sum_{j=0}^{k} (-1)^{k-j} \left\{ {k \atop j} \right\} i^{\overline{j}},$$

where $\left\{ {k \atop j} \right\}, j = 0, \ldots, k$ are the second kind Stirling numbers.

With the help of equations (8),(9) we define two special polynomials which will be used in section 3:

$$\mathcal{A}_k(i) = \{\underbrace{1, +, 1, +, \ldots, 1, +}_{k \text{ times}}, 1\}(i) \tag{12}$$

and

$$\mathcal{B}_k(i) = \langle \underbrace{1, +, 1, +, \ldots, 1, +}_{k \text{ times}}, 1 \rangle(i). \tag{13}$$

Observe, that $\mathcal{A}_k(i) \leq \mathcal{B}_k(i)$ for all natural k and i and that

$$\mathcal{A}_k(i) < \mathcal{B}_k(i) \tag{14}$$

for all $k \geq 2, i > 1$. Furthermore, $lc(\mathcal{A}_k(i)) = lc(\mathcal{B}_k(i))$, therefore for any fixed k

$$\lim_{i \to \infty} \frac{\mathcal{B}_k(i)}{\mathcal{A}_k(i)} = 1. \tag{15}$$

2.5 Loop optimization tools and mixed chains

The operators **V**, E and E^{-1} and the CR/BCR simplification rules form a toolkit for loop optimization [6, 13]. Consider the following loop:

```
x := g(0);
for i := 1 to n do
    y := f(x); ... x := g(i); ... z := h(x); ...
od
```

After constructing a CR $\Phi(i) = g(i)$ one can try to use the chained nature of the values of x, substitute $\Phi(i)$ in place of x into $f(x)$ and $h(x)$, and obtain the CR representation for these expressions with the help of CR-simplifications. In order to preserve the semantics of the loop one has to substitute $\Phi(i)$ into $h(x)$ and $E^{-1}(\Phi(i))$ into $f(x)$. The reasonable question about the existence of $E^{-1}(\Phi(i))$ arises here. It is easy to see for example that for Φ defined in (3) $E^{-1}(\Phi(i))$ is not defined at $i = 0$ because of division by zero. This does not mean that the optimization connected with substitution can not be performed for $g(i) = \Phi(i)$ from (3). It is still possible by simply splitting the first step of the loop, and applying substitution to the body of the shortened loop. This will be valid, because the loop without the first step computes values of $E(\Phi(i))$ and now $E^{-1}(E(\Phi(i))) = \Phi(i)$ is defined. Such a splitting is not necessary in the cases when $g(i)$ is defined by a simple CR $\Phi(i)$. It is easy to show [13] that in such a case either $E^{-1}(\Phi(i))$ is defined, or the loop degenerates (computes $x = 0$ for most of the values of i).

Sometimes the loop already implements some chained evaluations rules, and they can have either forward or backward nature. In order to handle such situations properly we need CR/BCR conversion tools, which are defined as follows

$$\texttt{CRtoBCR}(\Phi) = \begin{cases} c, \text{ if } \Phi \text{ is a constant expression } c; \\ \langle \varphi_0, \odot_1, \texttt{CRtoBCR}(E^{-1}(\Phi_1)) \rangle, \\ \quad \text{if } \Phi = \{\varphi_0, \odot_1, \Phi_1\}; \\ P(\texttt{CRtoBCR}(\Phi^{(1)}), \ldots, \texttt{CRtoBCR}(\Phi^{(m)})), \\ \quad \text{if } \Phi = P(\Phi^{(1)}, \ldots, \Phi^{(m)}). \end{cases}$$

and

$$\texttt{BCRtoCR}(\Phi) = \begin{cases} c, \text{ if } \Phi \text{ is a constant expression } c; \\ \{\varphi_0, \odot_1, \texttt{BCRtoCR}(E(\Phi_1))\}, \\ \quad \text{if } \Phi = \langle \varphi_0, \odot_1, \Phi_1 \rangle; \\ P(\texttt{BCRtoCR}(\Phi^{(1)}), \ldots, \texttt{BCRtoCR}(\Phi^{(m)})), \\ \quad \text{if } \Phi = P(\Phi^{(1)}, \ldots, \Phi^{(m)}). \end{cases}$$

During the CR-based loop optimization we have different choices:
1) convert all computations to forward chains;
2) convert all computations to backward chains;
3) preserve as much of the loop semantics (in particular chains already existing within a loop) as possible.
The first two goals are achieved with the help of conversion tools, substitutions and CR/BCR simplifications. The third goal leads to the consideration of mixed chains of the form $\{\psi_0, *, \psi_1, +, \langle \psi_2, +, \{\psi_3, +, \ldots, +, \psi_k\}\}\}$, with different orders of $\langle,\rangle$ and $\{,\}$ brackets. Computations based on mixed chains of recurrences have the same general scheme of interpretation. Moreover, the operators **V**, E and E^{-1} were defined to allow mixed chains as input. Partial conversion might also be needed during mixed chains simplification, for example for simplification of an expression of the form $\{\varphi_0, +, \Phi_1\} \pm \langle \psi_0, +, \Psi_1 \rangle$. Consider for example the following loop:

```
x := 0; t1 := 0; y := 0;
for i := 1 to n do
  x := x + 2;                                   ⟨0, +, 2⟩
  t1 := t1 + x;                                 ⟨0, +, 0, +, 2⟩
  t2 := 4·t1·(4·t1 + x) − x^4;  ⟨0, +, 16, +, −192, +, 240⟩
  y := y + t2;                  ⟨0, +, 0, +, 16, +, −192, +, 240⟩
od
```

This loop is annotated by BCRs which were constructed for x and then by substitutions and simplifications for other variables. If the only variable used after the loop is y, we would have optimizied loop performing only 4 additions at each iteration. We constructed BCRs because the first two assignments in the loop already implement a BCR-like computations. If the order of these two assignments was different, we would end up with mixed CR/BCR scheme, implementing evaluation of $t2$ with a CR and evaluation of y with a mixed chain.

Remark. The need for mixed computations can also appear

because BCRs have some disadvantages in comparison with CRs with regard to numeric stability. A skillful choice of the mixed scheme could keep advantages of CR-representation with regard to stability and advantages of BCR-representation with regard to "avoiding unsafe computational steps." Returning to the example in section 2.3 we can define $F(i)$ by the mixed chain

$$\Delta = \langle 1, +, \{1, *, \frac{\{1, +, 1\}}{\{n, +, -1\}}\}\rangle(i),$$

which gives a safe scheme of computations, and most of computations are based on forward chains.

3. STABILITY OF CR-COMPUTATIONS

Since the CR-technique is based on the use of previously computed values to compute the next value, any computational numeric error in the previous steps will be passed on to the new value in addition to any error in the current step. In general, this error is within reasonable limits, but for a large number of iterations, the error can become significant. In this section we first analyze the propagated error and then show an approach to improve the error characteristics of CR computations.

3.1 Error analysis for simple CRs

The errors in function evaluation using CRs could arise due to several sources:
1) representation error (error caused by loss of information at the time of initialization due to the finite word size);
2) CR-computation error (floating point roundoff error that occurs when we compute the successive values of the simple CRs contained in CR-expression $\Phi(i)$);
3) "usual" error (the error of evaluation of the operations from initial expression $F(i)$ which remain in the CR-expression obtained by $F(i)$) and so on.
We will be interested mostly in the characterization of the relative errors and will use the standard model [8], assuming that representation error of exact quantity v is defined by

$$\mathrm{fl}(v) = v(1+\delta), |\delta| \le u,$$

and floating point operations roundoff error is defined by

$$\mathrm{fl}(t \odot v) = (t \odot v)(1+\delta), |\delta| \le u, \odot \in \{+, -, *, /\},$$

where u is a hardware dependent unit of roundoff. We also assume in this section that $i\delta \ll 1$ and $k\delta \ll 1$.

We first show how representation error influences the CR-computation error under the assumption that arithmetic is exact. Consider a simple pure-sum CR

$$\Phi(i) = \{\varphi_0, +, \varphi_1, +, \ldots, +, \varphi_k\}(i)$$

and the approximate simple pure-sum CR

$$\tilde{\Phi}(i) = \{\tilde{\varphi}_0, +, \tilde{\varphi}_1, +, \ldots, +, \tilde{\varphi}_k\}(i),$$

obtained after floating point initialization of Φ. Using the model mentioned above and CR-simplification rules from section 2 we can write

$$\begin{aligned}&\tilde{\Phi}(i) = \\ &= \{\varphi_0(1+\delta), +, \varphi_1(1+\delta), +, \ldots, +, \varphi_k(1+\delta)\}(i) = \\ &= (1+\delta)\{\varphi_0, +, \varphi_1, +, \ldots, +, \varphi_k\}(i) = (1+\delta)\Phi(i),\end{aligned}$$

or $\tilde{\Phi}(i) - \Phi(i) = \delta\Phi(i)$.

The last equality suggests that if arithmetic is exact then pure-sum CR evaluation is numerically stable with respect to small perturbations in values of elements φ_j of a CR.

Consider now a simple pure-product CR

$$\Phi(i) = \{\varphi_0, *, \varphi_1, *, \ldots, *, \varphi_k\}(i)$$

and the approximate simple pure-product CR

$$\tilde{\Phi}(i) = \{\tilde{\varphi}_0, *, \tilde{\varphi}_1, *, \ldots, *, \tilde{\varphi}_k\}(i),$$

obtained after initialization of Φ. With previous assumptions and $\varphi_j > 0$ we get $\tilde{\varphi}_j = (1+\delta)\varphi_j, j = 0, 1, \ldots k$. Using CR-simplification rules we can rewrite

$$\begin{aligned}\tilde{\Phi}(i) - \Phi(i) &= \{\underbrace{1+\delta, *, \ldots, 1+\delta, *}_{k \text{ times}}, 1+\delta\}\Phi(i) - \Phi(i) = \\ &= ((1+\delta)^{\mathcal{A}_k(i)} - 1)\Phi(i).\end{aligned}$$

Since $\delta \ll 1$ we deduce $(1+\delta)^{\mathcal{A}_k(i)} \simeq 1 + \delta\mathcal{A}_k(i)$ and finally get $\tilde{\Phi} - \Phi \simeq \delta\mathcal{A}_k(i)\Phi(i)$.

This means that pure-product CR evaluation is unstable, since relative error grows proportionally to the values of polynomial $\mathcal{A}_k(i)$ when i increases.

Now we describe the influence of roundoff error at every step of CR-computations. Let Φ be an exact pure-sum CR and $\tilde{E}$ denotes the shift in floating point environment. Then, using (6) twice we get

$$\tilde{E}(\Phi) = \mathrm{fl}(\Phi + \Phi_1) = (\Phi + \Phi_1)(1+\delta) = E(\Phi)(1+\delta).$$

Evaluation of a CR Φ over consecutive i points corresponds to the evaluation of $\tilde{E}^i\Phi$. Noting that $\tilde{E}$ and E commute under the standard floating point model, we have

$$\begin{aligned}\tilde{E}^i(\Phi) &= \tilde{E}^{i-1}(E(\Phi)(1+\delta)) = (1+\delta)E(\tilde{E}^{i-1}(\Phi)) = \ldots \\ &\ldots = (1+\delta)^i E^i(\Phi).\end{aligned}$$

From here we get

$$\tilde{E}^i(\Phi) - E^i(\Phi) \simeq i\delta E^i(\Phi), \qquad (16)$$

which means that relative error grows linearly in the number of evaluation points i. This is more optimistic than [4] which suggests that the relative error accumulates proportionally to $i \cdot k \cdot \delta$ for polynomial CR of length k. Formula (16) can also be used to estimate absolute error behavior in case when $\tilde{E}^i(\Phi)$ is close to 0 (i.e. when relative error analysis can not be used).

Observe that the analysis above estimates worst case accumulation when all roundings are done in the same direction. In real life floating point environment the accumulated error grows much slower. It is interesting to note also that the accumulated error for high degree polynomials can be less than that for low degree polynomials. Figure 1 shows the result of an experimental evaluation of two polynomials over 10000 points: $f(x) = x^3/11$ and $g(x) = x^7/110000$ for $x = 0.0, 0.001, \ldots, 10.0$ (note that $f(10) = g(10)$). Computation was performed in Maple's hardware floating point environment and the figure shows the accumulated relative

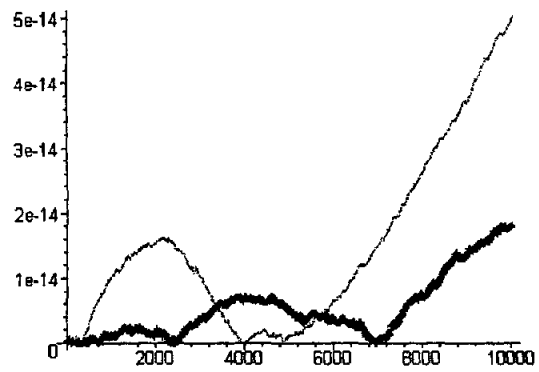

Figure 1: Absolute value of relative error over 10000 evaluation points for $f(x)$ (light) and $g(x)$ (dark).

error for both polynomials against the number of the evaluation point. Although both curves are under the line $\delta \cdot i$, $g(x)$ has a smaller accumulated error at the end point.

This somewhat surprising fact has a very simple explanation. In real life floating point environment rounding is done in different directions and accumulated errors in longer chains have better chances for mutual cancellation than in shorter chains.

Now consider a pure-product CR Φ of length k. Then

$$\tilde{E}(\Phi) = \mathrm{fl}(\Phi * \Phi_1) = (\Phi * \Phi_1)(1+\delta)^{\mathcal{A}_k(i)} = E(\Phi)(1+\delta)^{\mathcal{A}_k(i)}.$$

After i steps of evaluation

$$\begin{aligned}\tilde{E}^i(\Phi) &= \tilde{E}^{i-1}(E(\Phi)(1+\delta)^{\mathcal{A}_k(i)}) = \\ &= (1+\delta)^{\mathcal{A}_k(i)} E(\tilde{E}^{i-1}(\Phi)) = \dots \\ &\dots = (1+\delta)^{\sum_{j=1}^{i} \mathcal{A}_k(j)} \cdot E^i(\Phi)\end{aligned}$$

and (since $(1+\delta)^{\sum_{j=1}^{i} \mathcal{A}_k(j)} \simeq 1 + \delta \sum_{j=1}^{i} \mathcal{A}_k(j)$ for $\delta \ll 1$)

$$\tilde{E}^i(\Phi) - E^i(\Phi) \simeq \mathcal{C}_k(i)\delta E^i(\Phi),$$

where $\mathcal{C}_k(i) = \sum_{j=1}^{i} \mathcal{A}_k(j) = \{0, +, \underbrace{1, +, \dots, 1, +}_{k \text{ times}}, 1\}$ is a polynomial of degree $k+1$ in i. The function $\mathcal{C}_k(i)$ describes the cumulative error effect in pure-product chains.

We summarize this section by a description of the "express" method of getting a priori worst case bounds for accumulation of relative error for CR-expressions. Given a CR-expression $\Phi(i)$ we define the *error indicator* to be

$$\mathcal{I}_\Phi(i) = \begin{cases} i, & \text{if } \Phi \text{ has no pure-product subchains,} \\ \mathcal{C}_{\mathbf{el}(\Phi)}(i), & \text{if } \Phi \text{ has pure-product subchains.} \end{cases} \quad (17)$$

The value $\delta\mathcal{I}_\Phi(i)$ roughly describes the accumulation of the relative error after i steps.

All the reasonings of this subsection can be repeated for exponential BCRs by substitution $\mathcal{B}_k$ from (13) in place of $\mathcal{A}_k$. Obvious disadvantages of BCRs in comparison with CRs here follow immediately from (14). But due to (15) the difference in accumulated error in exponential CRs and BCRs for the same expressions becomes less significant as i grows. For polynomial BCRs this difference is even less significant. Similarly to the way we obtained (16) it is possible to show that for a polynomial BCR Ψ of length k

$$\tilde{E}(\Psi) \simeq E(\Psi) + \delta E^2(\Psi)$$

and

$$\frac{\tilde{E}^i(\Psi) - E^i(\Psi)}{E^i(\Psi)} \simeq \delta(i + \frac{iE^i(\Psi_1)}{E^i(\Psi)}).$$

Here $iE^i(\Psi_1)$ is a polynomial of the degree k with the leading coefficient k times larger than the leading coefficient of $E^i(\Psi)$. If computed values are not close to 0 (i.e., relative error analysis is relevant) the last fraction above is bounded by a constant on the evaluation interval. This means that the relative error accumulation for a polynomial BCR will be very close to the one for CR. Running BCR evaluation for $f(x)$ and $g(x)$ shown in Figure 1 gives the same result as CR evaluation.

3.2 Improvement of the error characteristic of the CRs

As shown in the previous subsection, for a large number of iterations the cumulative error effect of the CR-computations can become significant. It is not surprising, since we trade off accuracy for efficiency when we pass from the initial computational scheme $F(i)$ to the CR-expression $\Phi(i)$. It looks quite reasonable to pay a little bit back, i.e. to lose some efficiency in order to get more accurate computational scheme. This can be rectified by "refreshing" the CR, i.e., by reinitializing the value of components periodically.

If the refreshing is done over the regular number of points, we find that this is analogous to a well known program optimizing transformation, called "loop unrolling" [1]. In the general case we will use multidimensional loop-unrolling [9], but here we start with a two-dimensional one as an example.

Given $F(i)$ which has to be evaluated for $i = 0, \dots, n$, assuming that $n+1 = m \cdot q$, we can compute the required values using inner unrolling

$$F(j \cdot q + l - 1), j = 0, \dots, m-1; l = 1, \dots, q;$$

or outer unrolling

$$F((l-1) \cdot m + j), j = 0, \dots, m-1; l = 1, \dots, q.$$

In both cases we can construct a two-dimensional CRs. It is easy to see that while in linear CR technique the error is accumulated through $n+1 = m \cdot q$ steps of computation, after unrolling it is accumulated through no more than $m+q$ steps.

Several features can be noted here:
1. If linear CRs are constructed symbolically the main part of the work to obtain two-dimensional CRs can be done by means of substitutions. It follows from the fact ([9, 14]) that components of CRs (as expressions) are constructed almost from the same set of constants, operation-signs and variables as initial expression $F(i)$. Here we can essentially use the remember tables feature of the **Maple** system [11].
2. The efficiency of the computations in two-dimensional

unrolled loop will decrease by a factor of two at most. Indeed, let the CR $\Phi(i)$ have a Cost Index k. Then the outer CR $\Phi(j)$ will have the same Cost Index and its components (as CRs w.r.t. variable l) will have Cost Index not greater than k. The total complexity for linear CR-computations will be equal to $t_1 = (n+1)k = mq \cdot k$. The total complexity for two-dimensional CR-computations will be not greater than $t_2 = mq \cdot k + m \cdot k^2$ (here $mq \cdot k$ is the complexity of computations in the inner loop, and $m \cdot k^2$ bounds the complexity of computations in the outer loop). In most practical cases we can reasonably assume that $m \simeq q \simeq \sqrt{n+1}$ and $k < \sqrt{n+1}$. Thus, we get $t_2 < 2t_1$.
3. Function evaluation in loops using recurrence relations is inherently parallelizable. This transformation allows us to exploit both parallelization and CR-based improvement of the code generated [5].

In the general case we assume that we are provided with the closed form function $F(i), i = 0, 1, \ldots, n$, the value of initialization error δ and the user-predefined bound ϵ of accumulated relative error (it is assumed that $\delta < \epsilon$). After constructing the CR-expression $\Phi(i)$, the function $\mathcal{I}_\Phi(i)$ from (17) simulates cumulative error effect. We can find m such that $\mathcal{I}_\Phi(m) \leq \frac{\epsilon}{\delta}, \mathcal{I}_\Phi(m+1) > \frac{\epsilon}{\delta}$. If such an m does not exist, there is no point in using CR-technique for given $F(i)$. Otherwise, the value m provides us with the number of steps over which we can use CR, while remaining within the user predefined error bounds. If $m \geq n$ we can use the linear CR-scheme. Otherwise we have to find the minimal natural p, such that $\left\lfloor \frac{m}{p} \right\rfloor^p \geq n$ and use p-dimensional loop unrolling of $\Phi(i)$. The obtained p-dimensional CR will be about p times slower than the linear CR, but will accumulate error not more than over m steps. It is worth to mention that in applications such as plotting 2 or 3 dimensional unrolling typically suffice.

4. CONCLUSION

The method to estimate the worst-case CR-caused accumulated error described above can be useful when the CR-technique is applied to expedite numeric computations "on the flight" (i.e. in the interpretation mode). If we use this technique to generate numerical programs, it is possible to employ other (more careful) techniques to analyze an accumulated error. For example, there are no restrictions for combining the above method with the run-time analysis approach considered in [8].

As mentioned above, in plotting applications the error accumulation effect is not very dramatic, because of the modest number of evaluation points. The live demo of the Java CR engine is available from
`http://scg1.uwaterloo.ca/JMCR.html`.
It is possible to run and compare times for different computational tasks with the SurfacePlotter interpreter and Java CR engine. As well, the SurfacePlotter with its computational part replaced with JMCR can be run to compare the plotting quality. Our C implementation [9] is recently ported to Maple 6 [11] using new Maple *external call* facilities. This expedites our earlier internal Maple implementation by an order of magnitude. Relevant Maple wrappers and dynamic link library are available from the author by request.

5. ACKNOWLEDGEMENT

The author would like to thank Howard Cheng (University of Waterloo) for his help in the preparation of this paper.

6. REFERENCES

[1] AHO, A., AND ULLMAN, J. *The Theory of Parsing, Translation and Compiling, Vol.2.* Englewood Cliffs, N.J.: Prentice-Hall, 1972.

[2] AVITZUR, R., BACHMANN, O., AND KAJLER, N. From Honest to Intelligent Plotting. In *International Symposium on Symbolic and Algebraic Computation* (1995), Montreal, Canada, ACM Press, pp. 32–41.

[3] BACHMANN, O., WANG, P.S., AND ZIMA, E.V. Chains of Recurrences – a method to expedite the evaluation of closed-form functions. In *International Symposium on Symbolic and Algebraic Computation* (1994), Oxford, UK, ACM Press, pp. 242-249.

[4] BACHMANN, O. Chains of recurrences. PhD thesis, Kent State Univ., Kent, OH - 44240, USA, December 1996.

[5] CASAVANT, T., VADIVELU, K., AND ZIMA, E. Mapping Techniques for Parallel Evaluation of Chains of Recurrences. In *International Parallel Processing Symposium* (1996), pp. 620–624.

[6] VAN ENGELEN, R. Symbolic Evaluation of Chains of Recurrences for Loop Optimization. Technical Report TR-000102, Department of Computer Science, Florida State University, 2000.

[7] GRAHAM, R., KNUTH, D., AND PATASHNIK, O. *Concrete Mathematics.* Addison-Wesley, 1994.

[8] HIGHAM, N.J. *Accuracy and stability of numerical algorithms.* SIAM, 1996.

[9] KISLENKOV, V., MITROFANOV, V., AND ZIMA, E.V. Multidimensional of Chains of Recurrences. In *International Symposium on Symbolic and Algebraic Computation* (1998), Rostock, Germany, pp. 199–206.

[10] KNUTH, D.E. *The art of computer programming. V.2. Seminumerical Algorithms.* Second edition. Addison-Wesley, 1981.

[11] MONAGAN, M.B., GEDDES, K.O., HEAL, K., LABAHN, G., VORKETTER, S., AND MCCARRON, J. *The Maple Programming Guide,* Springer-Verlag, 2000.

[12] ZIMA, E.V. Automatic Construction of Systems of Recurrence Relations. *USSR Comput. Maths. Math. Phys.*, Vol.24, N 6, 1984, pp. 193–197.

[13] ZIMA, E.V. System of Recurrence Relations and Loops Optimization. PhD thesis, Moscow State University, 1985.

[14] ZIMA, E.V. Simplification and Optimization Transformations of Chains of Recurrences. In *International Symposium on Symbolic and Algebraic Computation* (1995), Montreal, Canada, ACM Press, pp. 42–50.