Simplification and Optimization Transformations of Chains of Recurrences *

Eugene V. Zima
Department of Computational Mathematics and Cybernetics
Moscow State University, Moscow, 119899, Russia
zima@cs.msu.su

Abstract

The problem of expediting the evaluation of closed-form functions at regular intervals is considered. The Chain of Recurrences technique to expedite computations is extended by rational simplifications and examined as a form of internal representation, oriented towards fast evaluation. Optimizing transformations of Chains of Recurrences are proposed.

1 Introduction

A common component in the analysis and solution of many problems, is the iterative evaluation of a function $G(x)$ over a number of points in an interval. More specifically, given a starting point x_0 and an increment h, evaluation of the function $G(x_0 + ih)$ for $i = 0, 1, \ldots, n$ occurs frequently in applications such as plotting graphs of functions, simulations, and signal processing applications. Straightforward evaluation of functions (especially obtained as the result of symbolic transformations in Computer Algebra Systems) may not be efficient. One way to speed up this process is to compute the function incrementally, i.e., use the results of one iteration in calculating the value of the function in the next iteration. The basic idea behind the technique discussed is algebraically converting the given function to a Chain of Recurrences (CR) that defines the same computation. For example, to compute the values

$$G(x) = \exp(0.2x^2 - 2x - 1)$$

for $x = 0.1 * i, i = 0, 1, \ldots, n$ we can construct the chain of recurrences:

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

$$f_1(x) = \begin{cases} \exp(-0.198), & i = 0 \\ f_1(i-1) * \exp(0.004), & i > 0, \end{cases}$$

such that $G(0.1i) = f_0(i)$ for $i = 0, 1, \ldots$. There will be only two multiplications performed at each step of the loop which computes $f_0(i)$ values:

```
f0:=exp(-1);f1:=exp(-0.198);f2:=exp(0.004);write(f0);
for i :=1 to n do
  begin  f0:=f0*f1; f1:=f1*f2; write(f0)  end;
```

This approach has been shown to provide quite impressive reductions in computation time [1, 2, 3]. This paper extends the set of rules to construct Chains of Recurrences, described in [3, 4] and introduces optimizing transformations of the CR-representation.

One of the main features of the CR-technique is "unconditional" (in some general sense) use of the transfer from the original expression to the CR-representation. With polynomials, for example, the CR-scheme is faster than direct (or Horner) evaluation, independently of the relative speed of addition and multiplication operations. We will try to keep this feature for new simplifying rules and optimizing transformations. We do not consider optimizing algorithms which need knowledge about the relative speed of operations, and algorithms which require backtracking. The advantages of the considered algorithms are their relative simplicity and low-level of time consumption.

In Section 2, we recall the base definitions of the CR-technique and define the CR-interpreting scheme explicitly. Section 3 extends the set of simplifying rules by rational simplifications. In Section 4, optimizing transformations of the CR-expressions are considered. Effective algorithms to find common subchains and "equal-with-delay-subchains" are given. In some sense these algorithms are analogous to the optimizing algorithms used to find common subexpressions, but they appear much simpler because they exploit the linear nature of CR-evaluation.

2 CR-approach to the construction of internal representations, oriented towards fast iterative evaluation

In this section, after recalling main points of the CR-technique, we describe the CR-interpreting scheme explicitly.

2.1 Preliminary definitions and examples

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

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

*Work reported herein was supported in part by the Russian Fund for Fundamental Research under Grant 95-01-01138a.

ISSAC'95 - 7/95 Montreal, Canada

which defines over N∪{0} the function $f_0(i)$: $f_0(i) = \varphi_0 + \sum_{j=0}^{i-1} f_1(j)$, if $\odot = +$, or $f_0(i) = \varphi_0 \prod_{j=0}^{i-1} f_1(j)$, if $\odot = *$. Further we will call (1) a *Base Recurrence* (BR) and use shorthand $f_0 = \{\varphi_0, \odot, f_1\}$ to denote it.

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

$$f_j(i) = \begin{cases} \varphi_j, & \text{if } i = 0, \\ f_j(i-1) \odot_{j+1} f_{j+1}(i-1), & \text{if } i > 0. \end{cases} \quad (2)$$

Further, to denote CRs (2), we will use the shorthand

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

and call

- $k = L(\Phi)$ the *length* of Φ,
- Φ a *pure-sum* CR, if $\odot_1 = \odot_2 = \ldots = \odot_k = +$,
- Φ a *pure-product* CR, if $\odot_1 = \odot_2 = \ldots = \odot_k = *$,
- Φ a *simple* CR, if f_k is a constant,
- $\mathtt{Comp}(j, \Phi) = \varphi_j, (0 \le j \le k)$ the *j-th component* of a simple CR Φ,
- $lc(\Phi) = \mathtt{Comp}(k, \Phi)$ - *last component* of a simple CR Φ,
- $\Phi_r = \{\varphi_r, \odot_{r+1}, \varphi_{r+1}, \ldots \odot_k, f_k\}$, $(0 \le r \le k)$ - *r-order subchain* of the CR Φ [1].

The CR for $G(x)$ from Introduction can also be rewritten as

$$\{\exp(-1), *, \exp(-0.198), *, \exp(0.004)\}(i).$$

It is a pure-product, simple CR of the length 2.

We will generally denote BRs by lowercase letters (like f and g), CRs by uppercase Greek letters (like Φ and Ψ), components of CRs by indexed lowercase Greek letters (like φ_0 and ψ_1), and subchains by indexed uppercase Greek letters. The shorthand notations $\Phi(i) = \Phi$ and $\varphi_i(x_0, h) = \varphi_i$ will also be used. We also replace the notation

$$\{\varphi_0, \odot_1, \varphi_1, \odot_2, \varphi_2, \ldots \odot_k, f_k\} \text{ by}$$

$$\{\varphi_0, \odot_1, f_1\} \text{ ,where } f_1 = \{\varphi_1, \odot_2, \varphi_2, \ldots \odot_k, f_k\}$$

when needed.

Let $G(x)$ be a polynomial in x of degree n:

$$G(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0.$$

Then there exists a n-length simple pure-sum CR

$$\Phi = \{\varphi_0, +, \varphi_1, +, \varphi_2, \ldots, +, \varphi_n\},$$

such that [3, 4] $\Phi(i) = G(x_0 + i * h)$ for $i = 0, 1, \ldots$. In fact, the sequence $\varphi_0, \ldots, \varphi_n$ is nothing more than the table of finite differences of $G(x_0 + i * h)$ taken at the point $i = 0$. It is possible to write out explicit formulae, that connect $G(x)$ coefficients $(a_n, \ldots, a_0)$ and components of Φ ($\varphi_n, \ldots, \varphi_0$) [3]. Further, we will only use the relation between the leading coefficient of polynomials and the last component of correspondent CRs:

$$\varphi_n = a_n n! h^n. \quad (3)$$

For a given $G(x), x_0$ and h, we are interested in constructing a CR Φ such that $\Phi(i) = G(x_0 + i * h)$. A general algorithm to construct CRs for a given function $G(x)$ was considered in [3, 4]. This algorithm can be applied to any function. Instead of finding recurrences for only a particular class of functions, it automatically generates recurrence representations for a wide variety of common functions in order to obtain more efficient computational procedures for their evaluation. The algorithm is based on:

- replacing the trivial subexpression x by the base recurrence $\{x_0, +, h\}$ on the parse tree of G, and
- application of operations from the expression $G(x)$ to recurrences already obtained during end-order traversal of the parse tree, in order to construct CRs which represent larger subexpressions of the given original expression[2].

The algorithm to construct CRs has four parameters: initial expression, variable, starting value of this variable and step (we'll refer to this algorithm as $\mathtt{CRmake}(G(x), x, x_0, h)$). Note that x_0 and h can be any symbolic expressions. The result of applying the algorithm to a closed form function $G(x)$ is a CR, or an expression with CRs as elementary operands - *CR-expression*. More precisely, we call an expression Φ to be a *CR-expression* if it represents one of the following functions over N∪{0}:

- a constant expression;
- a CR $\{\varphi_0, \odot_1, \varphi_1, \odot_2, \ldots, \odot_k, f_k\}$ where f_k is a CR-expression;
- a function $F(\Phi_1, \Phi_2, \ldots \Phi_m)$ of m arguments, where $\Phi_1, \Phi_2, \ldots, \Phi_m$ are CR-expressions.

The definition of the length of a CR is generalized by defining the *Cost Index (CI)* of a CR-expression Φ to be:

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

Here q is the number of operations in the expression F [3]. The cost index of a CR-expression gives an indication of its evaluation cost (it counts the number of operations needed to evaluate a CR-expression at one point).

2.2 CR simplifying rules and normal form

Let us recall the simplifying rules, which are used at the time of CR-construction. Rules considered in [3] are given in Figure 1 under numbers 1-14 (the last column represents a decrease of CI when the corresponding rule is applied).

Remark. Explicit formula for δ_t (rule 11) is

$$\delta_t = \sum_{u=max(0,t-l)}^{min(t,k)} \binom{t}{u} \varphi_u \sum_{v=t-u}^{min(t,l)} \binom{u}{t-v} \psi_v \quad (4)$$

[1] Note, that the r-order subchain of the CR Φ is CR of the length $k - r$

[2] Our desire to represent by CRs, as large a subexpression as possible has a very simple explanation: any successful step of application of the CR-simplifying rules decreases complexity of future computational scheme.

[3] For simplicity, we count all + and * as binary operations.

N	before simplification	after simplification	CI decrease
1.	$\{\varphi_0, +, f_1\} \pm c$	$\{\varphi_0 \pm c, +, f_1\}$	1
2.	$c * \{\varphi_0, *, f_1\}$	$\{c * \varphi_0, *, f_1\}$	1
3.	$c * \{\varphi_0, +, f_1\}$	$\{c * \varphi_0, +, c * f_1\}$	≥ 0
4.	$c^{\{\varphi_0, +, f_1\}}$	$\{c^{\varphi_0}, *, c^{f_1}\}$	≥ 0
5.	$\{\varphi_0, *, f_1\}^c$	$\{\varphi_0^c, *, f_1^c\}$	≥ 0
6.	$\log(\{\varphi_0, *, f_1\})$	$\{\log(\varphi_0), +, \log(f_1)\}$	≥ 0
7.	$\{\varphi_0, +, f_1\} \pm \{\psi_0, +, g_1\}$	$\{\varphi_0 \pm \psi_0, +, f_1 \pm g_1\}$	≥ 1
8.	$\{\varphi_0, *, f_1\} * \{\psi_0, *, g_1\}$	$\{\varphi_0 \psi_0, *, f_1 g_1\}$	≥ 1
9.	$\{\varphi_0, +, \ldots, +, \varphi_k\} + \{\psi_0, +, \ldots, +, \psi_l\}$	$\{\varphi_0 + \psi_0, +, \ldots, \varphi_l + \psi_l, +, \varphi_{l+1}, \ldots, +, \varphi_k\}$	$l+1$
10.	$\{\varphi_0, *, \ldots, *, \varphi_k\} * \{\psi_0, *, \ldots, *, \psi_l\}$	$\{\varphi_0 * \psi_0, *, \ldots, \varphi_l * \psi_l, *, \varphi_{l+1}, \ldots, *, \varphi_k\}$	$l+1$
11.	$\{\varphi_0, +, \ldots, +, \varphi_k\} * \{\psi_0, +, \ldots, +, \psi_l\}$	$\{\delta_0, +, \delta_1, +, \ldots, +, \delta_{k+l}\}$	1
12.	$\{\varphi_0, *, \ldots, *, \varphi_k\}^{\{\psi_0, +, \ldots, +, \psi_l\}}$	$\{\xi_0, *, \xi_1, *, \ldots, *, \xi_{k+l}\}$	1
13.	$\varphi_1 \in \mathbf{Z}, \varphi_1 > 0$ $\{\varphi_0, +, \varphi_1\}!$	$\{\varphi_0!, *, \tau_0, +, \tau_1, +, \ldots, +, \tau_{\varphi_1}\}$	≥ 0
14.	$\varphi_1 \in \mathbf{Z}, \varphi_1 < 0$ $\{\varphi_0, +, \varphi_1\}!$	$\{\varphi_0!, *, \{\rho_0, +, \rho_1, +, \ldots, +, \rho_{\lvert\varphi_1\rvert}\}^{-1}\}$	≥ 0
15.	$\{\varphi_0, \odot_1, \ldots, \odot_k, \varphi_k, +, 0\}$	$\{\varphi_0, \odot_1, \ldots, \odot_k, \varphi_k\}$	1
16.	$\{\varphi_0, \odot_1, \ldots, \odot_k, \varphi_k, *, 1\}$	$\{\varphi_0, \odot_1, \ldots, \odot_k, \varphi_k\}$	1
17.	$\{c\}$	c	0

Figure 1: CRs' simplifying rules (here c is a constant expression; $k \geq l$ in 9 and 10)

for $t = 0, 1, \ldots, k+l$. In fact it shows how to get the table of finite differences for the polynomial $h(i) = f(i)g(i)$ (when the corresponding tables for $f(i)$ and $g(i)$ are given) without referring to coefficients of the polynomials. This formula is not used directly to construct CRs (see [3]), but is useful as an auxiliary. For example, the relation between the last components of factors and product:

$$\delta_{k+l} = \binom{k+l}{k} \varphi_k \psi_l, \tag{5}$$

extracted from the explicit formula, is analogous to the relation between leading coefficients of correspondent polynomials and will be used in the next section.

After adding elementary simplifying rules to the considered list (rules 15-17) we say that the CR-expression Φ is represented by the *normal* form, if neither rules 1-14 nor rules 15-17 can be applied to it. Further we will consider only CR-expressions in normal form and try to support the point of view on CRs as internal representation of expressions. For some classes of expressions this representation gives the canonical form. For example, CR-representation of polynomials is analogous to dense [5] representation of polynomials. Further we will extend the set of simplifying rules to obtain a canonical representations for rational functions.

Remark. Rules 1-17 assume that we have an algorithm that recognizes 0, 1, and *constants*. (Most of the optimizing transformations assume the same.) Components of CRs (as expressions) are constructed almost from the same set of constants, operation-signs and variables as initial expression $G(x)$ (only new variables x_0, h could be added to the initial list of variables). For example, if $G(x)$ is a polynomial in x with rational coefficients, then components of the CR for $G(x)$ will be polynomials in two variables (x_0, h) with rational coefficients. If we have a "zero-recognition" algorithm in the domain of $G(x)$, we can assume the existence of a "zero-recognition" algorithm in the domain of all CR components obtained by $G(x)$.

2.3 CR-evaluation and the shift operator

The distinctive feature of the CR-representation is its orientation towards fast iterative computation. Consider the shift operator E:

$$E_u(f(u)) = f(u+1),$$

and examine how it is applied to CR-expressions [4]. Given CR-expression Φ, we can define the function $\mathtt{Value}(\Phi)$:

$$\mathtt{Value}(\Phi) = \begin{cases} c, & \text{if } \Phi \text{ is a constant expression } c \\ \varphi_0, & \text{if } \Phi = \{\varphi_0, \odot_1, \varphi_1, \ldots, \odot_k, f_k\} \\ F(\mathtt{Value}(\Phi_1), \mathtt{Value}(\Phi_2), \ldots, \mathtt{Value}(\Phi_m)), & \\ & \text{if } \Phi = F(\Phi_1, \Phi_2, \ldots, \Phi_m) \end{cases}$$

and describe the result of the application of E to Φ:

$$E(\Phi) = \begin{cases} c, & \text{if } \Phi \text{ is a constant expression } c \\ \{\varphi_0 \odot_1 \varphi_1, \odot_1, \varphi_1 \odot_2 \varphi_2, \ldots & \\ \ldots, \varphi_{k-1} \odot_k \mathtt{Value}(f_k), \odot_k, E(f_k)\}, & \\ \quad \text{if } \Phi = \{\varphi_0, \odot_1, \varphi_1, \ldots, \odot_k, f_k\} & \\ F(E(\Phi_1), E(\Phi_2), \ldots, E(\Phi_m)), & \\ \quad \text{if } \Phi = F(\Phi_1, \Phi_2, \ldots, \Phi_m) & \end{cases}$$

Definitions of $E(\Phi)$ and $\mathtt{Value}(\Phi)$ are coordinated with the definition of $CI(\Phi)$ and explain this last. After constructing

[4] Further we will only use shift with respect to variable i and omit index near E-sign.

the CR-expression Φ we can compute values of $G(x_0 + ih)$, $i = 0, 1, \ldots, N$, interpreting Φ in the following manner:

```
Initialize(Φ);
for i:= 0 to N do
  begin write( Value(Φ) );
    Φ:=E(Φ)
  end;
```

Here `Initialize(Φ)` initializes components of Φ with actual values of x_0, h and other variables involved (if needed). Function `Value(Φ)` returns the current value of G, and Φ:=$E(\Phi)$ updates the CR-expression for the next point.

Example 1 Consider the expression

$$z(i) = \frac{(2i)!}{3^{i^2+1}} + ai + b.$$

Result of the call `CRmake`$(z(i), i, 0, 1)$ is the CR-expression

$$\{b, +, a\} + \{1/3, *, \{2, +, 10, +, 8\} \cdot \{1/3, *, 1/9\}\}.$$

Below we present a Maple-V session, which constructs this CR-expression and gives the results of interpreting for the first several steps (square brackets are used in Maple-session instead of curly braces):

```
> zz:=(2*i)!/3^(i^2+1)+a*i+b;

                          (2 i)!
                zz := --------- + a i + b
                            2
                         (i  + 1)
                        3

> g:=CRmake(zz,i,0,1): lprint(g);

[b,+, a]+[1/3,*, [2,+, 10,+, 8]*[1/3,*, 1/9] ]
> Value(g);

                              b + 1/3
> g:=E_(g): lprint(g);
[b+a,+, a]+[2/9,*, [12,+, 18,+, 8]*[1/27,*, 1/9] ]
> Value(g);

                             b + a + 2/9
> g:=E_(g): lprint(g);
[b+2*a,+, a]+[8/81,*, [30,+, 26,+, 8]*[1/243,*, 1/9] ]
> Value(g);

                            b + 2 a + 8/81
> g:=E_(g): lprint(g);
[b+3*a,+,a]+[80/6561,*, [56,+,34,+,8]*[1/2187,*, 1/9] ]
> Value(g);

                                    80
                            b + 3 a + ----
                                    6561
```

3 Rational simplifications of CRs

The algorithm `CRmake` does not simplify rational subexpressions of the given $G(x)$. Here we will consider new CR-simplifying rules and define operations on CRs, which will be useful in handling rational functions and complicated formulae, involving factorials, polynomials etc. Most of these simplifications could be done during preprocessing of the input of `CRmake` by Computer Algebra System (CAS). We consider CR equivalent of rational simplifications to underscore the role of CRs as the form of internal representation. Further, some of the transformations (should they be done during preprocessing) can not be expressed without consideration of recurrences (see last example of this section).

3.1 Simplifications based on division of CRs

Given simple pure-sum CRs

$$\Phi = \{\varphi_0, +, \ldots, \varphi_k\} \text{ and } \Psi = \{\psi_0, +, \ldots, \psi_l\}, \ (k \geq l),$$

obtained for polynomials $F(x)$ and $G(x)$ respectively (assume, we only deal with polynomials in x with rational coefficients). We need the simple pure-sum CRs Ξ of length $k-l$ and Δ of length $m < l$ such that $\Phi = \Xi\Psi + \Delta$. We can compute this by forming the *quotient* $U(x)$ and *remainder* $V(x)$ of the given polynomials and by constructing CRs for U and V. But it can also be done directly, using the algorithm that divides simple pure-sum CRs. It is similar to the corresponding algorithm for polynomials in a dense representation:

Algorithm **DivPureSumCRs**
Input: simple pure-sum CRs
$\Phi = \{\varphi_0, +, \ldots, \varphi_k\}$ and $\Psi = \{\psi_0, +, \ldots, \psi_l\}$ $(k \geq l)$.
Output: simple pure-sum CRs Ξ and Δ,
such, that $L(\Xi) = k - l$, $L(\Delta) < l$ and $\Phi = \Xi\Psi + \Delta$.

$\Xi := 0; \Delta := \Phi;$
`for t:=k-l downto 0 do`
`  begin`
$\quad \xi_t := \frac{\mathsf{Comp}(t+l, \Delta)}{\psi_l\binom{t+l}{t}};$
$\quad \Gamma := \{\underbrace{0, +, 0, +, \ldots, 0}_{t \text{ times}}, +, \xi_t\};$
$\quad \Delta := \Delta - \Psi\Gamma; \ \ \Xi := \Xi + \Gamma$
`  end;`

On each step of the loop we get the next component of the quotient Ξ (starting from the last one) like in the polynomial case, where we get the next coefficient of the quotient $U(x)$ (starting from the leading coefficient). Assume that Φ and Ψ were obtained via `CRmake` with rational numbers as actual parameters for x_0, h. All the components of Φ and Ψ are rational numbers. It follows from (4), (5) and simplifying rule 11, that $lc(\Psi\Gamma) = lc(\Delta)$ before performing subtraction on each step of the loop. We assume that after each evaluation of the difference $\Delta - \Psi\Gamma$, the result is transformed to normal form by reduction of zero $lc(\Delta - \Psi\Gamma)$. Auxiliary CR Γ is an analogy of the monomial in polynomial case, and multiplication $\Psi\Gamma$ is performed quite fast (even with the aid of explicit formula (4)), because of this specific of Γ. Denote the result of multiplication $\Psi\Gamma$ as $\Lambda = \{\lambda_0, +, \ldots, \lambda_{t+l}\}$. Then we can derive from (4):

$$\lambda_s = \begin{cases} 0, & \text{if } s < t \\ \xi_t \sum_{r=s-t}^{min(s,l)} \binom{s}{r} \psi_r \binom{r}{s-t}, & \text{if } s \geq t \end{cases}$$

Now, assume that Φ and Ψ were obtained via `CRmake` with symbolic actual parameters for x_0, h. We need to assure that the operation

$$\frac{\mathsf{Comp}(t+l, \Delta)}{\psi_l\binom{t+l}{t}}$$

is valid as division of polynomials in two variables. This follows immediately from the existence of the quotient U and remainder V for initial polynomials F and G.

Since we have the algorithm to divide simple pure-sum CRs, we can easily develop an algorithm to find GCD of simple pure-sum CRs. If initial expressions are polynomials with coefficients from a ring, we can define for CR-representation (as for any other representations of polynomials) algorithms to compute content, primitive part, pseudo-remainder etc., as it is done in [5].

Consider now the CR-expression $\frac{\Phi}{\Psi} = \frac{\{\varphi_0,+,\ldots,\varphi_k\}}{\{\psi_0,+,\ldots,\psi_l\}}$. If $l < k$ we can use the simplifying rule:

$$\frac{\Phi}{\Psi} = \Xi + \frac{\Delta}{\Psi}.$$

Here $CI(LHS) = k + l + 1 \geq k + 2 + m = CI(RHS)$. Another possible simplification – reduction of the Φ and Ψ on $\mathtt{GCD}(\Phi, \Psi)$.

3.2 Absorption of polynomials by factorials

Besides extension of the list of CR-simplification rules, the later algorithm gives us an opportunity to define (again in terms of CR-representation) additional rules, which could be quite useful when we handle complicated formulae with factorials (for example, products of binomial coefficients). Given CRs

$$\Phi = \{\varphi_0, *, \varphi_1, +, \ldots, +, \varphi_k\} \text{ and } \Psi = \{\psi_0, +, \ldots, +, \psi_l\}$$

with rational components. We need to simplify CR-expressions $\Phi\Psi$ or $\frac{\Phi}{\Psi}$. Consider first $\Phi\Psi$ which could be rewritten in terms of BRs as

$$\{\varphi_0\psi_0, *, \frac{\Phi_1(\Psi + \Psi_1)}{\Psi}\}$$

(recall that $\Psi_1 = \{\psi_1, +, \ldots, +, \psi_l\}$).

If $\Psi|\Phi_1(\Psi + \Psi_1)$ then (denoting $\Gamma = \Phi_1(\Psi + \Psi_1)/\Psi$) we have $L(\Gamma) = L(\Phi_1) = k - 1$, and instead of evaluation of the CR-expression $\Phi\Psi$ with $k + l + 1$ operations we can evaluate CR $\{\varphi_0\psi_0, *, \Gamma\}$ with k operations. Decrease of the computational complexity is evident in this case. This transformation is an analogous to the transformation

$$(2i + 1)!(4i^2 + 10i + 6) \to (2i + 3)!$$

If $\mathbf{not}(\Psi|\Phi_1(\Psi + \Psi_1))$, but $\Psi = \Xi\Delta$ and

$$\Xi|\Phi_1(\Xi + \Xi_1), \tag{6}$$

then the transformation

$$\Phi\Xi\Delta \to \tilde{\Phi}\Delta \tag{7}$$

is possible and a decrease of CI is obtained in this way. It has the following analogy in the usual notation:

$$(i + 5)!(2i + 1)!(4i^2 + 10i + 6)(i + 6)(3i - 1) \to \\ \to (i + 6)!(2i + 3)!(3i - 1).$$

We need not factorize Ψ to find such a factor Ξ of maximal length. Observe, that

1. If $\Psi = \Xi\Delta$, then $\Psi + \Psi_1 = (\Xi + \Xi_1)(\Delta + \Delta_1)$;
2. If $\Xi|\Phi_1(\Xi + \Xi_1)$ and $\Xi|\Psi$, then $\Xi|\mathtt{GCD}(\Psi, \Phi_1(\Psi + \Psi_1))$;
3. If $\mathbf{not}(\Psi|\Phi_1(\Psi + \Psi_1))$, then $L(\mathtt{GCD}(\Psi, \Phi_1(\Psi + \Psi_1))) < L(\Psi)$;
4. $\Xi = \mathtt{GCD}(\Psi, \Phi_1(\Psi + \Psi_1))$ not necessary enjoy the property (6) (it could have redundant co-factors, since $\mathtt{GCD}(\Delta, \Delta + \Delta_1)$ could be non-trivial [5]).

Thus, after several steps of computing GCD of simple pure-sum CRs we either find the non-trivial factor Ξ of CR Ψ, which satisfies (6), or prove non-existence of such a factor. If the factor $\Xi, L(\Xi) > 0$ is found, then with the help of transformation such as (7) we can decrease CI of the expression $\Phi\Psi$.

For CR-expression $\frac{\Phi}{\Psi}$ the situation is analogous, just observe, that in terms of BRs

$$\frac{\Phi}{\Psi} = \{\frac{\varphi_0}{\psi_0}, *, \frac{\Phi_1\Psi}{\Psi + \Psi_1}\}.$$

CR-expressions with subexpressions $\{\varphi_0, *, \Phi_1^{-1}\}$ (such as in rule 14, Figure 1) are handled after obvious pre-transformation to the form $\frac{1}{\{\frac{1}{\varphi_0}, *, \Phi_1\}}$.

Example 2 Consider an expression

$$y(i) = \frac{(3i + 1)!}{(2i + 2)!}(18i^3 + 45i^2 + 34i + 8)$$

which we need to compute for $i = 0, 1, \ldots$.
The $\mathtt{CRmake}(y, i, 0, 1)$ call returns the CR-expression

$$\Phi = \{\frac{1}{2}, *, \frac{\{24, +, 186, +, 324, +, 162\}}{\{12, +, 18, +, 8\}}\} \cdot \{8, +, 97, +, 198, +, 108\}.$$

Performing transformations like (7) gives us the CR-expression

$$\Psi = \{4, *, \frac{\{105, +, 375, +, 432, +, 162\}}{\{4, +, 14, +, 8\}}\}.$$

The decrease in computational complexity is obvious:

$$CI(\Phi) = 11, \text{ when } CI(\Psi) = 7.$$

We cannot describe function $y(i)$ only in terms of factorials, but equality $y(i) = \Psi(i)$ implies that $y(i)$ satisfies the following recurrence (in usual notation):

$$y(i) = \begin{cases} 4, & \text{if } i = 0, \\ y(i-1) \cdot \frac{3i(3i+2)(3i+4)}{(2i-1)(2i+2)}, & \text{if } i > 0. \end{cases}$$

4 Expediting transformations of CR-expressions

Given $G(x), x, x_0, h$ we have the CR-expression

$$\Phi = \mathtt{CRmake}(G(x), x, x_0, h).$$

Suppose we transform $G(x)$ into equivalent $\tilde{G}(x)$ and get

$$\Psi = \mathtt{CRmake}(\tilde{G}(x), x, x_0, h).$$

It is quite possible, that $CI(\Psi) < CI(\Phi)$. In this section we will consider transformations, which could decrease CI of the CR-expressions. The set of all possible transformations could be separated (conditionally) on

- *pre-transformations* – transformations of the initial expression $G(x)$ which guarantee a decrease of CI after CR construction, and

[5] Take, for example, $\Delta(i) = (3i + 2)(3i + 5)$, than $\Delta + \Delta_1 = (3i + 5)(3i + 8)$.

- *post-transformations* – transformations of the CR-expression Φ with the same purposes.

Since we try to handle CR-expressions as a usual form of internal representation, we will consider transformations of CR-expressions. These transformations could be separated (again conditionally) as

- *"blind"* – transformations which use only knowledge about kinds (pure-sum, pure-product, etc.) and lengths of CRs, involved in an expression, and
- *"non-blind"* – transformations that have access to the internal CR-representation.

"Blind" transformations of CR-expressions were considered in [6]. They were based, in particular, on commutativity, associativity and distributivity laws and application of CR-simplifying rules. Some of these transformations do not require backtracking and can be performed during CR-construction. (Actually, current Maple-V implementation of `CRmake` does these transformations on the first pass.)

Here we will consider "non-blind" transformations, which (at first glance) are very similar to well-known "common-subexpressions" [7] optimizing transformations. But the linear nature of simple CRs allows the faster solution of an analogous problem.

4.1 Looking for common subchain of two CRs

Let us consider two simple CRs

$$\Phi = \{\varphi_0, \odot_1, \varphi_1, \odot_2, \ldots, \varphi_{k-1}, \odot_k, \varphi_k\}$$

and

$$\Psi = \{\psi_0, \ominus_1, \psi_1, \ominus_2, \ldots, \psi_{l-1}, \ominus_l, \psi_l\}.$$

We say that Φ and Ψ have a *common subchain* of length t $(0 \le t \le \min(k,l))$, if

$$\begin{aligned} &\varphi_{k-j} = \psi_{l-j}, \qquad j = 0, 1, \ldots, t, \qquad \text{and} \\ &\odot_{k-j} = \ominus_{l-j}, \qquad j = 0, 1, \ldots, t-1. \end{aligned}$$

If Φ and Ψ have a common subchain of length $t > 0$, then, the CRs Φ_{k-t} and Ψ_{l-t} define the same function. Therefore, instead of evaluating Φ and Ψ, we can evaluate Φ and $\tilde{\Psi} = \{\psi_0, \ominus_1, \ldots, \psi_{l-t-1}, \ominus_{l-t}, \Phi_{k-t}\}$ in order to decrease the computational complexity of t operations. To avoid "repeated" shifting, when interpreting a CR-expression, we need to correct the definition of $E(\Psi)$ from previous section:

$$E(\Phi) = \begin{cases} c, \quad \text{if } \Phi \text{ is a constant expression } c; \\ \{\varphi_0 \odot_1 \varphi_1, \odot_1, \varphi_1 \odot_2 \varphi_2, \ldots, \\ \ldots, \varphi_{k-1} \odot_k \varphi_k, \odot_k, \varphi_k\}, \\ \quad \text{if } \Phi \text{ is simple CR } \{\varphi_0, \odot_1, \varphi_1, \ldots, \odot_k, \varphi_k\}; \\ \{\varphi_0 \odot_1 \varphi_1, \odot_1, \varphi_1 \odot_2 \varphi_2, \ldots \\ \ldots, \varphi_{k-1} \odot_k \mathtt{Value}(\Xi_t), \odot_k, \Xi_t\}, \\ \quad \text{if } \Phi \text{ is simple CR } \{\varphi_0, \odot_1, \varphi_1, \ldots, \odot_k, \Xi_t\} \\ \quad \text{which refers to the common subchain } \Xi_t; \\ \{\varphi_0 \odot_1 \varphi_1, \odot_1, \varphi_1 \odot_2 \varphi_2, \ldots \\ \ldots, \varphi_{k-1} \odot_k \mathtt{Value}(f_k), \odot_k, E(f_k)\}, \\ \quad \text{if } \Phi = \{\varphi_0, \odot_1, \varphi_1, \ldots, \odot_k, f_k\}; \\ F(E(\Phi_1), E(\Phi_2), \ldots, E(\Phi_m)), \\ \quad \text{if } \Phi = F(\Phi_1, \Phi_2, \ldots, \Phi_m). \end{cases}$$

The definition of CI can be corrected easily in the same way. If Φ and Ψ have a common subchain of length $t = 0$, we can not decrease the computational complexity, but we can decrease the amount of memory needed for CR-representation. The algorithm to find a common subchain of two CRs is given below.

Algorithm **CommonSubChain**
Input: simple CRs $\Phi = \{\varphi_0, \odot_1, \varphi_1, \odot_2, \ldots, \varphi_{k-1}, \odot_k, \varphi_k\}$ and $\Psi = \{\psi_0, \ominus_1, \psi_1, \ominus_2, \ldots, \psi_{l-1}, \ominus_l, \psi_l\}$, $(k \ge l)$
Output: integer t such that $\Phi_{k-t} = \Psi_{l-t}$, i.e., the length of the maximal common subchain of the given CRs (if $t < 0$, then Φ and Ψ do not have a common subchain).

```
if  φ_k = ψ_l  then t:=0 else return(-1);
while  t < l  do
  if  (⊙_{k-t} = ⊖_{l-t})  and  (φ_{k-t-1} = ψ_{l-t-1})
     then t:=t+1
     else return(t);
return(t)
```

This algorithm is very simple (unlike algorithms to find common subexpressions) and effective: on one hand, we halt the algorithm immediately if the result of the first comparison is $false$; on the other hand, any new step of the algorithm will be effective (the value of t increases with the future optimizing effect).

4.2 Evaluation of common sub-chain with delay

Let us consider two simple CRs

$$\Phi = \{\varphi_0, \odot_1, \varphi_1, \odot_2, \ldots, \varphi_{k-1}, \odot_k, \varphi_k\}$$

and

$$\Psi = \{\psi_0, \ominus_1, \psi_1, \ominus_2, \ldots, \psi_{k-1}, \ominus_k, \psi_k\}$$

of the same length, and an integer α. We say that Φ is equal to Ψ with *delay* α, if $\Phi = E^{\alpha}(\Psi)$.

Suppose, that CRs Φ and Ψ were obtained as the result of `CRmake` call with symbolic parameter x_0. This means, that φ_j and ψ_j depend on x_0. If $\Phi = E^{\alpha}(\Psi)$, then

$$\varphi_j = \mathbf{subst}\ (x_0 = x_0 + \alpha h, \psi_j), j = 0, 1, \ldots, k,$$

and

$$\odot_j = \ominus_j, j = 1, 2, \ldots, k.$$

In particular,

$$\varphi_k = \psi_k,$$

and

$$\varphi_{k-1} = \begin{cases} \psi_{k-1} + \alpha\varphi_k, \text{if } \odot_k = +, \\ \psi_{k-1}\varphi_k^{\alpha}, \text{if } \odot_k = *. \end{cases} \tag{8}$$

Given simple CRs Φ and Ψ, the last four equalities allow us to formulate an effective algorithm to find their subchains which are equal except for a delay.

Algorithm **DelayedSubChain**
Input: simple CRs $\Phi = \{\varphi_0, \odot_1, \varphi_1, \odot_2, \ldots, \varphi_{k-1}, \odot_k, \varphi_k\}$ and $\Psi = \{\psi_0, \ominus_1, \psi_1, \ominus_2, \ldots, \psi_{l-1}, \ominus_l, \psi_l\}$, such that $(k \ge l)$ and $\varphi_k = \psi_l$
Output: integer t and α such that $\Phi_{k-t} = E^{\alpha}(\Psi_{l-t})$, i.e., the length of maximal delayed subchain of given CRs and the value of the delay (if $t < 0$, then Φ and Ψ do not have subchains which are equal except for a delay).

```
if  ⊙_k ≠ ⊖_l  then return(-1, 0);
taking into account the values of φ_k, φ_{k-1}, ⊙_k, ψ_{l-1},
check if there exists non-zero α, which enjoy (8) ;
```

```
if α does not exist
  then return(-1,-1)
  else t:=1;
while  t < l  do
  if   (⊙_{k-t} = ⊖_{l-t})  and
       (φ_{k-t-1} = subst(x_0 = x_0 + αh, ψ_{l-t-1}))
    then t:=t+1
    else return(t,α);
return(t,α)
```

This algorithm uses the linear nature of CRs, and is also effective: if there is no solution, the algorithm halts after the first or the second step, but each additional step increases the value of t and the possible optimizing effect.

If Φ and Ψ have equal with delay $\alpha > 0$ subchains of the length t ($t > 0$), i.e. $\Phi_{k-t} = E^{\alpha}(\Psi_{l-t})$, then, instead of evaluating Φ and Ψ, we can evaluate

$$\Phi \text{ and } \tilde{\Psi} = \{\psi_0, \ominus_1, \ldots, \psi_{l-t-1}, \ominus_{l-t}, E^{-\alpha}(\Phi_{k-t})\}$$

in order to replace t operations $\ominus_{l-t+1}, \ldots, \ominus_l$ by α assignments, which implement a "delay-line" (keeping values, computed α steps before) in the computational scheme.

Example 3 Assume we need to compute

$$F(x) = 3x^2 + 11x + 10 \text{ and } G(x) = x^3 - 2x^2 + x + 1$$

for $x = 0, 1, 2, \ldots$. The `CRmake` call with symbolic parameter x_0 and $h = 1$ gives us
$\Phi = \{3x_0^2 + 11x_0 + 10, +, 6x_0 + 14, +, 6\}$ for $F(x)$ and
$\Psi = \{x_0^3 - 2x_0^2 + x_0 + 1, +, 3x_0^2 - x_0, +, 6x_0 + 2, +, 6\}$ for $G(x)$.
Since $lc(\Phi) = lc(\Psi)$ we can try to find common-with-delay subchain of Φ and Ψ. $DelayedSubChain(\Phi, \Psi)$ returns $t = 2$ and $\alpha = 2$, i.e. $\Phi = E^2(\Psi_1)$. Therefore, we can use the relation $\Psi = \{1, +, E^{-2}(\Phi)\}$ in order to replace 2 additions by 2 assignments. The program to compute $F(x)$ and $G(x)$ values could be as following:

```
f0:=10; f1:=14; f2:=6;
g0:=1; dl1:=0; dl2:=2; write(f0,g0);
for i:=1 to n do  begin
     g0:=g0+dl1; dl1:=dl2; dl2:=f0;
     f0:=f0+f1; f1:=f1+f2; write(f0,g0)  end
```

4.3 Based on common subchains transformation of the CR computational scheme

Given the CRs Φ and Ψ, denote the length of their common subchain (i.e. the result of **CommonSubChain** call) by $CSC(\Phi, \Psi)$. We can define a over the set of simple CRs as a binary relation, such that $a : \Phi a \Psi \leftrightarrow CSC(\Phi, \Psi) \geq 0$. It is easy to show that a is the equivalence relation. Therefore, it separates the set of simple CRs into classes: two CRs Φ and Ψ belong to the same class if and only if $\Phi a \Psi$. Moreover, two CRs Φ and Ψ belong to the same class if and only if $lc(\Phi) = lc(\Psi)$.

Assume we have the set of simple CRs

$$\Phi^{(1)}, \Phi^{(2)}, \ldots, \Phi^{(n)} \tag{9}$$

of length $c_1, c_2, \ldots, c_n$ respectively. In order to effectively find all possible common subchains and organize the interpretation of this set of CRs correctly, assume that

1. we are able to recognize the equality of the components $\varphi_j^{(m)}$,$\varphi_i^{(l)}$ from given CRs;
2. in the case of inequality, we are able to order $\varphi_j^{(m)}, \varphi_i^{(l)}$ w.r.t. some natural ordering (for example, with the help of a procedure like **ordp** in Reduce-3.5).

This means that we have an order $\preceq$ defined on the set of expressions $\varphi_j^{(m)}$. Assume additionally, that $+ \preceq *$. Optimizing transformation of (9) has been performed in several simple steps.

Rewrite given CRs $\Phi^{(1)}, \Phi^{(2)}, \ldots, \Phi^{(n)}$ in reverse order:

$$\varphi_{c_1}^1, \odot_{c_1}^1, \varphi_{c_1-1}^1, \odot_{c_1-1}^1, \ldots, \odot_1^1, \varphi_0^1$$
$$\ldots$$
$$\varphi_{c_n}^n, \odot_{c_n}^n, \varphi_{c_n-1}^n, \odot_{c_n-1}^n, \ldots, \odot_1^n, \varphi_0^n$$

We can now sort them as words with letters $\varphi_j^{(m)}, +, *$. First, let us sort these words with respect to the first letter, $\varphi_{c_j}^j$, and denote the result of that sorting as $\Psi^{(1)}, \Psi^{(2)}, \ldots, \Psi^{(n)}$.

It is now easy to find all non-overlapping segments $[u, v]$ ($u, v \in \mathbf{N}, 1 \leq u, v \leq n$) of the maximal length, such that $lc(\Psi^{(i)}) = lc(\Psi^{(j)})$ for all $i, j \in [u, v]$ (all CRs from the same segment belong to the same class w.r.t. binary relation a). Therefore, only CRs from the same segment might have common subchains. If all segments are trivial ($u = v$), then neither CRs with common subchains nor CRs with "equal-with-delay-subchains" in (9) can be found, and the search is done for this case.

All non-trivial segments are processed separately. Let us describe briefly the process of extracting common subchains from p CRs within a single class (distinct segment). We can sort CRs from distinct segment as words and denote the result of the sorting as $\Gamma^{(1)}, \ldots, \Gamma^{(p)}$. After sorting we compute the sequence $t_i, i = 0, \ldots, p$:
$t_i = CSC(\Gamma_{i+1}, \Gamma_i), i = 1, \ldots, p-1, t_0 = t_p = -1$,
and find the locations of the local minima and maxima in the sequence $\{t_i\}$, i.e. numbers
$l_0, l_1, \ldots, l_q; k_1, \ldots, k_q$, such that $l_0 = 0, l_q = p$,
$l_0 < k_1 < l_1 < k_2 < \ldots < k_q < l_q$ and

$$\begin{array}{ll} t_i < t_{k_j} & \text{for } k_j < i \leq l_j, \\ t_i \leq t_{k_j} & \text{for } k_j > i \geq l_{j-1}, \\ t_i > t_{l_j} & \text{for } k_j \leq i < l_j, \\ t_i \geq t_{l_j} & \text{for } k_{j+1} \geq i > l_j. \end{array}$$

Observe, that:

1. for any s such that $l_i < s \leq l_{i+1}$ and for any w: $1 \leq w \leq p$, $CSC(\Gamma_s, \Gamma_w) \leq CSC(\Gamma_s, \Gamma_{k_{i+1}})$. Therefore, for any $s : l_i < s \leq l_{i+1}$ and $s \neq k_{i+1}$, Γ_s can be rewritten with reference to its common subchain with $\Gamma_{k_{i+1}}$ for maximal "profit".
2. for any s, w, i: $s < l_i < w$, $s, w \in [1, p]$

   $$CSC(\Gamma_s, \Gamma_w) \leq t_{l_i}$$

   Therefore, the rewriting of Γ_s, $l_i < s \leq l_{i+1}$ with reference on $\Gamma_{k_{i+1}}$ does not influence the relations between length of common subchains in other CRs.

Let us rewrite all CRs $\Gamma_i, i \neq k_j$ with reference to corresponding Γ_{k_j} and remove them from consideration by assigning them a priority value of 0. We can repeat this process (computing values of t_i, finding the locations of minima and maxima, rewriting of CRs with reference to other CRs and so on) with the rest of CRs. When we remove new candidates, we assign a priority value of 1 $(2, 3, \ldots)$ to them. After several steps (not greater than $\log_2 p$) we will have a

sole CR Δ, as the remainder from initial $\Gamma^{(1)}, \ldots, \Gamma^{(p)}$ and will assign the highest priority value to it. All the other CRs refer on Δ directly or indirectly. The decrease of CI due to the use of common subchains will be maximal.

To define the CR-interpreting scheme correctly we need to separate the CR-expressions involved into two parts: the set of simple CRs and the remaining CR-expressions. When the shift operator is applied to a CR-representation, it treats the second part as it was described earlier and delays treatment of the first part. When all expressions from the second part have been treated, we can shift the simple CRs, taking into account priority values (starting to shift CRs with higher priority) and the most recent definition of E (see section 4.1), which allows us to avoid repeated shifting.

Example 4 Consider the set of simple CRs which belong to the same class w.r.t binary relation a:

$$\begin{aligned}
\Phi^{(1)} &= \{0,+,1,+,6,+,6\} &&= i^3\\
\Phi^{(2)} &= \{1,+,6,+,6\} &&= 3i^2+3i+1\\
\Phi^{(3)} &= \{2,+,3,+,6,+,6\} &&= i^3+2i+2\\
\Phi^{(4)} &= \{4,+,3,+,6,+,6\} &&= i^3+2i+4\\
\Phi^{(5)} &= \{7,+,6,+,6\} &&= 3i^2+3i+7
\end{aligned}$$

After rewriting them in reverse order and sorting we get following table:

								t_j
$\Phi^{(2)}$	6	+	6	+	1			2
$\Phi^{(1)}$	6	+	6	+	1	+	0	1
$\Phi^{(3)}$	6	+	6	+	3	+	2	2
$\Phi^{(4)}$	6	+	6	+	3	+	4	1
$\Phi^{(5)}$	6	+	6	+	7			-1

From this table we can derive $l_0 = 0, l_1 = 2, l_2 = 5, k_1 = 1, k_2 = 3$. Now, we can rewrite $\Phi^{(j)}, j = 1, 4, 5$ by referring to $\Phi^{(2)}$ and $\Phi^{(3)}$:

$$\begin{aligned}
\Phi^{(1)} &= \{0,+,\Phi^{(2)}\}\\
\Phi^{(4)} &= \{4,+,\Phi_1^{(3)}\}\\
\Phi^{(5)} &= \{7,+,\Phi_2^{(3)}\}
\end{aligned}$$

and remove them from consideration by assigning a priority value of 0. (Recall, that $\Phi_r^{(m)}$ denotes an r-order subchain of the CR $\Phi^{(m)}$, e.g. $\Phi_1^{(3)} = \{3,+,6,+,6\}$.)

The remaining CRs are:

								t_j
$\Phi^{(2)}$	6	+	6	+	1			1
$\Phi^{(3)}$	6	+	6	+	3	+	2	-1

On this point $l_0 = 0, l_1 = 2, k_1 = 1$ and we can rewrite $\Phi^{(3)}$ by referring to $\Phi^{(2)}$:

$$\Phi^{(3)} = \{2,+,3,+,\Phi_1^{(2)}\}$$

and remove it from consideration by assigning a priority value of 1. The last CR - $\Phi^{(2)}$ gets a priority value of 2. As a result we have following computational scheme:

$$\begin{aligned}
\Phi^{(2)} &= \{1,+,6,+,6\}\\
\Phi^{(3)} &= \{2,+,3,+,\Phi_1^{(2)}\}\\
\Phi^{(1)} &= \{0,+,\Phi^{(2)}\}\\
\Phi^{(4)} &= \{4,+,\Phi_1^{(3)}\}\\
\Phi^{(5)} &= \{7,+,\Phi_2^{(3)}\}
\end{aligned}$$

The CI (computational complexity) of this scheme is equal to 7, where the CI of initial scheme is equal to 13. The shift operator is applied to this set of CRs in the same order as they are written above (this order is coordinated with priority values, obtained on previous steps).

Remark. Examples of sources, from which we can get a CR-expression with a large number of simple CRs are
1. Huge expressions, obtained as the result of symbolic transformations in CAS.
2. Two (or multi) dimensional computations. Assume, for example, we have to compute

$$\frac{i^3(j^2/2-j/2+1)+3i^2j+i(j^2+2j)+2j^2-j}{2^j(i^3+3i^2j+i(3j+2)+7j+2)}$$

for $i = 0, 1, \ldots, n; j = 0, 1, \ldots, m$. After constructing CRs w.r.t. variable j we get the CR-expression

$$\frac{\{i^3,+,3i^2+3i+1,+,i^3+2i+4\}}{\{1,*,2\}\{i^3+2i+2,+,3i^2+3i+7\}}. \qquad (10)$$

To optimize computations in the outer loop (w.r.t. variable i) we need to construct CRs for all the components of (10) w.r.t. variable i. As the result of this construction we will get the set of CRs, considered in the example above.

5 Conclusion

We considered new CR simplification rules and CR optimizing transformations, which allow additional decreases in the computational complexity with the help of CR technique. All the algorithms proposed are effective, "unconditional" (w.r.t. relative speed of different arithmetic operations) and easy to implement. They are implemented in Maple-V Computer Algebra system [8]. In addition, we have implemented a CR code generator. From the theory of compiling point of view, problems in the implementation of the CR-interpreter and the CR-code-generator are very similar. The code which is generated by a given CR-expression looks like a CR-interpreting scheme from the subsection 2.3. We need to choose a mapping function which maps a given CR-expression onto the set of Maple variables. When that is done, we generate "Initialize" section of code consisting of initial assignments, and the contents of the loop in the code, which implements the shift operator for a given CR-expression explicitly (according to the chosen mapping) [2]. After (or during) code-generation, any other known optimizing transformations could be applied in order to obtain faster code [2].

Acknowledgements

I would like to thank several people who provided me with useful comments on earlier drafts, and helped in preparing final version of this paper: Sergei Abramov, Nicolai Ardelyan, Manuel Bronstein, Thomas Casavant, Niklaus Mannhart, Bruno Salvy, Todd Scheetz, Karthi Vadivelu and ISSAC referees.

References

[1] Zima E.V. Recurrent Relations and Speed Up of Computations using Computer Algebra Systems. in "Design and Implementation of Symbolic Computation Systems" (ed. J.P.Fitch), DISCO'92 Proceedings, Springer Verlag LNCS v.721, 1993, pp.152–161.

[2] Zima E.V. Numeric Code Optimization in Computer Algebra Systems and Recurrent Relations Technique, Proc. ISSAC'93, Kiev, Ukraine, July 1993, ACM Press, pp.42-46.

[3] Bachmann O., Wang P.S., Zima E.V. Chains of Recurrences – a method to expedite the evaluation of closed-form functions. Proc. ISSAC'94, Oxford, UK, July 1994, ACM Press, pp. 242-249.

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

[5] Davenport J., Siret Y., Tournier E. Calcul formel, Masson, 1987.

[6] Zima E.V. Transformations of expressions associated with systems of recursive relations. Moscow Univ. Computat. Maths. and Cybernetics, 1985, N 1, pp.60-66 (translation from Russian).

[7] van Hulzen J.A. SCOPE, a Source-Code Optimization PackagE for REDUCE, Twente Univ., The Netherlands, 1994.

[8] Char B.W., Geddes K.O. (and others) Maple-V. Language Reference Manual. Springer-Verlag, 1991.