RWTH Aachen

Department of Computer Science
Technical Report

Syntax-Directed Derivative Code (Part I: Tangent-Linear Code)

Uwe Naumann

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2005-16

RWTH Aachen · Department of Computer Science · August 2005

Syntax-Directed Derivative Code (Part I: Tangent-Linear Code)

Uwe Naumann

LuFG Software and Tools for Computational Engineering, Department of Computer Science
RWTH Aachen University, D-52056 Aachen Germany
WWW: `http://www.stce.rwth-aachen.de/SDDC`, Email: `naumann@stce.rwth-aachen.de`

Abstract. This is the first instance in a series of papers on single-pass generation of various types of derivative code by syntax-directed translation. We consider the automatic generation of tangent-linear code by forward mode automatic differentiation implemented as the bottom-up propagation of synthesized attributes on the abstract syntax tree. A proof-of-concept implementation is presented based on a simple LALR(1) parser generated by the parser generator `bison`. The proposed technique can be generalized easily to provide a method for computing directional derivatives of mathematical vector functions that are implemented as computer programs in the context of computer algebra systems and compilers for scientific computing. The main advantage of the syntax-directed approach to automatic differentiation is its elegance in terms of the implementation.

1 Motivation and Summary of Results

This paper presents a method for generating *tangent-linear* versions of numerical simulation programs[1] that implement vector functions

$$F : I\!R^n \to I\!R^m, \quad \mathbf{y} = F(\mathbf{x}), \quad \mathbf{x} = (x_k)_{k=1,\ldots,n},\ \mathbf{y} = (y_l)_{l=1,\ldots,m} \quad , \tag{1}$$

automatically by syntax-directed translation. Such tangent-linear programs $\dot{F} = \dot{F}(\mathbf{x}, \dot{\mathbf{x}})$ compute directional derivatives $\dot{\mathbf{y}}$, that is, products of the Jacobian matrix

$$F' = (f'_{l,k})_{k=1,\ldots,n}^{l=1,\ldots,m} \equiv \left(\frac{\partial y_l}{\partial x_k}\right)_{k=1,\ldots,n}^{l=1,\ldots,m} \in I\!R^{m\times n} \tag{2}$$

with a direction $\dot{\mathbf{x}}$ in the input space $I\!R^n$. Formally,

$$\dot{\mathbf{y}} = \dot{F}(\mathbf{x}, \dot{\mathbf{x}}) \equiv F' \cdot \dot{\mathbf{x}} \quad . \tag{3}$$

To motivate the need for tangent-linear codes we consider a system of nonlinear equations

$$F(\mathbf{x}) = 0, \quad F : I\!R^n \to I\!R^n \quad . \tag{4}$$

Given a good start estimate $\mathbf{x}^0$, the system can be solved by Newton's method with quadratic convergence as follows:

$$\begin{aligned} \delta\mathbf{x}^i &= -(F'(\mathbf{x}^i))^{-1} \cdot F(\mathbf{x}^i) \\ \mathbf{x}^{i+1} &= \mathbf{x}^i + \delta\mathbf{x}^i \end{aligned}$$

for increasing integer values i. At each step the algorithm requires the Jacobian F' of F at the current estimate $\mathbf{x}^i$. Finite difference quotients can be used to

[1] Without loss of generality, we focus on a subset of C.

approximate the entries of the Jacobian at the cost of $n+1$ and $2n$ function evaluations when using forward (or backward) and centered differences, respectively. However, it is well-known that step size control is a problem as illustrated in Figure 1 and discussed, for example, in [13]. To avoid these problems, the tangent-linear program can be run with $\dot{\mathbf{x}}$ ranging over the Cartesian basis vectors in $I\!R^n$ to obtain F' at roughly the same cost as that of (centered) finite differences but with machine accuracy.

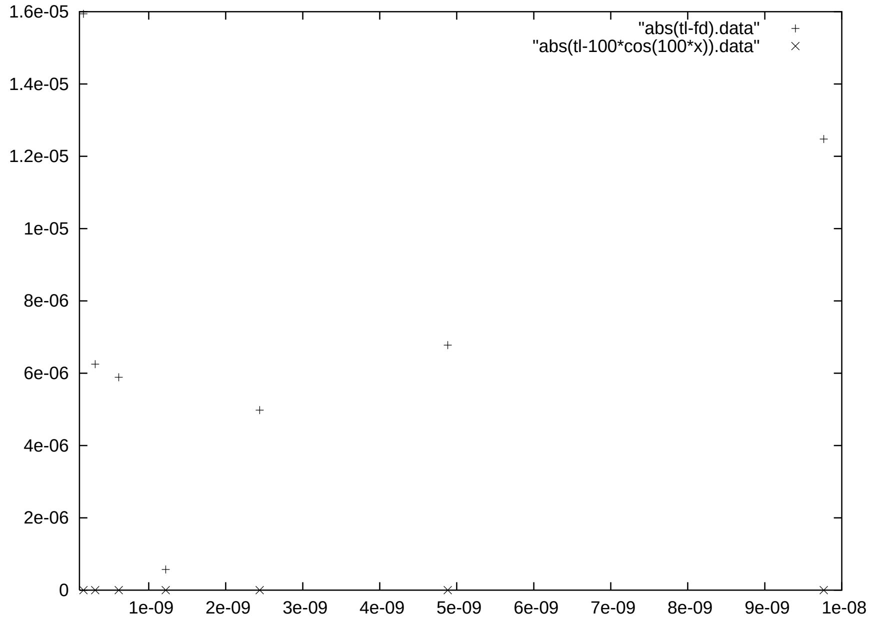

Fig. 1. Problems with finite differences: We show the absolute error of finite difference approximation of $\frac{\partial y}{\partial x}$ for $y = \sin(100 * x)$ at $x = 0.5$ and $h \in (10^{-10}, 10^{-8})$ (marked with "+" symbols). The values computed by the tangent-linear code are identical with those of the hand-coded derivative (correct up to machine accuracy) $\frac{\partial y}{\partial x} = 100 * \cos(100 * x)$. The "×" points mark the vanishing absolute error for this case.

Choosing a matrix-free approach, the Newton step can be obtained as the solution of the linear system

$$F'(\mathbf{x}^i)\delta\mathbf{x}^i = -F(\mathbf{x}^i) \tag{5}$$

at each Newton iteration i. Direct methods may be prohibitive due to the potentially large size of the problem. Iterative methods are likely to be more suitable. For example, iterative refinement computes the iterates as

$$\delta\mathbf{x}^{i+1} = \delta\mathbf{x}^i + B(F(\mathbf{x}^i) - F'(\mathbf{x}^i)\cdot\delta\mathbf{x}^i) \tag{6}$$

for a suitable preconditioner B, such as an approximate inverse of $F'(\mathbf{x}^i)$. Note that Equation (6), as well as matrix-free Krylov methods such as GMRES, involves the computation of the product of the Jacobian with a vector. A (forward) finite difference approximation of the Jacobian vector product in Equation (3) can be computed by perturbing the scalar input of the modified function

$$\tilde{F} : I\!R \rightarrow I\!R^m, \quad \tilde{\mathbf{y}} = \tilde{F}(s) \equiv F((s-1)\cdot\dot{\mathbf{x}} + \mathbf{x}) \tag{7}$$

at $s = 1$.[2] Hence, we compute $\mathbf{y} = F(\mathbf{x})$ and $\mathbf{y}' = F(h \cdot \dot{\mathbf{x}} + \mathbf{x})$ to get

$$\dot{\mathbf{y}} \approx \frac{\mathbf{y}' - \mathbf{y}}{h} \quad .$$

There is no need to form the whole Jacobian explicitly. Additional scaling in dependence of the value of a norm of $\dot{\mathbf{x}}$ may be required to avoid numerical instabilities as pointed out, for example, in [14]. Moreover, the usual problems with finite differences may occur. Tangent-linear codes should be used in order to circumvent both problems.

The structure of this paper is as follows. In Section 2 we summarize the theoretical concepts behind forward mode automatic differentiation in the context of tangent-linear code generation by source transformation. The syntax-directed translation algorithm for straight-line programs is introduced in Section 3 – the heart of this paper. Generalizations for subroutines with intraprocedural flow of control and programs with interprocedural flow of control are discussed. A simple proof-of-concept implementation is presented in Section 4, making detailed references to the source code that is appended in Section A. We draw conclusions in Section 5 and give an outlook to part II of this paper which deals with the syntax-directed generation of adjoint code.

2 Fundamentals

For a given implementation of a vector function as defined in Equation (1) as a computer program[3] we use *automatic differentiation (AD)* [17], [9][4] by source transformation to generate code for computing directional first derivatives of F. Therefore the computation of $\mathbf{y} = F(\mathbf{x})$ is expected to decompose into a sequence of elemental assignments

$$v_j = \varphi_j(v_i)_{i \prec j} \tag{8}$$

for $j = 1, \ldots, p+m$, and $i \prec j$ if and only if v_i is an argument of φ_j. Equation (8) is also referred to as the *code list* of F. We set $v_{i-n} = x_i$ for $i = 1, \ldots, n$ and $v_{p+j} = y_j$ for $j = 1, \ldots, m$. The v_k, $k = 1 - n, \ldots, p + m$, are called *code list variables*. Forward mode AD transforms F into the tangent-linear model $\dot{F}$ that computes a total (or directional) derivative as defined in Equation (3). The $m \times n$ Jacobian of F is defined in Equation (2). The transformation of the program semantics is achieved by applying well-known differentiation rules to the *elemental* functions

$$\varphi_j \in \{+, -, *, /, \sin, \exp, \ldots\}^5$$

[2] The basic idea is the following: We need

$$\frac{\partial x_i}{\partial s} = \dot{x}_i$$

for $i = 1, \ldots, n$. Therefore, we set $\mathbf{x} = \dot{\mathbf{x}} \cdot s + \tilde{\mathbf{x}}$ to determine suitable values for $\tilde{\mathbf{x}}$. For $s = 1$ the right-hand side of $\tilde{\mathbf{x}} \equiv \mathbf{x} - \dot{\mathbf{x}}$ can be substituted in $F(\mathbf{x})$ to get Equation (7).

[3] From now on we will use F to refer to this implementation as a computer program.

[4] We follow the notation therein as closely as possible.

[5] We consider a subset of the arithmetic operators and intrinsic functions provided by most programming languages.

followed by the exploitation of the chain rule as

$$\dot{v}_j = \sum_{i \prec j} c_{ji} \cdot \dot{v}_i \tag{9}$$

for $j = 1, \ldots, p+m$ and total derivatives $\dot{v}_k$ of the code list variables v_k, $k = 1-n, \ldots, p+m$. The elemental functions φ_j are assumed to be continuously differentiable in a neighborhood of the current argument. The corresponding local partial derivatives are denoted by

$$c_{ji} = \frac{\partial \varphi_j}{\partial v_i} \quad \text{for } j = 1, \ldots, p+m, \; i \prec j \quad ,$$

where the *independent* variables $v_{i-n} = x_i$, $i = 1, \ldots, n$, are assumed to be mutually independent. The basic approach dates back to [19]. Since then, AD has been used in the context of a large number of projects in computational science and engineering providing fast and accurate derivative information for a wide variety of highly relevant applications. Many of them are documented in the proceedings of the four international conferences [7], [2], [6], and [5]. Source transformation tools for AD (see, for example, ADIFOR [3], ADIC [4], the differentiation-enabled NAGWare Fortran 95 compiler [16], TAPENADE [12], TAF [8], and OpenAD [20]) that provide a basic forward mode generate tangent-linear code as an augmentation of the code list by statements for computing directional derivatives as illustrated by the following example which shows the tangent-linear code of "$y = \sin(x * 2)$".

$$\begin{array}{ll}
\dot{v}_1 = \dot{x} & v_1 = x \\
\dot{v}_2 = 0 & v_2 = 2 \\
\dot{v}_3 = \dot{v}_1 * v_2 + v_1 * \dot{v}_2 & v_3 = v_1 * v_2 \\
\dot{v}_4 = \cos(v_3) * \dot{v}_3 & v_4 = \sin(v_3) \\
\dot{y} = \dot{v}_4 & y = v_4 \quad .
\end{array}$$

Less trivial examples follow after explaining the syntax-directed approach to the generation of tangent-linear code in the following section.

3 Syntax-Directed Tangent-Linear Codes

The main conceptual issues of the syntax-directed construction of tangent-linear codes can be discussed in the context of simple sequences of scalar assignments as defined in Definition 1. The presence of control-flow structures – both intra- (loops, branches, or arbitrary jumps in the form of `goto` statements) and inter-procedural (subroutine calls and calls of user-defined functions) – adds little to the algorithmic details behind the proposed method as explained in Section 3.4 and in Section 3.5.

Definition 1. *A* straight-line program (SLP) *is a sequence of scalar assignments described by the context-free grammar $G = (N, T, P, s)$ with nonterminal symbols*

$$N = \left\{ \; s \textit{ (straight-line program)} \quad a \textit{ (assignment)} \quad e \textit{ (expression)} \; \right\}$$

terminal symbols

$$T = \left\{ \begin{array}{ll} V & (\textit{program variables; see line 16 in Section A.2, Listing 1.5}) \\ C & (\textit{constants; line 21}) \\ F & (\textit{unary intrinsic; line 15}) \\ O & (\textit{binary operator; line 27}) \\ \multicolumn{2}{l}{\ , \ ; \ ) \ (\quad (\textit{remaining single character tokens; line 27})} \end{array} \right\}$$

start symbol s, *and production rules*

$$P = \left\{ \begin{array}{lll} (P1) & s :: a & (\textit{see line 32 in Section A.3, Listing 1.6}) \\ (P2) & s :: as & (\textit{line 36}) \\ (P3) & a :: V = e; & (\textit{line 68}) \\ (P4) & e :: V & (\textit{line 102}) \\ (P5) & e :: C & (\textit{line 109}) \\ (P6) & e :: F(e) & (\textit{line 94}) \\ (P7) & e :: eOe & (\textit{line 76 and line 85}) \end{array} \right\}$$

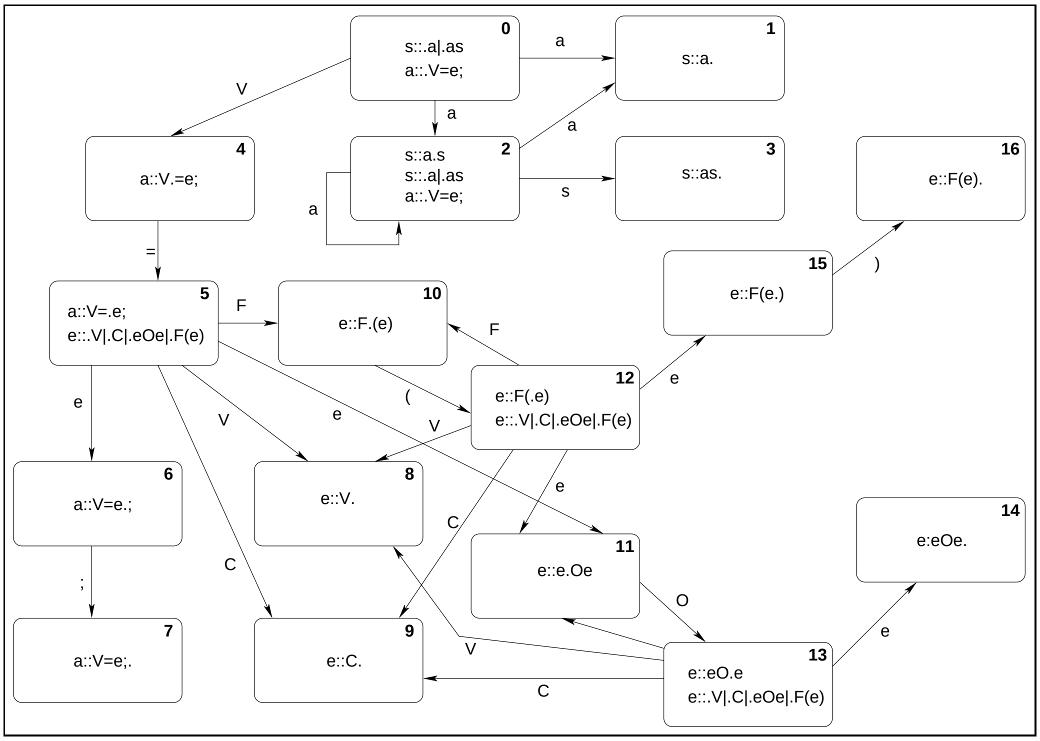

Fig. 2. Characteristic finite automaton of SLPs

References to the source code of a simple proof-of-concept implementation (see also Section 4) are given for terminal symbols (references into the grammar configuration file of the lexical analyzer) and production rules (references into the grammar configuration file of the syntax analyzer).

By considering unary intrinsics and binary arithmetic operators we cover a large fraction of the elemental functions provided by modern programming languages. A generalization to include other potentially relevant cases such as unary operators or intrinsics that take more than one argument complicates the grammar and thus the parsing procedure while not adding anything conceptual to the approach of generating tangent-linear code by syntax-directed translation. Hence we decided to limit the amount of syntax to a minimum that allows us to introduce the theoretical idea behind syntax-directed compilation of tangent-linear code and to verify the numerical correctness of the generated code. The latter is done by comparing the values of the directional derivatives with approximations obtained by finite difference quotients as described in Section 4.

Programs come as sequences of ASCII characters that need to be transformed into a structured form to allow for semantic transformation according to well-defined syntactic rules. A lexical analyzer is used to partition the input code into a sequence of tokens that correspond to terminal symbols in Definition 1. The correctness of the input in form of the token stream is verified by a shift-reduce parser. See [1] for the technical details. We use an LALR(1)-parsing algorithm based on a *push-down automaton (PDA)* $A = (\alpha, \sigma)$ that consists of a *characteristic finite automaton (CFA)* α and a *shift-reduce stack (SRS)* σ to store states of α. The CFA $\alpha = (V_\alpha, E_\alpha)$ of an SLP is defined as follows

$$V_\alpha = \left\{ \begin{array}{l} 0 = [a|as|V = e;] \quad 1 = [\mathbf{P1}] \quad 2 = [s|a|as|V = e;] \\ 3 = [\mathbf{P2}] \quad 4 = [= e;] \quad 5 = [e;|V|C|F(e)|eOe] \\ 6 = [;] \quad 7 = [\mathbf{P3}] \quad 8 = [\mathbf{P4}] \\ 9 = [\mathbf{P5}] \quad 10 = [(e)] \quad 11 = [Oe] \\ 12 = [e)|V|C|F(e)|eOe] \quad 13 = [e|V|C|F(e)|eOe] \quad 14 = [\mathbf{P6}] \\ 15 = [)] \quad 16 = [\mathbf{P7}] \end{array} \right\} \tag{10}$$

$$E_\alpha = \left\{ \begin{array}{l} (0,1)[a], (0,2)[a], (0,4)[V] \\ (2,1)[a], (2,2)[a], (2,3)[s] \\ (4,5)[=] \\ (5,6)[e], (5,8)[V], (5,9)[C], (5,10)[F], (5,10)[e] \\ (6,7)[;] \\ (10,12)[(] \\ (11,13)[O] \\ (12,8)[V], (12,9)[C], (12,10)[F], (12,11)[e], (12,15)[e] \\ (13,8)[V], (13,9)[C], (13,10)[F], (13,11)[e], (13,14)[e] \\ (15,16)[)] \end{array} \right\} \tag{11}$$

The symbol "|" is used to separate alternative parts of right-hand sides of the production rules that are associated with the CFA vertices. The CFA is nondeterministic. A graphical representation is given in Figure 2. A lookahead of one on the input token stream is required to make it deterministic. The lookahead is realized by the function $\lambda \equiv \lambda(f, i) \in T$ that returns for the given position i in an SLP the next token.

3.1 Example

We use the PDA to construct the abstract syntax tree (AST) $\mathbf{A} = (\mathbf{V}, \mathbf{E})$ of the assignment "$y = \sin(x * 2);$" which is transformed by the lexical analyzer into the token stream

$$"V = F(VOC);" \quad .$$

A representation of the parse table is shown in Table 1. The AST **A** can be

	σ	$i \in V_\alpha$	Read	Action	λ	Line
1		0	V	S		68
2	0	4	$V =$	S		68
3	0,4	5	$V = F$	S		94
4	0,4,5	10	$V = F($	S		94
5	0,4,5,10	12	$V = F(V$	S		94
6	0,4,5,10,12	8		R($P4$)		102
7	0,4,5,10	12	V=F(e	S	O	94
8	0,4,5,10,12	11	V=F(eO	S		76
9	0,4,5,10,12,11	13	V=F(eOC	S		76
10	0,4,5,10,12,11,13	9		R($P5$)		109
11	0,4,5,10,12,11	13	V=F(eOe	S	)	76
12	0,4,5,10,12,11,13	14		R($P7$)		76
13	0,4,5,10	12	V=F(e	S	)	94
14	0,4,5,10,12	15	V=F(e)	S		94
15	0,4,5,10,12,15	16		R($P6$)		94
16	0,4	5	V=e	S	;	68
17	0,4,5	6	V=e;	S		68
18	0,4,5,6	7		R($P3$)		68
19		0	a	S	$\varnothing$	32
20	0	1		R($P1$)		32
21		0	s			23

Table 1. Extended Parse Table: A total of 21 shift (S in "Action" column) or reduce (R in "Action" column together with the production rule that is matched) operations are performed. We show the contents of the stack σ, the current state i in the characteristic finite automaton, the string read so far, the lookahead token λ, and the corresponding line number in the parser configuration file in Section A.3, Listing 1.6. In row 19 we get empty lookahead $\varnothing$ as a result of the sequence of assignments containing only a single element.

extracted by applying the reduce operations in reverse order. It is shown in Figure 3.

3.2 Code List

The first step of the compilation of tangent-linear code is to break complex right-hand sides of assignments down to elemental assignments by constructing statement-level code lists. Initializing $j = \nu = 1$ we extend the production rules by custom reduce actions as shown in Figure 4. We use the increment operator $j{+}{+}$ as defined in the C standard. Otherwise, the addition " $+$ " serves as a string concatenation operator. All actions performed are purely symbolic in the sense that they derive new strings from given ones according to well-defined rules stated in Section 2. The attribute a that is associated with all vertices in the AST is synthesized via concatenation of strings during the bottom-up parsing.

$$
\begin{array}{llllll}
s\ (6) & \rightarrow\ a\ (5) & & & & \\
 & & |\rightarrow\ V(\text{"}y\text{"}) & & & \\
 & & |\rightarrow = & & & \\
 & & |\rightarrow\ e\ (4) & & & \\
 & & & |\rightarrow\ F(\text{"}\sin\text{"}) & & \\
 & & & |\rightarrow\ (& & \\
 & & & |\rightarrow\ e\ (3) & & \\
 & & & & |\rightarrow\ e\ (1)\ \rightarrow\ V(\text{"}x\text{"}) & \\
 & & & & |\rightarrow\ O(\text{"}*\text{"}) & \\
 & & & & |\rightarrow\ e\ (2)\ \rightarrow\ C(\text{"}2\text{"}) & \\
 & & & |\rightarrow\ ) & &
\end{array}
$$

Fig. 3. AST of "$V = F(VOC);$".

$$
P_{cl} = \left\{
\begin{array}{ll}
(P1) & \mathbf{v}_\nu.a = \mathbf{v}_{\nu-1}.a \\
 & \qquad \nu{+}{+} \\
(P2) & \mathbf{v}_\nu.a = \mathbf{v}_{\mu_1}.a + \mathbf{v}_{\mu_2}.a \\
 & \qquad \text{where } \mathbf{v}_{\mu_1} \hat{=} a \text{ and } \mathbf{v}_{\mu_2} \hat{=} s \text{ in } as \\
 & \qquad \nu{+}{+} \\
(P3) & \mathbf{v}_\nu.a = \mathbf{v}_{\nu-1}.a + V.a + \text{"} = \text{"} + \text{"}v_{[\mathbf{v}_{\nu-1}.j]}\text{"} + \text{"};\text{"} \\
 & \qquad c = 1;\ \nu{+}{+} \\
(P4) & \mathbf{v}_\nu.a = \text{"}v_{[c]}\text{"} + \text{"} = \text{"} + V.a + \text{"};\text{"} \\
 & \mathbf{v}_\nu.j = c{+}{+} \\
 & \qquad \nu{+}{+} \\
(P5) & \mathbf{v}_\nu.a = \text{"}v_{[c]}\text{"} + \text{"} = \text{"} + C.a + \text{"};\text{"} \\
 & \mathbf{v}_\nu.j = c{+}{+} \\
 & \qquad \nu{+}{+} \\
(P6) & \mathbf{v}_\nu.a = \mathbf{v}_{\nu-1}.a + \text{"}v_{[c]}\text{"} + \text{"} = \text{"} \\
 & \qquad + F.a + \text{"}(\text{"} + \text{"}v_{[\mathbf{v}_{\nu-1}.j]}\text{"} + \text{"});\text{"} \\
 & \mathbf{v}_\nu.j = c{+}{+} \\
 & \nu{+}{+} \\
(P7) & \mathbf{v}_\nu.a = \mathbf{v}_{\mu_1}.a + \mathbf{v}_{\mu_2}.a + \text{"}v_{[c]}\text{"} + \text{"} = \text{"} \\
 & \qquad + \text{"}v_{[\mathbf{v}_{\mu_1}.j]}\text{"} + O.a + \text{"}v_{[\mathbf{v}_{\mu_2}.j]}\text{"} + \text{"};\text{"} \\
 & \qquad \text{where } \mathbf{v}_{\mu_1} \hat{=} e^1 \text{ and } \mathbf{v}_{\mu_2} \hat{=} e^2 \text{ in } e^1 O e^2 \\
 & \mathbf{v}_\nu.j = c{+}{+} \\
 & \nu{+}{+}
\end{array}
\right\}
$$

Fig. 4. Production rules for syntax-directed compilation of assignment-level code lists for SLPs. We use the notation "$\hat{=}$" in the sense of "corresponds to," that is for example, $\mathbf{v}_{\mu_1}$ is the vertex in the AST that corresponds to the first non-terminal on the right-hand side of rule (P2).

It contains the code list of the previously parsed part of the code, respectively. The second attribute j is used to store the index of the corresponding code list variable inside of expressions on the right-hand side of assignments. Expressions in square brackets depend on one of the algorithm's counters c or ν. They need to be transformed into the corresponding string, for example, "$v_{[\mathbf{v}_{\nu-1}.j]}$" becomes "$v_3$" if the value of the j-attribute of $\mathbf{v}_{\nu-1} \in \mathbf{V}$ is equal to 3.

To improve the structure of the output one may think of a semicolon at the end of an assignment as followed by a newline character (see implementation in Section 4). For example, the application of the symbolic rules in Figure 4 to "$y = \sin(x * 2);$" proceeds as follows.

(P4 with $\nu = 1$, $[c] = 1$, and $V.a =$ "x")

(P5 with $\nu = 2$, $[c] = 2$, and $C.a =$ "2")

(P7 with $\nu = 3$, $\mu_1 = 1$, $\mu_2 = 2$,
$\mathbf{v}_{\mu_1}.a =$ "$v_1 = x;$", $\mathbf{v}_{\mu_2}.a =$ "$v_2 = 2;$", $[c] = 3$,
$[\mathbf{v}_{\mu_1}.j] = 1$, $O.a =$ "$*$", and $[\mathbf{v}_{\mu_2}.j] = 2$)

(P6 with $\nu = 4$, $\mathbf{v}_{\nu-1}.a =$ "$v_1 = x;$
$v_2 = 2;$
$v_3 = v_1 * v_2;$",
$[c] = 4$, $F.a =$ " sin ", and $[\mathbf{v}_{\nu-1}.j] = 3$)

(P3 with $\nu = 5$, $\mathbf{v}_{\nu-1}.a =$ "$v_1 = x;$
$v_2 = 2;$
$v_3 = v_1 * v_2;$
$v_4 = \sin(v_3);$",
$V.a =$ "y", and $[\mathbf{v}_{\nu-1}.j] = 4$)

(P1 with $\nu = 6$, $\mathbf{v}_{\nu-1}.a =$ "$v_1 = x;$
$v_2 = 2;$
$v_3 = v_1 * v_2;$
$v_4 = \sin(v_3);$
$y = v_4;$")

The complete AST is available after the last reduction operation ($P1$) that is performed by the shift-reduce parsing procedure in Table 1. Hence, the synthesized attribute $\mathbf{v}_6.a$ contains the entire code list in form of a copy of $\mathbf{v}_5.a$.

3.3 Tangent-Linear SLP

Tangent-linear code is constructed by augmenting the code list locally with statements for computing directional derivatives of the elemental functions. The production rules for syntax-directed compilation of tangent-linear SLPs are shown in Figure 5. We use the notation $V.\dot{a}$ to refer to the string that marks the directional derivative component of the variable marked by the string $V.a$, that is, if

$$
P_{tlc} = \left\{ \begin{array}{ll}
(P1) & \mathbf{v}_{\nu}.a = \mathbf{v}_{\nu-1}.a \\
& \quad \nu{+}{+} \\
& \quad \text{(See lines 32–35 in Section A.3, Listing 1.6.)} \\
(P2) & \mathbf{v}_{\nu}.a = \mathbf{v}_{\mu_1}.a + \mathbf{v}_{\mu_2}.a \\
& \qquad\qquad \text{where } \mathbf{v}_{\mu_1} \hat{=} a \text{ and } \mathbf{v}_{\mu_2} \hat{=} s \text{ in } as \\
& \quad \nu{+}{+} \\
& \quad \text{(See lines 36 – 42.)} \\
(P3) & \mathbf{v}_{\nu}.a = \mathbf{v}_{\nu-1}.a + V.\dot{a} + ''='' + ''\dot{v}_{[\mathbf{v}_{\nu-1}.j]}'' + '';'' \\
& \quad + V.a + ''='' + ''v_{[\mathbf{v}_{\nu-1}.j]}'' + '';'' \\
& \quad c = 1;\ \nu{+}{+} \\
& \quad \text{(See lines 68 – 75.)} \\
(P4) & \mathbf{v}_{\nu}.a = ''\dot{v}_{[c]}'' + ''='' + V.\dot{a} + '';'' \\
& \quad + ''v_{[c]}'' + ''='' + V.a + '';'' \\
& \mathbf{v}_{\nu}.j = c{+}{+} \\
& \quad \nu{+}{+} \\
& \quad \text{(See lines 102 – 108.)} \\
(P5) & \mathbf{v}_{\nu}.a = ''\dot{v}_{[c]}'' + ''=0;'' \\
& \quad + ''v_{[c]}'' + ''='' + C.a + '';'' \\
& \mathbf{v}_{\nu}.j = c{+}{+} \\
& \quad \nu{+}{+} \\
& \quad \text{(See lines 109 – 114.)} \\
(P6) & \mathbf{v}_{\nu}.a = \mathbf{v}_{\nu-1}.a + ''\dot{v}_{[c]}'' + ''='' \\
& \quad + \dfrac{\partial F.a}{\partial ''v_{[\mathbf{v}_{\nu-1}.j]}''} + ''('' + ''\dot{v}_{[\mathbf{v}_{\nu-1}.j]}'' + '');'' \\
& \quad + ''v_{[c]}'' + ''='' \\
& \quad + F.a + ''('' + ''v_{[\mathbf{v}_{\nu-1}.j]}'' + '');'' \\
& \mathbf{v}_{\nu}.j = c{+}{+} \\
& \nu{+}{+} \\
& \quad \text{(See lines 94 – 101.)} \\
(P7) & \mathbf{v}_{\nu}.a = \mathbf{v}_{\mu_1}.a + \mathbf{v}_{\mu_2}.a + ''\dot{v}_{[c]}'' + ''='' \\
& \quad + ''\dot{v}_{[\mathbf{v}_{\mu_1}.j]}'' + ''*'' + \dfrac{\partial O.a}{\partial ''v_{[\mathbf{v}_{\mu_1}.j]}''} + ''+'' \\
& \quad + \dfrac{\partial O.a}{\partial ''v_{[\mathbf{v}_{\mu_2}.j]}''} + ''*'' + ''\dot{v}_{[\mathbf{v}_{\mu_2}.j]}'' + '';'' + ''v_{[c]}'' \\
& \quad + ''='' + ''v_{[\mathbf{v}_{\mu_1}.j]}'' + O.a + ''v_{[\mathbf{v}_{\mu_2}.j]}'' + '';'' \\
& \qquad\qquad \text{where } \mathbf{v}_{\mu_1} \hat{=} e^1 \text{ and } \mathbf{v}_{\mu_2} \hat{=} e^2 \text{ in } e^1 O e^2 \\
& \mathbf{v}_{\nu}.j = c{+}{+} \\
& \nu{+}{+} \\
& \quad \text{(See lines 76 – 84 and 85 – 93.)}
\end{array} \right\}
$$

Fig. 5. Production rules for syntax-directed compilation of tangent-linear SLPs.

$V.a = "x"$, then $V.\dot{a} = "\dot{x}"$. In (P6) the symbolic transformation

$$\frac{\partial F.a}{\partial \, "v_{[\mathbf{v}_{\nu-1}.j]}"}$$

is defined according to the differentiation rules of the elemental functions, for example, if $F.a = "\sin"$ and $"v_{[\mathbf{v}_{\nu-1}.j]}" = "v_1"$, then

$$\frac{\partial F.a}{\partial \, "v_{[\mathbf{v}_{\nu-1}.j]}"} = \frac{\partial \, "\sin(v_1)"}{\partial \, "v_1"} = "\cos(v_1)" \quad .$$

We assume that partial derivatives of unary intrinsics are again unary functions of the same argument. The symbolic rule can be modified to account for deviations, for example,

$$\frac{\partial \, "\exp"}{\partial \, "v_{[\mathbf{v}_{\nu-1}.j]}"} = "v_{[\mathbf{v}_{\nu}.j]}" \quad .$$

Similarly, in (P7)

$$\frac{\partial O.a}{\partial "v_{[\mathbf{v}_{\mu_1}.j]}"} = "v_{[\mathbf{v}_{\mu_2}.j]}"$$

if $O.a = " * "$.

The generation of the tangent-linear code for $"y = \sin(x * 2);"$ proceeds as follows.

(P4 with $\nu = 1$, $[c] = 1$, and $V.a = "x"$)

(P5 with $\nu = 2$, $[c] = 2$, and $C.a = "2"$)

(P7 with $\nu = 3$, $\mu_1 = 1$, $\mu_2 = 2$, $[c] = 3$,
$\mathbf{v}_{\mu_1}.a = "\dot{v}_1 = \dot{x};\ v_1 = x;"$,
$\mathbf{v}_{\mu_2}.a = "\dot{v}_2 = 0;\ v_2 = 2;"$,
$[\mathbf{v}_{\mu_1}.j] = 1$, $O.a = " * "$, and $[\mathbf{v}_{\mu_2}.j] = 2$)

(P6 with $\nu = 4$, $[c] = 4$,
$\mathbf{v}_{\nu-1}.a = "\dot{v}_1 = \dot{x};\ v_1 = x;$
$\dot{v}_2 = 0;\ v_2 = 2;$
$\dot{v}_3 = \dot{v}_1 * v_2 + v_1 * \dot{v}_2;\ v_3 = v_1 * v_2;"$,
$F.a = "\sin"$, and $[\mathbf{v}_{\nu-1}.j] = 3$)

(P3 with $\nu = 5$,
$\mathbf{v}_{\nu-1}.a = "\dot{v}_1 = \dot{x};\ v_1 = x;$
$\dot{v}_2 = 0;\ v_2 = 2;$
$\dot{v}_3 = \dot{v}_1 * v_2 + v_1 * \dot{v}_2;\ v_3 = v_1 * v_2;$
$\dot{v}_4 = \cos(v_3) * \dot{v}_3;\ v_4 = \sin(v_3);"$,
$V.a = "y"$, and $[\mathbf{v}_{\nu-1}.j] = 4$)

(P1 with $\nu = 6$,
$\mathbf{v}_{\nu-1}.a = "\dot{v}_1 = \dot{x};\ v_1 = x;$
$\dot{v}_2 = 0;\ v_2 = 2;$

$$\begin{array}{l}\dot{v}_3 = \dot{v}_1 * v_2 + v_1 * \dot{v}_2;\;\; v_3 = v_1 * v_2;\\ \dot{v}_4 = \cos(v_3) * \dot{v}_3;\;\; v_4 = \sin(v_3);\\ \dot{y} = \dot{v}_4;\;\; y = v_4;")\end{array}$$

Again, the synthesized attribute $\mathbf{v}_6.a$ contains the tangent-linear code after the last reduction performed in Table 1.

3.4 Intraprocedural Flow of Control

The flow of control of the tangent-linear program is the same as for the original program. Hence, any control flow statements such as loops and branches can be treated in a straight-forward fashion by simply unparsing them. In Section A we present a proof-of-concept implementation of a syntax-directed tangent-linear code generator for single subroutines using the GNU tools `flex` and `bison`. The functionality of the software is documented in Section 4. For brevity we refrain from the presentation of a corresponding formalism that does not add anything conceptually new. Obviously, the syntactic richness of the language accepted by both the scanner and the parser can be extended easily, for example, by other control-flow structures and conditional expressions.

The presence of control flow statements has a more significant impact on the syntax-directed compilation of adjoint codes which will be discussed in the second part of this series of publications.

3.5 Interprocedural Flow of Control

The reversal of interprocedural flow of control in the context of adjoint codes generated by syntax-directed translation represents a separate topic that will be discussed in the third part of this series of papers. For tangent-linear codes the presence of subroutine calls requires their replacement with calls of the corresponding tangent-linear subroutine. For example, if $\texttt{foo}(\mathbf{x},\mathbf{y})$ implements F from Equation (1), then its call is replaced with $\dot{\texttt{foo}}(\mathbf{x},\dot{\mathbf{x}},\mathbf{y},\dot{\mathbf{y}})$. No conceptual difficulties arise in the context of syntax-directed translation.

3.6 Vector Mode, etc.

The extension to vector mode that computes

$$\dot{\mathbf{y}} = F'(\mathbf{x}) \cdot \dot{\mathbf{x}}$$

for $\dot{\mathbf{x}} \in I\!R^{n\times l}$ and, hence, $\dot{\mathbf{y}} \in I\!R^{m\times l}$ is trivial. Any tangent-linear assignment

$$\dot{v}^j = \sum_{i\prec j} c_{j,i}\dot{v}^i$$

generated so far is simply replaced by the corresponding loop over the l components of the vectors $\dot{v}^j$ and $\dot{v}^i$ as follows

$$\dot{v}^j_k = \sum_{i\prec j} c_{j,i}\dot{v}^i_k \quad k = 1,\ldots,l \quad .$$

Obviously, this simple modification can be done in the context of syntax-directed translation. The same statement applies to the generation of tangent-linear code for the propagation of sparse gradients in sparse forward mode AD (see ADIFOR's SparsLinC library [3]) and to codes for computing higher derivatives by propagation of truncated Taylor series [10]. In general, any transformation that simply augments the code list with additional statements whose syntax depends only on the code parsed so far fits into the current framework. However, we do not want to imply that the practical realization of these concepts needs to be straight-forward. Often enough the technical details require sophisticated algorithmic solutions and a high level of proficiency in computer programming.

4 Implementation

The source of our simple proof-of-concept implementation (called `sdtlc` for syntax-directed tangent-linear code compiler) that uses the compiler tools `flex`[6] and `bison`[7] is shown in Section A. It is meant to serve as a starting point for further development of a syntax-directed tangent-linear code compiler that covers larger fractions of C and possibly C++. Moreover, given an LALR(1) grammar of some imperative programming language the generation of tangent-linear code becomes conceptually easy. We provide the source code on the project's website to support potential other development projects in this direction.

In the following we present a small case study that is supposed to illustrate the current functionality of `sdtlc`.

Listing 1.1 shows a small input file that needs to be transformed into tangent-linear code.

Listing 1.1. `test.in`

```
t=0;
while (x<t) {
  if (x<y) {
    x=y+1;
  }
  x=sin(x*y);
}
```

We call `sdtlc test.in > test.out` to obtain the output in Listing 1.2.

Listing 1.2. `test.out`

```
double v1, v1_;
double v2, v2_;
double v3, v3_;
double v4, v4_;
v1_=0; v1=0;
t_=v1_; t=v1;
while (x<t) {
if (x<y) {
v1_=y_; v1=y;
v2_=0; v2=1;
v3_=v1_+v2_; v3=v1+v2;
x_=v3_; x=v3;
}
```

[6] http://www.gnu.org/software/flex/

[7] http://www.gnu.org/software/bison/

```
v1_=x_; v1=x;
v2_=y_; v2=y;
v3_=v1_*v2+v1*v2_; v3=v1*v2;
v4_=cos(v3)*v3_; v4=sin(v3);
x_=v4_; x=v4;
}
```

To verify the correctness of the transformation we provide a driver that compares the values of the two gradient entries as computed by the tangent-linear code with an approximation obtained by applying forward finite differences. The driver contains wrappers for the original code and the tangent-linear code in the form of the subroutines `test` and `test_`.

Listing 1.3. `test.cpp`

```
#include <cmath>
#include <iostream>

using namespace std;

void test ( double &x, double y) {
double t;
#include "test.in"
}

void test_ ( double &x, double& x_, double y, double y_) {
double t,t_;
#include "test.out"
}

int main() {

  cout << "finite differences:" << endl;
  double h=1e-6, x=-.5, y=-5., x_=x+h, y_=y;
  test(x,y);
  test(x_,y_);
  cout << "dx/dx=" << (x_-x)/h << endl;

  x=-.5, x_=x, y_=y+h;
  test(x,y);
  test(x_,y_);
  cout << "dx/dy=" << (x_-x)/h << endl;

  cout << "tangent-linear code:" << endl;
  x=-.5, x_=1., y_=0.;
  test_(x,x_,y,y_);
  cout << "dx/dx=" << x_ << endl;

  x=-.5, x_=0., y_=1.;
  test_(x,x_,y,y_);
  cout << "dx/dy=" << x_ << endl;

  return 0;
}
```

The result of running the corresponding executable is shown below.

```
finite differences:
dx/dx=4.00571
dx/dy=0.400572
```

```
tangent-linear code:
dx/dx=4.00572
dx/dy=0.400572
```

Additional test cases can be found on the project's website.

5 Conclusion and Outlook

This paper discussed the automatic generation of tangent-linear code by single-pass source transformation AD without building an internal representation of the program. For the presentation of the formalism we have focused on assignment statements as the relevant syntactical units for semantic transformation. Control-flow statements are simply unparsed. The proposed approach is relatively simple to implement and generalize for full programming languages.

A substantial draw-back of this method is the missing static program analysis. An internal representation in form of a control flow graph on top of an implementation of the AST is required. The source transformation problem becomes much more complicated as it involves the development of a complete compiler front-end in addition to domain-specific data flow analyses [11]. However, it appears to be useful to apply the syntax-directed approach as much as possible during the first pass of a source transformation AD tool. The internal representation that becomes the subject of program analysis could already be that of a tangent-linear code. Thus, the optimization of this code amounts to the application of state-of-the-art compiler optimizations driven by domain-specific information, such as vanishing total derivatives of passive variables [11]. Preaccumulation of partial derivatives at the statement or basic block levels can be achieved by extending the syntax-directed approach by assembly of statically available parts of the code list (see [3], [15], and [18]). Pushing the limits of this method is the subject of ongoing research.

In part II of this series of papers we consider the syntax-directed generation of adjoint codes. A reversal of the data flow and hence also of the control flow is required. Both can be achieved during a single-pass compilation, leading to codes that enable us to compute derivatives independent of the dimension of the input space. Established applications for adjoint numerical codes are large-scale nonlinear optimization and numerical inverse methods in general. See [5] for a collection of recent activities in these areas.

Bibliography

[1] A. Aho, R. Sethi, and J. Ullman. *Compilers. Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[2] M. Berz, C. Bischof, G. Corliss, and A. Griewank, editors. *Computational Differentiation: Techniques, Applications, and Tools*, Proceedings Series. SIAM, 1996.

[3] C. Bischof, A. Carle, P. Khademi, and A. Maurer. The ADIFOR 2.0 system for automatic differentiation of Fortran 77 programs. *IEEE Comp. Sci. & Eng.*, 3(3):18–32, 1996.

[4] C. Bischof, L. Roh, and A. Mauer. ADIC — An extensible automatic differentiation tool for ANSI-C. *Software: Practice and Experience*, 27(12):1427–1456, 1997.

[5] M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, editors. *Automatic Differentiation: Applications, Theory, and Tools*, volume 50 of *Lecture Notes in Computational Science and Engineering*. Springer, 2005.

[6] G. Corliss, C. Faure, A. Griewank, L. Hascoët, and U. Naumann, editors. *Automatic Differentiation of Algorithms – From Simulation to Optimization*. Springer, 2002.

[7] G. Corliss and A. Griewank, editors. *Automatic Differentiation: Theory, Implementation, and Application*, Proceedings Series. SIAM, 1991.

[8] R. Giering and T. Kaminski. Applying TAF to generate efficient derivative code of Fortran 77-95 programs. *Proceedings in Applied Mathematics and Mechanics*, 2(1):54–57, 2003.

[9] A. Griewank. *Evaluating Derivatives. Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Applied Mathematics. SIAM, 2000.

[10] A. Griewank, J. Utke, and A. Walther. Evaluating higher derivative tensors by forward propagation of univariate Taylor series. *Mathematics of Computation*, 69:1117–1130, 2000.

[11] L. Hascoët, U. Naumann, and V. Pascual. "To be recorded" analysis in reverse-mode automatic differentiation. *Future Generation Computer Systems*, 21(8):1401–1417, 2005.

[12] L. Hascoët and V. Pascual. Tapenade 2.1 user's guide. Technical report 300, INRIA, 2004.

[13] M. Heath. *Scientific Computing. An Introductory Survey*. McGraw-Hill, 1998.

[14] C. Kelley. *Solving Nonlinear Equations with Newton's Method*. SIAM, 2003.

[15] U. Naumann. On optimal jacobian accumulation for single-expression-use programs. Preprint ANL-MCS/P944-0402, Mathematics and Computer Science Division, Argonne National Laboratory, 2002.

[16] U. Naumann and J. Riehme. A differentiation-enabled Fortran 95 compiler. *ACM Trans. Math. Software*, 2005. (to appear, ref. TOMS-2003-0052).

[17] L. Rall. *Automatic Differentiation: Techniques and Applications*, volume 120 of *LNCS*. Springer, 1981.

[18] J. Utke. Flattening basic blocks. In *[5]*. 2005.

[19] R. Wengert. A simple automatic derivative evaluation program. *Comm. ACM*, 7:463–464, 1964.

[20] C. Wunsch, C. Hill, P. Heimbach, U. Naumann, J. Utke, M. Fagan, and N. Tallent. OpenAD. Preprint ANL/MCS-P1230-0205, Argonne National Laboratory, February 2005.

A A Proof-of-Concept Implementation

A.1 ast.h

The parser generator `bison` is told to use AST vertices of type astNodeType to enable the propagation of the synthesized attribute a based on code list indexes j.

Listing 1.4. ast.h

```
typedef struct {
  char* a;
  int j;
} astNodeType;

#define YYSTYPE astNodeType
```

A.2 scanner.l

Listing 1.5. scanner.l

```
%{
#include "ast.h"
#include "parser.tab.h"
%}

whitespace         [ \t\n]+
symbol             [a-z]
const              [0-9]

%%

{whitespace} { }
"if" { return IF; }
"while" { return WHILE; }
"sin" { return SIN; }
{symbol} {
  yylval.a = (char*)malloc(2*sizeof(char));
  strcpy(yylval.a,yytext);
  return SYMBOL;
}
{const} {
  yylval.a = (char*)malloc((strlen(yytext)+1)*sizeof(char));
  strcpy(yylval.a,yytext);
  return CONSTANT;
}

. { return yytext[0]; }

%%
```

```

void lexinit(FILE *source)
{
  yyin=source;
}
```

A.3 parser.y

Listing 1.6. parser.y

```
%{

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "ast.h"

extern int yylex();
extern void lexinit(FILE*);

static int c, cmax=1;

%}

%token SYMBOL CONSTANT SIN IF WHILE

%left '+'
%left '*'

%%

code : sequence_of_statements
  {
    // print declarations of auxillary variables
    for (c=1;c<cmax;c++) printf("double v%d, v%d_;\n",c,c);
    // print tangent-linear code
    printf("%s",$1.a);
    free($1.a);
  }
  ;
sequence_of_statements : statement
  {
    $$=$1;
  }
  | statement sequence_of_statements
  {
    $$.a=(char*)malloc((strlen($1.a)+strlen($2.a)+1)*sizeof(char));
    sprintf($$.a,"%s%s",$1.a,$2.a);
    free($2.a); free($1.a);
  }
  ;
statement : assignment { $$=$1; }
  | if_statement { $$=$1; }
  | while_statement { $$=$1; }
  ;
if_statement : IF '(' condition ')' '{' sequence_of_statements '}'
  {
    $$.a=(char*)malloc((strlen($3.a)+strlen($6.a)+11)*sizeof(char));
    sprintf($$.a,"if (%s) {\n%s}\n",$3.a,$6.a);
```

```
    free($3.a); free($6.a);
  }
  ;
while_statement : WHILE '(' condition ')' '{' sequence_of_statements '}'
  {
    $$.a=(char*)malloc((strlen($3.a)+strlen($6.a)+14)*sizeof(char));
    sprintf($$.a,"while (%s) {\n%s}\n",$3.a,$6.a);
    free($3.a); free($6.a);
  }
  ;
condition : SYMBOL '<' SYMBOL
  {
    $$.a=(char*)malloc((strlen($1.a)+strlen($3.a)+2)*sizeof(char));
    sprintf($$.a,"%s<%s",$1.a,$3.a);
    free($1.a); free($3.a);
  }
  ;
assignment : SYMBOL '=' { c=1; } expression ';'
  {
    $$.a=(char*)malloc(
      (strlen($4.a)+2*strlen($1.a)+2*$4.j%10+13)*sizeof(char));
    sprintf($$.a,"%s%s_=v%d_; %s=v%d;\n",$4.a,$1.a,$4.j,$1.a,$4.j);
    free($1.a); free($4.a);
  }
  ;
expression : expression '*' expression
  {
    $$.a=(char*)malloc(
      (strlen($1.a)+strlen($3.a)+2*c%10+3*$1.j%10+3*$3.j%10+30)*sizeof(char));
    $$.j=c++; if (c>cmax) cmax=c;
    sprintf($$.a,"%s%sv%d_=v%d_*v%d+v%d*v%d_; v%d=v%d*v%d;\n",
            $1.a,$3.a,$$.j,$1.j,$3.j,$1.j,$3.j,$$.j,$1.j,$3.j);
    free($1.a); free($3.a);
  }
  | expression '+' expression
  {
    $$.a=(char*)malloc(
      (strlen($1.a)+strlen($3.a)+2*c%10+2*$1.j%10+2*$3.j%10+24)*sizeof(char));
    $$.j=c++; if (c>cmax) cmax=c;
    sprintf($$.a,"%s%sv%d_=v%d_+v%d_; v%d=v%d+v%d;\n",
      $1.a,$3.a,$$.j,$1.j,$3.j,$$.j,$1.j,$3.j);
    free($1.a); free($3.a);
  }
  | SIN '(' expression ')'
  {
    $$.a=(char*)malloc((strlen($3.a)+2*c%10+3*$3.j%10+30)*sizeof(char));
    $$.j=c++; if (c>cmax) cmax=c;
    sprintf($$.a,"%sv%d_=cos(v%d)*v%d_; v%d=sin(v%d);\n",
      $3.a,$$.j,$3.j,$3.j,$$.j,$3.j);
    free($3.a);
  }
  | SYMBOL
  {
    $$.a=(char*)malloc((2*c%10+2*strlen($1.a)+13)*sizeof(char));
    $$.j=c++; if (c>cmax) cmax=c;
```

```
    sprintf($$.a,"v%d_=%s_; v%d=%s;\n",$$.j,$1.a,$$.j,$1.a);
    free($1.a);
  }
  | CONSTANT
  {
    $$.a=(char*)malloc((2*c%10+strlen($1.a)+13)*sizeof(char));
    $$.j=c++; if (c>cmax) cmax=c;
    sprintf($$.a,"v%d_=0; v%d=%s;\n",$$.j,$$.j,$1.a);
    free($1.a);
  }
  ;

%%

int yyerror(char *msg) { printf("ERROR: %s \n",msg); return -1; }

int main(int argc, char* argv[])
{
  FILE *source_file = fopen(argv[1],"r");
  lexinit(source_file);
  yyparse();
  fclose(source_file);
  return 0;
}
```

Aachener Informatik-Berichte

This is a list of recent technical reports. To obtain copies of technical reports please consult http://aib.informatik.rwth-aachen.de/ or send your request to: Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Email: `biblio@informatik.rwth-aachen.de`

1987-01	*	Fachgruppe Informatik: Jahresbericht 1986
1987-02	*	David de Frutos Escrig, Klaus Indermark: Equivalence Relations of Non-Deterministic Ianov-Schemes
1987-03	*	Manfred Nagl: A Software Development Environment based on Graph Technology
1987-04	*	Claus Lewerentz, Manfred Nagl, Bernhard Westfechtel: On Integration Mechanisms within a Graph-Based Software Development Environment
1987-05	*	Reinhard Rinn: Über Eingabeanomalien bei verschiedenen Inferenzmodellen
1987-06	*	Werner Damm, Gert Döhmen: Specifying Distributed Computer Architectures in AADL*
1987-07	*	Gregor Engels, Claus Lewerentz, Wilhelm Schäfer: Graph Grammar Engineering: A Software Specification Method
1987-08	*	Manfred Nagl: Set Theoretic Approaches to Graph Grammars
1987-09	*	Claus Lewerentz, Andreas Schürr: Experiences with a Database System for Software Documents
1987-10	*	Herbert Klaeren, Klaus Indermark: A New Implementation Technique for Recursive Function Definitions
1987-11	*	Rita Loogen: Design of a Parallel Programmable Graph Reduction Machine with Distributed Memory
1987-12		J. Börstler, U. Möncke, R. Wilhelm: Table compression for tree automata
1988-01	*	Gabriele Esser, Johannes Rückert, Frank Wagner: Gesellschaftliche Aspekte der Informatik
1988-02	*	Peter Martini, Otto Spaniol: Token-Passing in High-Speed Backbone Networks for Campus-Wide Environments
1988-03	*	Thomas Welzel: Simulation of a Multiple Token Ring Backbone
1988-04	*	Peter Martini: Performance Comparison for HSLAN Media Access Protocols
1988-05	*	Peter Martini: Performance Analysis of Multiple Token Rings
1988-06	*	Andreas Mann, Johannes Rückert, Otto Spaniol: Datenfunknetze
1988-07	*	Andreas Mann, Johannes Rückert: Packet Radio Networks for Data Exchange
1988-08	*	Andreas Mann, Johannes Rückert: Concurrent Slot Assignment Protocol for Packet Radio Networks
1988-09	*	W. Kremer, F. Reichert, J. Rückert, A. Mann: Entwurf einer Netzwerktopologie für ein Mobilfunknetz zur Unterstützung des öffentlichen Straßenverkehrs
1988-10	*	Kai Jakobs: Towards User-Friendly Networking
1988-11	*	Kai Jakobs: The Directory - Evolution of a Standard
1988-12	*	Kai Jakobs: Directory Services in Distributed Systems - A Survey
1988-13	*	Martine Schümmer: RS-511, a Protocol for the Plant Floor

1988-14 * U. Quernheim: Satellite Communication Protocols - A Performance Comparison Considering On-Board Processing
1988-15 * Peter Martini, Otto Spaniol, Thomas Welzel: File Transfer in High Speed Token Ring Networks: Performance Evaluation by Approximate Analysis and Simulation
1988-16 * Fachgruppe Informatik: Jahresbericht 1987
1988-17 * Wolfgang Thomas: Automata on Infinite Objects
1988-18 * Michael Sonnenschein: On Petri Nets and Data Flow Graphs
1988-19 * Heiko Vogler: Functional Distribution of the Contextual Analysis in Block-Structured Programming Languages: A Case Study of Tree Transducers
1988-20 * Thomas Welzel: Einsatz des Simulationswerkzeuges QNAP2 zur Leistungsbewertung von Kommunikationsprotokollen
1988-21 * Th. Janning, C. Lewerentz: Integrated Project Team Management in a Software Development Environment
1988-22 * Joost Engelfriet, Heiko Vogler: Modular Tree Transducers
1988-23 * Wolfgang Thomas: Automata and Quantifier Hierarchies
1988-24 * Uschi Heuter: Generalized Definite Tree Languages
1989-01 * Fachgruppe Informatik: Jahresbericht 1988
1989-02 * G. Esser, J. Rückert, F. Wagner (Hrsg.): Gesellschaftliche Aspekte der Informatik
1989-03 * Heiko Vogler: Bottom-Up Computation of Primitive Recursive Tree Functions
1989-04 * Andy Schürr: Introduction to PROGRESS, an Attribute Graph Grammar Based Specification Language
1989-05 J. Börstler: Reuse and Software Development - Problems, Solutions, and Bibliography (in German)
1989-06 * Kai Jakobs: OSI - An Appropriate Basis for Group Communication?
1989-07 * Kai Jakobs: ISO's Directory Proposal - Evolution, Current Status and Future Problems
1989-08 * Bernhard Westfechtel: Extension of a Graph Storage for Software Documents with Primitives for Undo/Redo and Revision Control
1989-09 * Peter Martini: High Speed Local Area Networks - A Tutorial
1989-10 * P. Davids, Th. Welzel: Performance Analysis of DQDB Based on Simulation
1989-11 * Manfred Nagl (Ed.): Abstracts of Talks presented at the WG '89 15th International Workshop on Graphtheoretic Concepts in Computer Science
1989-12 * Peter Martini: The DQDB Protocol - Is it Playing the Game?
1989-13 * Martine Schümmer: CNC/DNC Communication with MAP
1989-14 * Martine Schümmer: Local Area Networks for Manufactoring Environments with hard Real-Time Requirements
1989-15 * M. Schümmer, Th. Welzel, P. Martini: Integration of Field Bus and MAP Networks - Hierarchical Communication Systems in Production Environments
1989-16 * G. Vossen, K.-U. Witt: SUXESS: Towards a Sound Unification of Extensions of the Relational Data Model

1989-17 * J. Derissen, P. Hruschka, M.v.d. Beeck, Th. Janning, M. Nagl: Integrating Structured Analysis and Information Modelling
1989-18 A. Maassen: Programming with Higher Order Functions
1989-19 * Mario Rodriguez-Artalejo, Heiko Vogler: A Narrowing Machine for Syntax Directed BABEL
1989-20 H. Kuchen, R. Loogen, J.J. Moreno Navarro, M. Rodriguez Artalejo: Graph-based Implementation of a Functional Logic Language
1990-01 * Fachgruppe Informatik: Jahresbericht 1989
1990-02 * Vera Jansen, Andreas Potthoff, Wolfgang Thomas, Udo Wermuth: A Short Guide to the AMORE System (Computing Automata, MOnoids and Regular Expressions)
1990-03 * Jerzy Skurczynski: On Three Hierarchies of Weak SkS Formulas
1990-04 R. Loogen: Stack-based Implementation of Narrowing
1990-05 H. Kuchen, A. Wagener: Comparison of Dynamic Load Balancing Strategies
1990-06 * Kai Jakobs, Frank Reichert: Directory Services for Mobile Communication
1990-07 * Kai Jakobs: What's Beyond the Interface - OSI Networks to Support Cooperative Work
1990-08 * Kai Jakobs: Directory Names and Schema - An Evaluation
1990-09 * Ulrich Quernheim, Dieter Kreuer: Das CCITT - Signalisierungssystem Nr. 7 auf Satellitenstrecken; Simulation der Zeichengabestrecke
1990-11 H. Kuchen, R. Loogen, J.J. Moreno Navarro, M. Rodriguez Artalejo: Lazy Narrowing in a Graph Machine
1990-12 * Kai Jakobs, Josef Kaltwasser, Frank Reichert, Otto Spaniol: Der Computer fährt mit
1990-13 * Rudolf Mathar, Andreas Mann: Analyzing a Distributed Slot Assignment Protocol by Markov Chains
1990-14 A. Maassen: Compilerentwicklung in Miranda - ein Praktikum in funktionaler Programmierung (written in german)
1990-15 * Manfred Nagl, Andreas Schürr: A Specification Environment for Graph Grammars
1990-16 A. Schürr: PROGRESS: A VHL-Language Based on Graph Grammars
1990-17 * Marita Möller: Ein Ebenenmodell wissensbasierter Konsultationen - Unterstützung für Wissensakquisition und Erklärungsfähigkeit
1990-18 * Eric Kowalewski: Entwurf und Interpretation einer Sprache zur Beschreibung von Konsultationsphasen in Expertensystemen
1990-20 Y. Ortega Mallen, D. de Frutos Escrig: A Complete Proof System for Timed Observations
1990-21 * Manfred Nagl: Modelling of Software Architectures: Importance, Notions, Experiences
1990-22 H. Fassbender, H. Vogler: A Call-by-need Implementation of Syntax Directed Functional Programming
1991-01 Guenther Geiler (ed.), Fachgruppe Informatik: Jahresbericht 1990
1991-03 B. Steffen, A. Ingolfsdottir: Characteristic Formulae for Processes with Divergence
1991-04 M. Portz: A new class of cryptosystems based on interconnection networks

1991-05 H. Kuchen, G. Geiler: Distributed Applicative Arrays
1991-06 * Ludwig Staiger: Kolmogorov Complexity and Hausdorff Dimension
1991-07 * Ludwig Staiger: Syntactic Congruences for w-languages
1991-09 * Eila Kuikka: A Proposal for a Syntax-Directed Text Processing System
1991-10 K. Gladitz, H. Fassbender, H. Vogler: Compiler-based Implementation of Syntax-Directed Functional Programming
1991-11 R. Loogen, St. Winkler: Dynamic Detection of Determinism in Functional Logic Languages
1991-12 * K. Indermark, M. Rodriguez Artalejo (Eds.): Granada Workshop on the Integration of Functional and Logic Programming
1991-13 * Rolf Hager, Wolfgang Kremer: The Adaptive Priority Scheduler: A More Fair Priority Service Discipline
1991-14 * Andreas Fasbender, Wolfgang Kremer: A New Approximation Algorithm for Tandem Networks with Priority Nodes
1991-15 J. Börstler, A. Zündorf: Revisiting extensions to Modula-2 to support reusability
1991-16 J. Börstler, Th. Janning: Bridging the gap between Requirements Analysis and Design
1991-17 A. Zündorf, A. Schürr: Nondeterministic Control Structures for Graph Rewriting Systems
1991-18 * Matthias Jarke, John Mylopoulos, Joachim W. Schmidt, Yannis Vassiliou: DAIDA: An Environment for Evolving Information Systems
1991-19 M. Jeusfeld, M. Jarke: From Relational to Object-Oriented Integrity Simplification
1991-20 G. Hogen, A. Kindler, R. Loogen: Automatic Parallelization of Lazy Functional Programs
1991-21 * Prof. Dr. rer. nat. Otto Spaniol: ODP (Open Distributed Processing): Yet another Viewpoint
1991-22 H. Kuchen, F. Lücking, H. Stoltze: The Topology Description Language TDL
1991-23 S. Graf, B. Steffen: Compositional Minimization of Finite State Systems
1991-24 R. Cleaveland, J. Parrow, B. Steffen: The Concurrency Workbench: A Semantics Based Tool for the Verification of Concurrent Systems
1991-25 * Rudolf Mathar, Jürgen Mattfeldt: Optimal Transmission Ranges for Mobile Communication in Linear Multihop Packet Radio Networks
1991-26 M. Jeusfeld, M. Staudt: Query Optimization in Deductive Object Bases
1991-27 J. Knoop, B. Steffen: The Interprocedural Coincidence Theorem
1991-28 J. Knoop, B. Steffen: Unifying Strength Reduction and Semantic Code Motion
1991-30 T. Margaria: First-Order theories for the verification of complex FSMs
1991-31 B. Steffen: Generating Data Flow Analysis Algorithms from Modal Specifications
1992-01 Stefan Eherer (ed.), Fachgruppe Informatik: Jahresbericht 1991
1992-02 * Bernhard Westfechtel: Basismechanismen zur Datenverwaltung in strukturbezogenen Hypertextsystemen
1992-04 S. A. Smolka, B. Steffen: Priority as Extremal Probability
1992-05 * Matthias Jarke, Carlos Maltzahn, Thomas Rose: Sharing Processes: Team Coordination in Design Repositories

1992-06 O. Burkart, B. Steffen: Model Checking for Context-Free Processes
1992-07 * Matthias Jarke, Klaus Pohl: Information Systems Quality and Quality Information Systems
1992-08 * Rudolf Mathar, Jürgen Mattfeldt: Analyzing Routing Strategy NFP in Multihop Packet Radio Networks on a Line
1992-09 * Alfons Kemper, Guido Moerkotte: Grundlagen objektorientierter Datenbanksysteme
1992-10 Matthias Jarke, Manfred Jeusfeld, Andreas Miethsam, Michael Gocek: Towards a logic-based reconstruction of software configuration management
1992-11 Werner Hans: A Complete Indexing Scheme for WAM-based Abstract Machines
1992-12 W. Hans, R. Loogen, St. Winkler: On the Interaction of Lazy Evaluation and Backtracking
1992-13 * Matthias Jarke, Thomas Rose: Specification Management with CAD
1992-14 Th. Noll, H. Vogler: Top-down Parsing with Simultaneous Evaluation on Noncircular Attribute Grammars
1992-15 A. Schuerr, B. Westfechtel: Graphgrammatiken und Graphersetzungssysteme(written in german)
1992-16 * Graduiertenkolleg Informatik und Technik (Hrsg.): Forschungsprojekte des Graduiertenkollegs Informatik und Technik
1992-17 M. Jarke (ed.): ConceptBase V3.1 User Manual
1992-18 * Clarence A. Ellis, Matthias Jarke (Eds.): Distributed Cooperation in Integrated Information Systems - Proceedings of the Third International Workshop on Intelligent and Cooperative Information Systems
1992-19-00 H. Kuchen, R. Loogen (eds.): Proceedings of the 4th Int. Workshop on the Parallel Implementation of Functional Languages
1992-19-01 G. Hogen, R. Loogen: PASTEL - A Parallel Stack-Based Implementation of Eager Functional Programs with Lazy Data Structures (Extended Abstract)
1992-19-02 H. Kuchen, K. Gladitz: Implementing Bags on a Shared Memory MIMD-Machine
1992-19-03 C. Rathsack, S.B. Scholz: LISA - A Lazy Interpreter for a Full-Fledged Lambda-Calculus
1992-19-04 T.A. Bratvold: Determining Useful Parallelism in Higher Order Functions
1992-19-05 S. Kahrs: Polymorphic Type Checking by Interpretation of Code
1992-19-06 M. Chakravarty, M. Köhler: Equational Constraints, Residuation, and the Parallel JUMP-Machine
1992-19-07 J. Seward: Polymorphic Strictness Analysis using Frontiers (Draft Version)
1992-19-08 D. Gärtner, A. Kimms, W. Kluge: pi-Red^+ - A Compiling Graph-Reduction System for a Full Fledged Lambda-Calculus
1992-19-09 D. Howe, G. Burn: Experiments with strict STG code
1992-19-10 J. Glauert: Parallel Implementation of Functional Languages Using Small Processes
1992-19-11 M. Joy, T. Axford: A Parallel Graph Reduction Machine
1992-19-12 A. Bennett, P. Kelly: Simulation of Multicache Parallel Reduction

1992-19-13 K. Langendoen, D.J. Agterkamp: Cache Behaviour of Lazy Functional Programs (Working Paper)

1992-19-14 K. Hammond, S. Peyton Jones: Profiling scheduling strategies on the GRIP parallel reducer

1992-19-15 S. Mintchev: Using Strictness Information in the STG-machine

1992-19-16 D. Rushall: An Attribute Grammar Evaluator in Haskell

1992-19-17 J. Wild, H. Glaser, P. Hartel: Statistics on storage management in a lazy functional language implementation

1992-19-18 W.S. Martins: Parallel Implementations of Functional Languages

1992-19-19 D. Lester: Distributed Garbage Collection of Cyclic Structures (Draft version)

1992-19-20 J.C. Glas, R.F.H. Hofman, W.G. Vree: Parallelization of Branch-and-Bound Algorithms in a Functional Programming Environment

1992-19-21 S. Hwang, D. Rushall: The nu-STG machine: a parallelized Spineless Tagless Graph Reduction Machine in a distributed memory architecture (Draft version)

1992-19-22 G. Burn, D. Le Metayer: Cps-Translation and the Correctness of Optimising Compilers

1992-19-23 S.L. Peyton Jones, P. Wadler: Imperative functional programming (Brief summary)

1992-19-24 W. Damm, F. Liu, Th. Peikenkamp: Evaluation and Parallelization of Functions in Functional + Logic Languages (abstract)

1992-19-25 M. Kesseler: Communication Issues Regarding Parallel Functional Graph Rewriting

1992-19-26 Th. Peikenkamp: Charakterizing and representing neededness in functional loginc languages (abstract)

1992-19-27 H. Doerr: Monitoring with Graph-Grammars as formal operational Models

1992-19-28 J. van Groningen: Some implementation aspects of Concurrent Clean on distributed memory architectures

1992-19-29 G. Ostheimer: Load Bounding for Implicit Parallelism (abstract)

1992-20 H. Kuchen, F.J. Lopez Fraguas, J.J. Moreno Navarro, M. Rodriguez Artalejo: Implementing Disequality in a Lazy Functional Logic Language

1992-21 H. Kuchen, F.J. Lopez Fraguas: Result Directed Computing in a Functional Logic Language

1992-22 H. Kuchen, J.J. Moreno Navarro, M.V. Hermenegildo: Independent AND-Parallel Narrowing

1992-23 T. Margaria, B. Steffen: Distinguishing Formulas for Free

1992-24 K. Pohl: The Three Dimensions of Requirements Engineering

1992-25 * R. Stainov: A Dynamic Configuration Facility for Multimedia Communications

1992-26 * Michael von der Beeck: Integration of Structured Analysis and Timed Statecharts for Real-Time and Concurrency Specification

1992-27 W. Hans, St. Winkler: Aliasing and Groundness Analysis of Logic Programs through Abstract Interpretation and its Safety

1992-28 * Gerhard Steinke, Matthias Jarke: Support for Security Modeling in Information Systems Design

1992-29 B. Schinzel: Warum Frauenforschung in Naturwissenschaft und Technik

1992-30 A. Kemper, G. Moerkotte, K. Peithner: Object-Orientation Axiomatised by Dynamic Logic

1992-32 * Bernd Heinrichs, Kai Jakobs: Timer Handling in High-Performance Transport Systems

1992-33 * B. Heinrichs, K. Jakobs, K. Lenßen, W. Reinhardt, A. Spinner: Euro-Bridge: Communication Services for Multimedia Applications

1992-34 C. Gerlhof, A. Kemper, Ch. Kilger, G. Moerkotte: Partition-Based Clustering in Object Bases: From Theory to Practice

1992-35 J. Börstler: Feature-Oriented Classification and Reuse in IPSEN

1992-36 M. Jarke, J. Bubenko, C. Rolland, A. Sutcliffe, Y. Vassiliou: Theories Underlying Requirements Engineering: An Overview of NATURE at Genesis

1992-37 * K. Pohl, M. Jarke: Quality Information Systems: Repository Support for Evolving Process Models

1992-38 A. Zuendorf: Implementation of the imperative / rule based language PROGRES

1992-39 P. Koch: Intelligentes Backtracking bei der Auswertung funktional-logischer Programme

1992-40 * Rudolf Mathar, Jürgen Mattfeldt: Channel Assignment in Cellular Radio Networks

1992-41 * Gerhard Friedrich, Wolfgang Neidl: Constructive Utility in Model-Based Diagnosis Repair Systems

1992-42 * P. S. Chen, R. Hennicker, M. Jarke: On the Retrieval of Reusable Software Components

1992-43 W. Hans, St.Winkler: Abstract Interpretation of Functional Logic Languages

1992-44 N. Kiesel, A. Schuerr, B. Westfechtel: Design and Evaluation of GRAS, a Graph-Oriented Database System for Engineering Applications

1993-01 * Fachgruppe Informatik: Jahresbericht 1992

1993-02 * Patrick Shicheng Chen: On Inference Rules of Logic-Based Information Retrieval Systems

1993-03 G. Hogen, R. Loogen: A New Stack Technique for the Management of Runtime Structures in Distributed Environments

1993-05 A. Zündorf: A Heuristic for the Subgraph Isomorphism Problem in Executing PROGRES

1993-06 A. Kemper, D. Kossmann: Adaptable Pointer Swizzling Strategies in Object Bases: Design, Realization, and Quantitative Analysis

1993-07 * Graduiertenkolleg Informatik und Technik (Hrsg.): Graduiertenkolleg Informatik und Technik

1993-08 * Matthias Berger: k-Coloring Vertices using a Neural Network with Convergence to Valid Solutions

1993-09 M. Buchheit, M. Jeusfeld, W. Nutt, M. Staudt: Subsumption between Queries to Object-Oriented Databases

1993-10 O. Burkart, B. Steffen: Pushdown Processes: Parallel Composition and Model Checking

1993-11 * R. Große-Wienker, O. Hermanns, D. Menzenbach, A. Pollacks, S. Repetzki, J. Schwartz, K. Sonnenschein, B. Westfechtel: Das SUKITS-Projekt: A-posteriori-Integration heterogener CIM-Anwendungssysteme

1993-12 * Rudolf Mathar, Jürgen Mattfeldt: On the Distribution of Cumulated Interference Power in Rayleigh Fading Channels
1993-13 O. Maler, L. Staiger: On Syntactic Congruences for omega-languages
1993-14 M. Jarke, St. Eherer, R. Gallersdoerfer, M. Jeusfeld, M. Staudt: ConceptBase - A Deductive Object Base Manager
1993-15 M. Staudt, H.W. Nissen, M.A. Jeusfeld: Query by Class, Rule and Concept
1993-16 * M. Jarke, K. Pohl, St. Jacobs et al.: Requirements Engineering: An Integrated View of Representation Process and Domain
1993-17 * M. Jarke, K. Pohl: Establishing Vision in Context: Towards a Model of Requirements Processes
1993-18 W. Hans, H. Kuchen, St. Winkler: Full Indexing for Lazy Narrowing
1993-19 W. Hans, J.J. Ruz, F. Saenz, St. Winkler: A VHDL Specification of a Shared Memory Parallel Machine for Babel
1993-20 * K. Finke, M. Jarke, P. Szczurko, R. Soltysiak: Quality Management for Expert Systems in Process Control
1993-21 M. Jarke, M.A. Jeusfeld, P. Szczurko: Three Aspects of Intelligent Cooperation in the Quality Cycle
1994-01 Margit Generet, Sven Martin (eds.), Fachgruppe Informatik: Jahresbericht 1993
1994-02 M. Lefering: Development of Incremental Integration Tools Using Formal Specifications
1994-03 * P. Constantopoulos, M. Jarke, J. Mylopoulos, Y. Vassiliou: The Software Information Base: A Server for Reuse
1994-04 * Rolf Hager, Rudolf Mathar, Jürgen Mattfeldt: Intelligent Cruise Control and Reliable Communication of Mobile Stations
1994-05 * Rolf Hager, Peter Hermesmann, Michael Portz: Feasibility of Authentication Procedures within Advanced Transport Telematics
1994-06 * Claudia Popien, Bernd Meyer, Axel Kuepper: A Formal Approach to Service Import in ODP Trader Federations
1994-07 P. Peters, P. Szczurko: Integrating Models of Quality Management Methods by an Object-Oriented Repository
1994-08 * Manfred Nagl, Bernhard Westfechtel: A Universal Component for the Administration in Distributed and Integrated Development Environments
1994-09 * Patrick Horster, Holger Petersen: Signatur- und Authentifikationsverfahren auf der Basis des diskreten Logarithmusproblems
1994-11 A. Schürr: PROGRES, A Visual Language and Environment for PROgramming with Graph REwrite Systems
1994-12 A. Schürr: Specification of Graph Translators with Triple Graph Grammars
1994-13 A. Schürr: Logic Based Programmed Structure Rewriting Systems
1994-14 L. Staiger: Codes, Simplifying Words, and Open Set Condition
1994-15 * Bernhard Westfechtel: A Graph-Based System for Managing Configurations of Engineering Design Documents
1994-16 P. Klein: Designing Software with Modula-3
1994-17 I. Litovsky, L. Staiger: Finite acceptance of infinite words

1994-18 G. Hogen, R. Loogen: Parallel Functional Implementations: Graphbased vs. Stackbased Reduction

1994-19 M. Jeusfeld, U. Johnen: An Executable Meta Model for Re-Engineering of Database Schemas

1994-20 * R. Gallersdörfer, M. Jarke, K. Klabunde: Intelligent Networks as a Data Intensive Application (INDIA)

1994-21 M. Mohnen: Proving the Correctness of the Static Link Technique Using Evolving Algebras

1994-22 H. Fernau, L. Staiger: Valuations and Unambiguity of Languages, with Applications to Fractal Geometry

1994-24 * M. Jarke, K. Pohl, R. Dömges, St. Jacobs, H. W. Nissen: Requirements Information Management: The NATURE Approach

1994-25 * M. Jarke, K. Pohl, C. Rolland, J.-R. Schmitt: Experience-Based Method Evaluation and Improvement: A Process Modeling Approach

1994-26 * St. Jacobs, St. Kethers: Improving Communication and Decision Making within Quality Function Deployment

1994-27 * M. Jarke, H. W. Nissen, K. Pohl: Tool Integration in Evolving Information Systems Environments

1994-28 O. Burkart, D. Caucal, B. Steffen: An Elementary Bisimulation Decision Procedure for Arbitrary Context-Free Processes

1995-01 * Fachgruppe Informatik: Jahresbericht 1994

1995-02 Andy Schürr, Andreas J. Winter, Albert Zündorf: Graph Grammar Engineering with PROGRES

1995-03 Ludwig Staiger: A Tight Upper Bound on Kolmogorov Complexity by Hausdorff Dimension and Uniformly Optimal Prediction

1995-04 Birgitta König-Ries, Sven Helmer, Guido Moerkotte: An experimental study on the complexity of left-deep join ordering problems for cyclic queries

1995-05 Sophie Cluet, Guido Moerkotte: Efficient Evaluation of Aggregates on Bulk Types

1995-06 Sophie Cluet, Guido Moerkotte: Nested Queries in Object Bases

1995-07 Sophie Cluet, Guido Moerkotte: Query Optimization Techniques Exploiting Class Hierarchies

1995-08 Markus Mohnen: Efficient Compile-Time Garbage Collection for Arbitrary Data Structures

1995-09 Markus Mohnen: Functional Specification of Imperative Programs: An Alternative Point of View of Functional Languages

1995-10 Rainer Gallersdörfer, Matthias Nicola: Improving Performance in Replicated Databases through Relaxed Coherency

1995-11 * M.Staudt, K.von Thadden: Subsumption Checking in Knowledge Bases

1995-12 * G.V.Zemanek, H.W.Nissen, H.Hubert, M.Jarke: Requirements Analysis from Multiple Perspectives: Experiences with Conceptual Modeling Technology

1995-13 * M.Staudt, M.Jarke: Incremental Maintenance of Externally Materialized Views

1995-14 * P.Peters, P.Szczurko, M.Jeusfeld: Oriented Information Management: Conceptual Models at Work

1995-15 * Matthias Jarke, Sudha Ram (Hrsg.): WITS 95 Proceedings of the 5th Annual Workshop on Information Technologies and Systems

1995-16 * W.Hans, St.Winkler, F.Saenz: Distributed Execution in Functional Logic Programming

1996-01 * Jahresbericht 1995

1996-02 Michael Hanus, Christian Prehofer: Higher-Order Narrowing with Definitional Trees

1996-03 * W.Scheufele, G.Moerkotte: Optimal Ordering of Selections and Joins in Acyclic Queries with Expensive Predicates

1996-04 Klaus Pohl: PRO-ART: Enabling Requirements Pre-Traceability

1996-05 Klaus Pohl: Requirements Engineering: An Overview

1996-06 * M.Jarke, W.Marquardt: Design and Evaluation of Computer–Aided Process Modelling Tools

1996-07 Olaf Chitil: The Sigma-Semantics: A Comprehensive Semantics for Functional Programs

1996-08 * S.Sripada: On Entropy and the Limitations of the Second Law of Thermodynamics

1996-09 Michael Hanus (Ed.): Proceedings of the Poster Session of ALP96 - Fifth International Conference on Algebraic and Logic Programming

1996-09-0 Michael Hanus (Ed.): Proceedings of the Poster Session of ALP 96 - Fifth International Conference on Algebraic and Logic Programming: Introduction and table of contents

1996-09-1 Ilies Alouini: An Implementation of Conditional Concurrent Rewriting on Distributed Memory Machines

1996-09-2 Olivier Danvy, Karoline Malmkjær: On the Idempotence of the CPS Transformation

1996-09-3 Víctor M. Gulías, José L. Freire: Concurrent Programming in Haskell

1996-09-4 Sébastien Limet, Pierre Réty: On Decidability of Unifiability Modulo Rewrite Systems

1996-09-5 Alexandre Tessier: Declarative Debugging in Constraint Logic Programming

1996-10 Reidar Conradi, Bernhard Westfechtel: Version Models for Software Configuration Management

1996-11 * C.Weise, D.Lenzkes: A Fast Decision Algorithm for Timed Refinement

1996-12 * R.Dömges, K.Pohl, M.Jarke, B.Lohmann, W.Marquardt: PRO-ART/CE* — An Environment for Managing the Evolution of Chemical Process Simulation Models

1996-13 * K.Pohl, R.Klamma, K.Weidenhaupt, R.Dömges, P.Haumer, M.Jarke: A Framework for Process-Integrated Tools

1996-14 * R.Gallersdörfer, K.Klabunde, A.Stolz, M.Eßmajor: INDIA — Intelligent Networks as a Data Intensive Application, Final Project Report, June 1996

1996-15 * H.Schimpe, M.Staudt: VAREX: An Environment for Validating and Refining Rule Bases

1996-16 * M.Jarke, M.Gebhardt, S.Jacobs, H.Nissen: Conflict Analysis Across Heterogeneous Viewpoints: Formalization and Visualization

1996-17 Manfred A. Jeusfeld, Tung X. Bui: Decision Support Components on the Internet

1996-18 Manfred A. Jeusfeld, Mike Papazoglou: Information Brokering: Design, Search and Transformation

1996-19 * P.Peters, M.Jarke: Simulating the impact of information flows in networked organizations

1996-20 Matthias Jarke, Peter Peters, Manfred A. Jeusfeld: Model-driven planning and design of cooperative information systems

1996-21 * G.de Michelis, E.Dubois, M.Jarke, F.Matthes, J.Mylopoulos, K.Pohl, J.Schmidt, C.Woo, E.Yu: Cooperative information systems: a manifesto

1996-22 * S.Jacobs, M.Gebhardt, S.Kethers, W.Rzasa: Filling HTML forms simultaneously: CoWeb architecture and functionality

1996-23 * M.Gebhardt, S.Jacobs: Conflict Management in Design

1997-01 Michael Hanus, Frank Zartmann (eds.): Jahresbericht 1996

1997-02 Johannes Faassen: Using full parallel Boltzmann Machines for Optimization

1997-03 Andreas Winter, Andy Schürr: Modules and Updatable Graph Views for PROgrammed Graph REwriting Systems

1997-04 Markus Mohnen, Stefan Tobies: Implementing Context Patterns in the Glasgow Haskell Compiler

1997-05 * S.Gruner: Schemakorrespondenzaxiome unterstützen die paargrammatische Spezifikation inkrementeller Integrationswerkzeuge

1997-06 Matthias Nicola, Matthias Jarke: Design and Evaluation of Wireless Health Care Information Systems in Developing Countries

1997-07 Petra Hofstedt: Taskparallele Skelette für irregulär strukturierte Probleme in deklarativen Sprachen

1997-08 Dorothea Blostein, Andy Schürr: Computing with Graphs and Graph Rewriting

1997-09 Carl-Arndt Krapp, Bernhard Westfechtel: Feedback Handling in Dynamic Task Nets

1997-10 Matthias Nicola, Matthias Jarke: Integrating Replication and Communication in Performance Models of Distributed Databases

1997-11 * R. Klamma, P. Peters, M. Jarke: Workflow Support for Failure Management in Federated Organizations

1997-13 Markus Mohnen: Optimising the Memory Management of Higher-Order Functional Programs

1997-14 Roland Baumann: Client/Server Distribution in a Structure-Oriented Database Management System

1997-15 George Botorog: High-Level Parallel Programming and the Efficient Implementation of Numerical Algorithms

1998-01 * Fachgruppe Informatik: Jahresbericht 1997

1998-02 Stefan Gruner, Manfred Nagel, Andy Schürr: Fine-grained and Structure-Oriented Document Integration Tools are Needed for Development Processes

1998-03 Stefan Gruner: Einige Anmerkungen zur graphgrammatischen Spezifikation von Integrationswerkzeugen nach Westfechtel, Janning, Lefering und Schürr

1998-04 * O. Kubitz: Mobile Robots in Dynamic Environments

1998-05 Martin Leucker, Stephan Tobies: Truth - A Verification Platform for Distributed Systems

1998-06 * Matthias Oliver Berger: DECT in the Factory of the Future
1998-07 M. Arnold, M. Erdmann, M. Glinz, P. Haumer, R. Knoll, B. Paech, K. Pohl, J. Ryser, R. Studer, K. Weidenhaupt: Survey on the Scenario Use in Twelve Selected Industrial Projects
1998-08 * H. Aust: Sprachverstehen und Dialogmodellierung in natürlichsprachlichen Informationssystemen
1998-09 * Th. Lehmann: Geometrische Ausrichtung medizinischer Bilder am Beispiel intraoraler Radiographien
1998-10 * M. Nicola, M. Jarke: Performance Modeling of Distributed and Replicated Databases
1998-11 * Ansgar Schleicher, Bernhard Westfechtel, Dirk Jäger: Modeling Dynamic Software Processes in UML
1998-12 * W. Appelt, M. Jarke: Interoperable Tools for Cooperation Support using the World Wide Web
1998-13 Klaus Indermark: Semantik rekursiver Funktionsdefinitionen mit Striktheitsinformation
1999-01 * Jahresbericht 1998
1999-02 * F. Huch: Verifcation of Erlang Programs using Abstract Interpretation and Model Checking — Extended Version
1999-03 * R. Gallersdörfer, M. Jarke, M. Nicola: The ADR Replication Manager
1999-04 María Alpuente, Michael Hanus, Salvador Lucas, Germán Vidal: Specialization of Functional Logic Programs Based on Needed Narrowing
1999-05 * W. Thomas (Ed.): DLT 99 - Developments in Language Theory Fourth International Conference
1999-06 * Kai Jakobs, Klaus-Dieter Kleefeld: Informationssysteme für die angewandte historische Geographie
1999-07 Thomas Wilke: CTL+ is exponentially more succinct than CTL
1999-08 Oliver Matz: Dot-Depth and Monadic Quantifier Alternation over Pictures
2000-01 * Jahresbericht 1999
2000-02 Jens Vöge, Marcin Jurdzinski: A Discrete Strategy Improvement Algorithm for Solving Parity Games
2000-03 D. Jäger, A. Schleicher, B. Westfechtel: UPGRADE: A Framework for Building Graph-Based Software Engineering Tools
2000-04 Andreas Becks, Stefan Sklorz, Matthias Jarke: Exploring the Semantic Structure of Technical Document Collections: A Cooperative Systems Approach
2000-05 Mareike Schoop: Cooperative Document Management
2000-06 Mareike Schoop, Christoph Quix (eds.): Proceedings of the Fifth International Workshop on the Language-Action Perspective on Communication Modelling
2000-07 * Markus Mohnen, Pieter Koopman (Eds.): Proceedings of the 12th International Workshop of Functional Languages
2000-08 Thomas Arts, Thomas Noll: Verifying Generic Erlang Client-Server Implementations
2001-01 * Jahresbericht 2000
2001-02 Benedikt Bollig, Martin Leucker: Deciding LTL over Mazurkiewicz Traces

2001-03 Thierry Cachat: The power of one-letter rational languages
2001-04 Benedikt Bollig, Martin Leucker, Michael Weber: Local Parallel Model Checking for the Alternation Free mu-Calculus
2001-05 Benedikt Bollig, Martin Leucker, Thomas Noll: Regular MSC Languages
2001-06 Achim Blumensath: Prefix-Recognisable Graphs and Monadic Second-Order Logic
2001-07 Martin Grohe, Stefan Wöhrle: An Existential Locality Theorem
2001-08 Mareike Schoop, James Taylor (eds.): Proceedings of the Sixth International Workshop on the Language-Action Perspective on Communication Modelling
2001-09 Thomas Arts, Jürgen Giesl: A collection of examples for termination of term rewriting using dependency pairs
2001-10 Achim Blumensath: Axiomatising Tree-interpretable Structures
2001-11 Klaus Indermark, Thomas Noll (eds.): Kolloquium Programmiersprachen und Grundlagen der Programmierung
2002-01 * Jahresbericht 2001
2002-02 Jürgen Giesl, Aart Middeldorp: Transformation Techniques for Context-Sensitive Rewrite Systems
2002-03 Benedikt Bollig, Martin Leucker, Thomas Noll: Generalised Regular MSC Languages
2002-04 Jürgen Giesl, Aart Middeldorp: Innermost Termination of Context-Sensitive Rewriting
2002-05 Horst Lichter, Thomas von der Maßen, Thomas Weiler: Modelling Requirements and Architectures for Software Product Lines
2002-06 Henry N. Adorna: 3-Party Message Complexity is Better than 2-Party Ones for Proving Lower Bounds on the Size of Minimal Nondeterministic Finite Automata
2002-07 Jörg Dahmen: Invariant Image Object Recognition using Gaussian Mixture Densities
2002-08 Markus Mohnen: An Open Framework for Data-Flow Analysis in Java
2002-09 Markus Mohnen: Interfaces with Default Implementations in Java
2002-10 Martin Leucker: Logics for Mazurkiewicz traces
2002-11 Jürgen Giesl, Hans Zantema: Liveness in Rewriting
2003-01 * Jahresbericht 2002
2003-02 Jürgen Giesl, René Thiemann: Size-Change Termination for Term Rewriting
2003-03 Jürgen Giesl, Deepak Kapur: Deciding Inductive Validity of Equations
2003-04 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke: Improving Dependency Pairs
2003-05 Christof Löding, Philipp Rohde: Solving the Sabotage Game is PSPACE-hard
2003-06 Franz Josef Och: Statistical Machine Translation: From Single-Word Models to Alignment Templates
2003-07 Horst Lichter, Thomas von der Maßen, Alexander Nyßen, Thomas Weiler: Vergleich von Ansätzen zur Feature Modellierung bei der Softwareproduktlinienentwicklung
2003-08 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke: Mechanizing Dependency Pairs

2004-01 * Fachgruppe Informatik: Jahresbericht 2003
2004-02 Benedikt Bollig, Martin Leucker: Message-Passing Automata are expressively equivalent to EMSO logic
2004-03 Delia Kesner, Femke van Raamsdonk, Joe Wells (eds.): HOR 2004 – 2nd International Workshop on Higher-Order Rewriting
2004-04 Slim Abdennadher, Christophe Ringeissen (eds.): RULE 04 – Fifth International Workshop on Rule-Based Programming
2004-05 Herbert Kuchen (ed.): WFLP 04 – 13th International Workshop on Functional and (Constraint) Logic Programming
2004-06 Sergio Antoy, Yoshihito Toyama (eds.): WRS 04 – 4th International Workshop on Reduction Strategies in Rewriting and Programming
2004-07 Michael Codish, Aart Middeldorp (eds.): WST 04 – 7th International Workshop on Termination
2004-08 Klaus Indermark, Thomas Noll: Algebraic Correctness Proofs for Compiling Recursive Function Definitions with Strictness Information
2004-09 Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Parameterized Power Domination Complexity
2004-10 Zinaida Benenson, Felix C. Gärtner, Dogan Kesdogan: Secure Multi-Party Computation with Security Modules
2005-01 * Fachgruppe Informatik: Jahresbericht 2004
2005-02 Maximillian Dornseif, Felix C. Gärtner, Thorsten Holz, Martin Mink: An Offensive Approach to Teaching Information Security: "Aachen Summer School Applied IT Security"
2005-03 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp: Proving and Disproving Termination of Higher-Order Functions
2005-04 Daniel Mölle, Stefan Richter, Peter Rossmanith: A Faster Algorithm for the Steiner Tree Problem
2005-05 Fabien Pouget, Thorsten Holz: A Pointillist Approach for Comparing Honeypots
2005-06 Simon Fischer, Berthold Vöcking: Adaptive Routing with Stale Information
2005-07 Felix C. Freiling, Thorsten Holz, Georg Wicherski: Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks
2005-08 Joachim Kneis, Peter Rossmanith: A New Satisfiability Algorithm With Applications To Max-Cut
2005-09 Klaus Kursawe, Felix C. Freiling: Byzantine Fault Tolerance on General Hybrid Adversary Structures
2005-10 Benedikt Bollig: Automata and Logics for Message Sequence Charts
2005-11 Simon Fischer, Berthold Vöcking: A Counterexample to the Fully Mixed Nash Equilibrium Conjecture
2005-12 Neeraj Mittal, Felix Freiling, Subbarayan Venkatesan, Lucia Draque Penso: Efficient Reductions for Wait-Free Termination Detection in Crash-Prone Systems
2005-13 Carole Delporte-Gallet, Hugues Fauconnier, Felix C. Freiling: Revisiting Failure Detection and Consensus in Omission Failure Environments
2005-14 Felix C. Freiling, Sukumar Ghosh: Code Stabilization
2005-15 Uwe Naumann: The Complexity of Derivative Computation

* These reports are only available as a printed version.

Please contact biblio@informatik.rwth-aachen.de to obtain copies.