ELSEVIER Theoretical Computer Science 187 (1997) 231–248

Theoretical Computer Science

Automatic generation of finite-element code by simultaneous optimization of expressions

Jože Korelc*

Faculty of Civil Engineering and Geodesy, University of Ljubljana, Jamova 2, SI-1000 Ljubljana, Slovenia
Institut of Mechanics, TH Darmstdat, Germany

Abstract

The paper presents a MATHEMATICA package SMS (Symbolic Mechanics System) for the automatic derivation of formulas needed in nonlinear finite element analysis. Symbolic generation of the characteristic arrays of nonlinear finite elements (e.g. nodal force vectors, stiffness matrices, sensitivity vectors) leads to exponential behavior, both in time and space. A new approach, implemented in SMS, avoids this problem by combining several techniques: symbolic capabilities of Mathematica, automatic differentiation technique, simultaneous optimization of expressions and a stochastic evaluation of the formulas instead of a conventional pattern matching technique. SMS translates the derived symbolic formulas into an efficient compiled language (FORTRAN or C). The generated code is then incorporated into an existing finite element analysis environment. SMS was already used to developed several new, geometrically and materially nonlinear finite elements with up to 72 degrees of freedom. The design and implementation of SMS are presented. Efficiency of the new approach is compared with the efficiency of the manually written code on an example.

Keywords: Finite element method; Automatic code generation; Code optimization

1. Introduction

Nowadays more and more complex mechanical models are applied for the simulation of engineering structures. The mathematical models for these problems are described by a system of partial differential equations. Most of the existing numerical methods for solving partial differential equations can be classified into two classes: Finite Difference Method (FDM) and Finite Element Method (FEM). Unfortunately, the applicability of the present finite elements is often limited and the search for finite elements which can provide a general tool for arbitrary problems in mechanics of solids has a long

* Correspondence address: Faculty of Civil Engineering and Geodesy, University of Ljubljana, Jamova 2, SI-1000 Ljubljana, Slovenia. E-mail: jkorelc@fagg.uni-lj.si.

history. In order to develop a new finite element model quite a lot of time is spent in deriving characteristic element quantities such as gradients, Jacobian, Hessian and coding of the program in a efficient compiled language. These quantities are required within the numerical solution procedure.

Using the general finite element environment for analyzing a variety of problems and for incorporating new elements, such as FEAP [21] or ABAQUS [20], is now already an everyday practice. The general finite element environments can handle, regardless of the type of elements, most of the required operations, such as:

(i) preprocessing of the input data;
(ii) manipulating and organizing of the data related to nodes and elements, material characteristics, displacements and stresses, etc.;
(iii) construction of the global matrices by invoking different elements subroutines;
(iv) solving the system of equations;
(v) post-processing and analysis of results.

The idea is to use symbolic algebraic systems for the derivation of characteristic element quantities and automatic code generator for the generation of numerical code. Symbolically generated code is then incorporated into the existing finite element environment and used within the numerical procedures.

The symbolic-numeric approach to FEM and FDM was extensively studied in the last few years. Based on the studies various systems for automatic code generation were developed. In many ways, the present stage of generation of finite difference code is more elaborated and more general than the generation of the FEM code. Various transformations, differentiations and matrix operations, and a large number of degrees of freedom, involved in the derivation of characteristic FEM quantities often lead to exponential growth in space and time. Therefore, additional structural knowledge about the problem is needed, which is not the case for FDM. Within automatic generation of Finite Difference code, the SINAPSE system by Kant [8] and Kant et al. [9] should be mentioned.

The article deals only with the automatic generation of FEM code. Several specialized systems have been developed, such as FINGER by Wang [24], PIER by Sharma [18], SFEAS by Leef and Yun [13], SGEM by Nagabhushanam et al. [16], etc. These specialized systems can be used to derive the vectors and matrices required in FEM. In these systems the general symbolic and algebraic computation (SAC) systems, such as MATHEMATICA [22], are typically combined with an automatic code generator. The most elaborate work has been done by Wang and coworkers. The symbolic system FINGER employs various expression optimization techniques (use of the symmetric properties of the formulas, automatic introduction of intermediate variables), leading to a highly efficient and short FEM code. The later extension (PFINGER) generates also parallel FEM code [23].

The major limitation of the symbolic systems, when applied to FEM code generation, as pointed out before and by many authors [24, 2], etc., is an uncontrollable growth of expressions. Investigations and experiments show that an efficient control of expression growth can be obtained by a cooperation of SAC and the following techniques: Automatic differentiation (AD) tools [5], Problem Solving Environments [3], and Theorem proving systems. However, the general problem of cooperative problem solving has at the present stage not been solved yet.

The idea presented in this paper is not to try to combine different systems, but to combine different techniques inside one system in order to avoid the above-mentioned problems. Thus, the main objective will be to combine techniques in such a way that leads to an optimal environment for the design and implementation of arbitrary finite elements. Among the presented systems the most versatile are indeed the SAC systems. They normally contain, beside the algebraic manipulation, graphics and numeric capabilities, also powerful programing languages. It is therefore quite easy to simulate other techniques inside the SAC system. A new approach is called Simultaneous Stochastic Simplification of numerical code. This approach combines a general computer algebra system MATHEMATICA with an automatic differentiation technique and an automatic theorem proving by examples. To alleviate the problem of the growth of expressions and redundant calculations, simultaneous simplification of symbolic expressions is used. Stochastic evaluation of the formulas is used for determining the equivalence of algebraic expressions, instead of the conventional pattern matching technique (see also [14]). According to the authors knowledge the combination of the SAC system, automatic differentiation technique and simultaneous stochastic simplification for the automatic FEM code generation has not been used before.

SMS was designed to approach the especially hard problems, where the general strategy to efficient FEM formulation has not yet been established. SMS was already used to developed several new, geometrically and materially nonlinear finite elements with up to 72 degrees of freedom.

2. SMS – symbolic mechanics system

The SMS system is written in the symbolic language of MATHEMATICA. It consists of about 200 functions and 10 000 lines of MATHEMATICA's source code. Typical SMS function takes the expression provided by the user, either interactively or in file, and returns an optimized version of the expression. Optimized version of the expression can result in a newly created intermediate symbol (v_i), or in an original expression in parts replaced by previously created intermediate symbols. In the first case SMS stores the new expression in an internal data base. The data base contains a global vector of all expressions, information about dependencies of the symbols, labels and names of the symbols, partial derivatives, etc. The data base is a global object which maintains informations during the MATHEMATICA session.

In this section the techniques implemented in SMS to alleviate the problem of exponential growth of expressions are discussed.

2.1. Simultaneous optimization and expression labelling

The classical way for optimizing expressions in computer algebra systems is searching for common sub-expressions at the end of the derivation, before the generation of the numerical code (see e.g. [24]). In the numerical code common sub-expressions appear as new intermediate variables. A new approach named "Simultaneous optimization of expressions" is proposed. It denotes that formulas are optimized, simplified and replaced by new intermediate variables simultaneously with the derivation of the problem. The optimized version is then used in further operations. If the optimization is performed simultaneously, the explicit form of the expression is obviously lost, since some parts are replaced by intermediate variables. The problems which arise will be assessed at the end of the chapter.

Let us consider a simple example to illustrate the procedure. Quantity u is interpolated by a linear combination (1) of the unknown parameters u_1, u_2, u_3 and shape functions (2).

$$u = \sum_{i=1}^{3} N_i u_i, \tag{1}$$

$$N_1 = \frac{x}{L}, \qquad N_2 = 1 - \frac{x}{L}, \qquad N_3 = \frac{x}{L}\left(1 - \frac{x}{L}\right). \tag{2}$$

Suppose that our solution procedure needs the gradient of a function $f = u^2$ with respect to the unknown parameters. An input for MATHEMATICA for this problem and the automatically generated FORTRAN code are as follows.

```
(* input for MATHEMATICA*)
SMSSet["v"]
ui=Array[u[#]&,3]
Ni={x/L,1-x/L,x/L (1-x/L)}//SMSReduce
udiscretized=Ni.ui//SMSReduce
f=udiscretized^2//SMSReduce
gradient=SMSD[f,#]&/@ui
SMSExport[gradient,"gradf","g"]
SMSWrite["test",Common->{"u"[3],"x","L"}]
```

```
C....Fortran code
SUBROUTINE gradf(g)
REAL*8 v(6),g(3),u(3),x,L
COMMON/SMS/v,u,x,L
v(3)=x/L
v(4)=1.d0 - v(3)
v(5)=v(3)*v(4)
v(6)=2.d0*(u(1)*v(3) + u(2)*v(4) + u(3)*v(5))
g(1)=v(3)*v(6)
g(2)=v(4)*v(6)
g(3)=v(5)*v(6)
END
```

Three characteristic steps can be recognized:

(i) At the beginning of the session the SMSSet function initialized the internal global data base.

(ii) During the session the SMSReduce function is applied at the end of the input expressions. SMSReduce performs simultaneous optimization. The SMSD function performs automatic differentiation which will be explained later.

(iii) At the end of the session the SMSWrite function writes the contents of the global vector of formulas to the file in a prescribed language format (e.g. FORTRAN or C).

The way how the intermediate variables are labeled is crucial for the interaction between the SMS and MATHEMATICA. New intermediate variables are labeled consecutively $(v_1, v_2, v_3, \ldots, v_n)$ in the same order as they are created, and these labels remain fixed during the MATHEMATICA session. This enables free manipulation with the formulas returned by the SMS system. With MATHEMATICA user can perform various algebraic transformations on a optimized formulas independently on SMS. Although intermediate variables are named consecutively, they are not always stored in a data base in the same order. Indeed, when two formulas contain common sub-expression, SMS immediately replaces sub-expression with a new intermediate variable which is stored in the data base in front of the considered formulas. The internal representation of the formulas in the data base can be continuously changed and optimized.

All labels can be changed at the end of the MATHEMATICA sessions, before the generation of the numerical code. Thus, at the end of the session all the formulas stored in the data base are reconsidered by performing those correctness-preserving transformations which are restricted during the session (see also subsection on automatic differentiation).

2.2. *Improved optimization procedures with stochastic evaluation of expressions*

In real mechanical problems it is almost impossible to recognize the identity of two expressions (e.g., the symmetry of the tangent stiffness matrix in nonlinear mechanical problems) automatically only by the pattern matching mechanisms. Normally, our goal is to recognize the symmetry automatically without introducing additional knowledge into the derivation such as tensor algebra, matrix transformations, etc. Commands in Mathematica such as Symplify, Together, Expand, etc. are useless in the case of large expressions. Additionally, these commands are efficient only when the whole expression is considered. When optimization is performed simultaneously with the derivation of the formulas, the explicit form of the expression is obviously lost, since some parts are replaced with intermediate variables. The only possible way at this stage of computer technology seems to be an algorithm which finds equivalence of expressions numerically. This relatively old idea (see, e.g., [14] or [4]) is rarely used, although it is essential for dealing with especially hard problems. However, numerical identity is not a mathematically rigorous proof for the identity of two expressions. Thus the correctness of the simplification can be determined only with a certain degree of probability. With regard to our experience this can be neglected in mechanical analysis when dealing with more or less 'smooth' functions. In other cases, expressions have to be evaluated with a characteristic set of examples. Searching for common sub-expressions means that only those expressions which are syntactically equal are recognized as common sub-expressions. Due to the stochastic evaluation algorithm, also some higher relations can be searched.

Higher relations between expressions automatically recognized by the SMS system are:

(a) integer values

$$e_1 := f_1 \;\Rightarrow\; e_1 := N$$

(b) opposite values

$$\left.\begin{array}{l} e_1 := f_1 \\ e_2 := -f_1 \end{array}\right\} \Rightarrow \left\{\begin{array}{l} e_1 := f_1 \\ e_2 := -e_1 \end{array}\right.$$

(c) inverse values

$$\left.\begin{array}{l} e_1 := f_1 \\ e_2 := \dfrac{1}{f_1} \end{array}\right\} \Rightarrow \left\{\begin{array}{l} e_1 := f_1 \\ e_2 := \dfrac{1}{e_1} \end{array}\right.$$

(d) intersections of common factors for multiplication and addition.

$$\begin{array}{l} e_1 := a_i \circ b \circ c \circ d_j \\ e_2 := \bar{a}_i \circ b \circ c \circ \bar{d}_j \end{array} \;\Rightarrow\; \begin{array}{l} v_1 := b \circ c \\ e_1 := a_i \circ d_j \circ v_1 \\ e_2 := \bar{a}_i \circ \bar{d}_j \circ v_1 \end{array}$$

where e_i are symbols, $a_i, \bar{a}_i, d_j, \bar{d}_j, b, c$ are arbitrary expressions or sub-expressions and v_i are new intermediate variables.

Subexpressions in the above cases do not need to be syntactically identical, which means that higher relations are recognized also in cases where term rewriting and pattern matching algorithms in MATHEMATICA fail.

2.3. Performing non-local operations

For 'non-local' operations, such as integration, the SMS system provides a set of functions which perform optimization in a 'smart' way. 'Smart' optimization means that only those parts of the expression that are not important for the implementation of 'non-local' operation are replaced by new intermediate variables.

Let us consider an expression f which depends on variables x, y and z:

$$f = x^2 + 2xy + y^2 + 2xz + 2yz + z^2. \tag{3}$$

Since integration of f with respect to x is to be performed, we perform 'smart' optimization of f by preserving the integration variable x, which leads to the optimized expression f_x

$$f_x = v_5 + v_7 x + x^2 \tag{4}$$

and the following vector of intermediate variables v_i

$$v_4 := 2y, \qquad v_3 := y^2 + z^2 + z v_4, \qquad v_5 := 2z + v_4. \tag{5}$$

After the integration, the resulting expression f_i (6) is used to obtain another formula f_y (7). f_y is identical to f_i; however, with an explicit variable y. New format is obtained by 'smart' restoring the expression f_i with respect to variable y.

$$f_i = \int f_x \, \mathrm{d}x = v_5 x + \tfrac{1}{2} v_7 x^2 + \tfrac{1}{3} x^3, \tag{6}$$

$$f_y = v_9 + v_{10} y + x y^2. \tag{7}$$

At the end of the MATHEMATICA session the global vector of formulas contains the following intermediate variables

$$\begin{aligned} &v_4 := 2y, \qquad v_6 := z^2, \qquad v_3 := v_6 + y^2 + v_4 z, \\ &v_9 := 2z, \qquad v_5 := v_4 + v_9, \qquad v_8 := x^2, \\ &v_7 := v_6 x + \frac{x^3}{3} + v_8 z, \qquad v_{10} := v_8 + v_9 x. \end{aligned} \tag{8}$$

Input for MATHEMATICA for this example is as follows:

```
SMSSet[v]
f=(x+a+b)^2//Expand;
fx=SMSSmartReduce[f,x,Collect[#,x]&];
fi=Integrate[fx,x];
fy=SMSSmartRestore[fi,a,Collect[#,a]&];
```

2.4. *Automatic differentiation technique*

Differentiation is an arithmetic operation which plays the crucial role in the development of new finite elements. The procedure implemented in the SMS system represents a version of automatic differentiation technique [5]. The automatic differentiation generates a programing code for the derivative from a code for the basic function. The vector of the new intermediate variables, generated during the simultaneous simplification of the expressions, is a kind of 'pseudo' code, which makes the automatic differentiation with SMS possible. SMS uses MATHEMATICA's symbolic differentiation functions for the differentiation of explicit parts of the expression. The version of reverse or forward mode of 'automatic differentiation' technique is then employed on the global level for the collection and expression of derivatives of the variables which are implicitly contained in the intermediate variables.

Higher order derivatives are difficult to implement by the standard automatic differentiation tools. Most of the automatic differentiation tools offer only the first derivatives.

Table 1
Correct simplification of large expression

Unsimplified	Simplification A	Simplification B
$x := L(a)$	$x := L(a)$	$v_1 := L(a)$
$y := L(a) + x^2$	$y := x + x^2$	$x := v_1$
$\frac{dy}{dx} = 2x$	$\frac{dy}{dx} = 1 + 2x$	$y := v_1 + x^2$
		$\frac{dy}{dx} = 2x$

When derivatives are derived by SMS, the results and all the intermediate formulas are stored on a global vector of formulas where they act as any other formula entered by the user. Thus, there is no limitation in SMS concerning the number of derivatives which are to be derived.

Differentiation is an example where the problems involved in simultaneous simplification are obvious. Table 1 considers the simple example of the two expressions x, y and the differentiation of y with respect to x. $L(a)$ is an arbitrary large expression and v_1 is an intermediate variable. From the computational point of view, simplification A is the most efficient and it also gives correct results for both values x and y. However, when used in a further operations, such as differentiation, it obviously leads to the wrong results. On the other hand, simplification B has one more assignment and gives correct results also for the differentiation. To achieve maximal efficiency both types of simplification are used in the SMS system. During the derivation of the formulas the type B simplification is performed.

At the end of the derivation, before the FORTRAN code is generated, the formulas that are stored in a global vector are reconsidered to achieve the maximum computational efficiency. At this stage the type A simplification is used. Itermediate variables with 0 or constant value that eventually appear during the automatic differentiation are also eliminated and their values are inserted into the original formulas.

2.5. *Human interaction issues*

An important question arises: how to understand the automatically generated formulas? The automatically generated code should not act like a "black box". For example, after the using automatic differentiation tools we have no insight in the actual structure of the derivatives.

While formulas are derived automatically with SMS, SMS tries to find the actual meaning of the intermediate variables and assigns appropriate names. By asking MATHEMATICA in an interactive dialog about certain symbols, we can retain this information and explore the structure of the generated formulas. With the command `SMSRestore` the intermediate variables can be replaced by their definitions and the structure of the formulas can be explored at several levels. In the following MATHE-

MATICA session the structure of the array F is explored at three levels.

```
In[30]:= F
Out[30]= {{F    ,F    }, {F    ,F    }}
            11   12         21   22
In[31]:= SMSRestore[ F[[1,1]],1]
Out[31]= 1 + u
             ,X
In[32]:=SMSRestore[ F[[1,1]] ,2]
Out[32]= 1 + N     ui  + N     ui   + N   ui  + N    ui
             1,X   1     2,X   2    3,X  3    4,X  4
```

Sometimes SMS finds more than one meaning for the same intermediate variable. By default it presents the last one. With the command SMSFace, we can explore different meanings of the generated formulas. In the next example, SMS finds three different meanings for an element of the stiffness matrix: (a) K_{11} is an element of the matrix, (b) K_{11} is the first derivative of the weak form of the element $\partial R_1/\partial u_1$ and (c) K_{11} is the second derivative of the strain energy function $\partial^2 \Pi/\partial u_1 \partial u_1$.

```
In[40]:= K[[1,1]]
Out[40]= K
          11
In[41]:=SMSFace[All];
In[42]:= K[[1,1]]
Out[42]= K   | R        | pi
         11   1,ui        ,ui ,ui
                 1          1   1
```

3. Application to the finite element method

3.1. Characteristic arrays of finite elements

We limit ourselves to the class of problems characterized by the functional

$$\Pi(\boldsymbol{u}) = \int_{\Omega} \Pi^*(\boldsymbol{u})\,\mathrm{d}\Omega, \tag{9}$$

where $\boldsymbol{u}$ are the unknown functions and Ω the domain of the problem. It is assumed also that this functional can be expressed explicitly as a function of the geometric, material and physical (displacements, load, ...) characteristics of the problem. Many problems in mechanics of solids can be expressed in this way and they are typically solved by a finite element method. For more details see e.g. [25].

In this class of problems a high abstract formulation of the problem and its implementation with the SMS system is straightforward. Within the finite element method the domain of the problem is divided into sub-domains (finite elements). First, trial functions for unknown functions $\boldsymbol{u}$ are assumed on a domain of a single element (10). $\boldsymbol{G}$ is a matrix of the interpolation functions and $\boldsymbol{a}$ is a vector of the unknown parameters defined on the domain of a single element.

$$\boldsymbol{u} = \boldsymbol{G}\boldsymbol{a}. \tag{10}$$

The solution will be achieved by invoking the stationarity of the functional ($\delta\Pi = 0$), where unknown parameters have to be calculated. Assuming Newton's iterative method we need two characteristic arrays: a gradient and Hessian of the functional with respect to the unknown parameters. The number of all unknown parameters of the problem depends on the problem and it can be very large. On the contrary, the number of unknown parameters of the singe element remains constant and is usually less than 100. Thus, our goal is to generate symbolically only the characteristic arrays of a single element. Global gradient and Hessian are then assembled numerically. The resulting system of linear equations is also solved numerically.

For the gradient of the functional (11) with respect to the unknown parameters of the element, first the derivatives of the functional are needed ($\boldsymbol{\Psi}$ is often referred to as nodal force vector):

$$\boldsymbol{\Psi} = \frac{\partial \Pi}{\partial \boldsymbol{a}} = \left\{ \frac{\partial \Pi}{\partial a_i} \right\}. \tag{11}$$

Second derivatives of the functional with respect to the unknown parameters lead to the Hessian matrix ($\boldsymbol{K}$ is referred to as tangent stiffness matrix of the element).

$$\boldsymbol{K} = \left[\frac{\partial \boldsymbol{\Psi}}{\partial a_j} \right] = \left[\frac{\partial^2 \Pi}{\partial a_i \, \partial a_j} \right]. \tag{12}$$

Additionally, some futher derivatives of the functional (sensitivity vectors) are required for special purposes, such as inverse analysis, optimization procedures, etc.

$$\boldsymbol{S} = \frac{\partial \boldsymbol{\Psi}}{\partial p} = \left\{ \frac{\partial^2 \Pi}{\partial a_i \, \partial p} \right\}. \tag{13}$$

The evaluation of the above quantities includes also the integration over the element domain. Only simple linear finite elements can be integrated in closed form, which is the reason why the numerical integration will be employed. Procedures for numerical integration are general and they can be performed by a prearranged interface subroutine. There is no need for the automatic generation of integration procedures. In the case of numerical integration only numerical code for the evaluation of the characteristic quantities is needed at the integration point. These will be marked with the asterisk ($\boldsymbol{\Psi}^*, \boldsymbol{K}^*, \ldots$).

From the above it can be seen that for symbolic generation of the characteristic arrays the appropriate input for SMS has to be written with the definitions for

(i) discretization of the unknown functions;
(ii) derivation of the functional;

(iii) derivation of the gradient;
(iv) derivation of the Hessian.

3.2. *Interface between the automatically generated code and the FEM environment*

The result of SMS is a global vector of formulas where all characteristic arrays and intermediate formulas are stored together. By separating the global vector of formulas into several functions which are called on sequentially, formulas are evaluated only when and where they are really needed. For a typical mechanical problem the following subroutines are generated

(i) Const – evaluates constant values independent on the element data
(ii) Data – evaluates the values depended on the element data
(iii) Point – evaluates the values depended on the integration point coordinates
(iv) Force – evaluates values depended on the first equation counter (i, Ψ_i)
(v) Tangent – evaluates values depended on the second equation counter (j, K_{ij}).

A typical interface subroutine between automatically generated code and FEE is as follows:

```
called once for all elements
Const
called for every element
Data
for $g = 1,2,\ldots$,n_gauss_points
  Point
  for $i = 1,2,\ldots$,n_parameters
    Force($i$)
    for $j = 1,2,\ldots$,n_parameters
      Tangent ($i,j$)
```

A subprogram, which is nested deeper, assumes that all the quantities that can be evaluated before are already evaluated and stored in the vector of values. The vector of values is common for all subroutines. Analogous structures can be written also for the post-processing of the results, sensitivity analysis, etc. The total size of the program code for calculating all the characteristic element quantities lies between 5 and 30 kbytes which is in the range of the average manually written program code.

3.3. *Generation of characteristic formulas*

If $N_{\mathrm{d.o.f}}$ unknown parameters are used for the discretization of the unknown fields, then an explicit form of the gradient and the Hessian will have at least $N_{\mathrm{d.o.f}} + (N_{\mathrm{d.o.f}})^2$ terms. Thus, explicit code for all terms can be generated only for one-dimensional and low order two-dimensional elements. For three-dimensional elements and higher order two-dimensional elements the representative formulas for an arbitrary term of the gradient or Hessian have to be derived. Herein we shall take advantages of the automatic differentiation technique. Another possibility would be to explore the symmetric patterns (see [24]).

The subset of unknown parameters, denoted by $\boldsymbol{a}_i$, is defined by

$$\boldsymbol{a}_i \subset \boldsymbol{a}, \qquad \bigcup_{i=1}^{L} \boldsymbol{a}_i = \boldsymbol{a}, \qquad \boldsymbol{a}_i \cap \boldsymbol{a}_j = \emptyset, \quad i \neq j. \tag{14}$$

Let $\bar{a}_i$ be an arbitrary element of the ith subset. At the evaluation time of the program, the actual index of an arbitrary element $\bar{a}_i$ becomes known. Thus, $\bar{a}_{ij}$ represents an element of subset i with index j. In a similar manner $\boldsymbol{G}_i$ represents ith subset of the set of interpolation functions $\boldsymbol{G}$. Let quantity k be expressed as a linear combination of $\boldsymbol{a}_i$ and $\boldsymbol{G}_i$. For example k can be one of the unknown functions $\boldsymbol{u}$. The tensor calculus then leads the following relation

$$k = \boldsymbol{a}_i . \boldsymbol{G}_i \;\Rightarrow\; \frac{\partial k}{\partial \bar{a}_{ij}} = \bar{G}_{ij}. \tag{15}$$

Thus, the formula (15) is a representative formula for the evaluation of derivatives of k with respect to the arbitrary element of subset $\boldsymbol{a}_i$. The same principle can also be used for more complicated formulas. Suppose that the gradient of the function $f(k)$ with respect to $\boldsymbol{a}$ has to be evaluated. Then the number of subsets (denoted by L) is equal to the number of representative formulas needed for the evaluation of the gradient.

For the finite elements that apear in mechanics of solids, the number of subsets depends on the variational formulation, and is different for 2D and 3D elements. For the sake of simplicity we consider a general three-dimensional isoparametric finite element with n nodes and n shape functions N_i (for more details see, e.g., [25]). Displacement of the ith node is prescribed by the displacement vector $\{u_i, v_i, w_i\}$. Thus the following set of $3n$ unknown nodal parameters can be obtained

$$\boldsymbol{a} = \{u_1, v_1, w_1, \ldots, u_n, v_n, w_n\}^{\mathrm{T}} \tag{16}$$

and the displacement field $\boldsymbol{u} = \{u, v, w\}$ is discretized as follows:

$$u = \sum_{i=1}^{n} N_i\, u_i, \qquad v = \sum_{i=1}^{n} N_i\, v_i, \qquad w = \sum_{i=1}^{n} N_i\, w_i. \tag{17}$$

3.4. Description level 3

Let us first take a look at how derivation is done manually for a typical mechanical problem. When elements are derived manually, typically representative code for an arbitrary node can be created. The result is the well known $\boldsymbol{B}_i$ matrix which relates strains with displacements and has the dimension *number of strain components* × *number of nodal degrees of freedom*. The representative gradient (18) is a sub-vector of the whole gradient and has dimension 3. The Hessian (19) is a 3×3 sub-matrix of the tangent matrix of the element

$$\{\boldsymbol{\Psi}_i\}_{3\times 1} = \int_\Omega [\boldsymbol{B}_i]^{\mathrm{T}}_{6\times 3}\,\{\boldsymbol{\sigma}\}_{6\times 1}\,\mathrm{d}\Omega \quad i=1,2,\ldots,n \tag{18}$$

$$[\boldsymbol{K}_{ij}]_{3\times 3} = \int_\Omega [\boldsymbol{B}_i]^{\mathrm{T}}_{6\times 3}\,[\boldsymbol{C}]_{6\times 6}\,[\boldsymbol{B}_j]_{6\times 3} + \ldots\,\mathrm{d}\Omega, \quad i,j = 1,2,\ldots,n. \tag{19}$$

Thus, the number of subsets of the manually made code is $L=3$. It is interesting to note that this is also an optimal choice for the automatic code generation.

Unknown nodal parameters can be organized by the spatial dimensions into 3 sets with n elements as follows:

$$\boldsymbol{a}_1 = \{u_1,\ldots,u_n\}^{\mathrm{T}}, \qquad \boldsymbol{a}_2 = \{v_1,\ldots,v_n\}^{\mathrm{T}}, \qquad \boldsymbol{a}_3 = \{w_1,\ldots,w_n\}^{\mathrm{T}}. \tag{20}$$

Only one set of interpolation functions $\boldsymbol{G}$ (22) is needed for the interpolation of the displacements

$$u = \boldsymbol{G}\,\boldsymbol{a}_1, \qquad v = \boldsymbol{G}\,\boldsymbol{a}_2, \qquad w = \boldsymbol{G}\,\boldsymbol{a}_3, \tag{21}$$

$$\boldsymbol{G} = \{N_1,\ldots,N_n\}. \tag{22}$$

A high abstract description, introduced in the previous section, can now be directly implemented, leading to the representative 3×1 sub-vector of gradient (23) and the representative 3×3 sub-matrix of Hessian (24).

$$\{\boldsymbol{\Psi}_i^*\}_{3\times 1} = \begin{Bmatrix} \dfrac{\partial \Pi^*}{\partial \bar{a}_{1i}} \\ \dfrac{\partial \Pi^*}{\partial \bar{a}_{2i}} \\ \dfrac{\partial \Pi^*}{\partial \bar{a}_{3i}} \end{Bmatrix}, \quad i=1,2,\ldots,n, \tag{23}$$

$$[\boldsymbol{K}_{ij}^*]_{3\times 3} = \begin{bmatrix} \dfrac{\partial \boldsymbol{\Psi}_i^*}{\partial \bar{a}_{1j}} & \dfrac{\partial \boldsymbol{\Psi}_i^*}{\partial \bar{a}_{2j}} & \dfrac{\partial \boldsymbol{\Psi}_i^*}{\partial \bar{a}_{3j}} \end{bmatrix}^{\mathrm{T}}, \quad i,j=1,2,\ldots,n. \tag{24}$$

3.5. Some other description levels

Organizing the unknown nodal parameters by spatial dimensions, such as in Eq. (20), is not the only possible way. Generally, arbitrary subsets can taken from the set of all unknown parameters. However, only a few of those possibilities are of practical interest.

One subset: One characteristic formula for arbitrary element of gradient and Hessian. In this case all unknown parameters are treated equally and no partitioning is needed.

$$\boldsymbol{a} = \{u_1, v_1, w_1,\ldots,u_n, v_n, w_n\}^{\mathrm{T}}. \tag{25}$$

Since there is only one set of parameters, $\bar{a}$ represents an arbitrary parameter and parameter $\bar{a}_i$ with the index i, determined at the evaluation time. The formulation leads to one formula for arbitrary elements of the gradient (26) and the Hessian (27).

$$\Psi_i^* = \frac{\partial \Pi^*}{\partial \bar{a}_i}, \quad i = 1,2,\ldots,3n; \tag{26}$$

$$K_{ij}^* = \frac{\partial \Psi_i^*}{\partial \bar{a}_j}, \quad i,j = 1,2,\ldots,3n. \tag{27}$$

3n subsets: Full explicit form of the gradient and the Hessian.

For elements with a small number of unknown parameters the easiest way is to generate explicit code for all elements of gradient and Hessian. Again no partitioning is needed and the unknown parameters can be treated as ordinary variables leading to

$$\{\boldsymbol{\Psi}^*\}_{3n} = \left\{ \begin{matrix} \dfrac{\partial \Pi^*}{\partial a_1} \\ \cdots \\ \dfrac{\partial \Pi^*}{\partial a_{3n}} \end{matrix} \right\}, \tag{28}$$

$$[\boldsymbol{K}^*]_{3n\times 3n} = \begin{bmatrix} \dfrac{\partial \Psi_1^*}{\partial a_1} & \cdots & \dfrac{\partial \Psi_1^*}{\partial a_{3n}} \\ \cdots & \cdots & \cdots \\ \dfrac{\partial \Psi_{3n}^*}{\partial a_1} & \cdots & \dfrac{\partial \Psi_{3n}^*}{\partial a_{3n}} \end{bmatrix}. \tag{29}$$

3.6. General SMS routine

In the previous section the general notation for the description of finite elements has been introduced. The actual implementation depends on the type of element and variational formulation. In this section the schematic SMS procedure for deriving the gradient and the Hessian of an arbitrary functional for an arbitrary number of subsets L is described. The arbitrary element can be derived in the following 6 steps:

Step 1: Definition of the sets of degrees of freedom a_i and the sets of generalized interpolation functions G_i with the command SMSNewGroup.

$$a_i = \texttt{SMSNewGroup}[\{a_{i1}, a_{i2}, \ldots\}], \quad i = 1,2,\ldots,L;$$

$$G_i, = \texttt{SMSNewGroup}[\{g_{i1}, g_{i2}, \ldots\}], \quad i = 1,2,\ldots,L.$$

Step 2: Trial function expansion of unknown functions u_i as linear combination of unknown parameters and interpolation functions ($u_i = \texttt{SMSDot}[G_j, a_j]$).

Step 3: Definition of a functional.

Step 4: For every set of unknown parameters create two indexes. The first one ($\bar{a}_{ik_i}$) represents an arbitrary element of the ith set which will be replaced at the evaluation time by index k_i. The second one ($\bar{a}_{il_i}$) will be replaced at the evaluation time by index l_i.

$$\bar{a}_{ik_i} = \texttt{SMSNewIndex}[a_i, k_i], \quad i = 1,2,\ldots,L;$$

$$\bar{a}_{il_i} = \texttt{SMSNewIndex}[a_i, l_i], \quad i = 1,2,\ldots,L.$$

Step 5: Derive L characteristic formulas for the gradient and L^2 characteristic formulas for the Hessian. The derivation is performed by automatic differentiation with respect to an arbitrary element of the set.

$$\boldsymbol{\Psi}^* = \{\texttt{SMSD}[\texttt{functional}^*, \bar{a}_{ik_i}], \ldots\}, \quad i = 1,2,\ldots,L;$$

$$\boldsymbol{K}^* = \{\texttt{SMSD}[\boldsymbol{\Psi}^*, \bar{a}_{il_i}], \ldots\} \texttt{// Transpose}, \quad i = 1,2,\ldots L.$$

Step 6: Automatic generation of FE subroutines.

Let us once again consider the example presented in Section 2.1. The code presented there is generated without the use of subsets. This time all parameters are grouped together in one set. A code for the evaluation of the gradient, automatically separated by the SMS system into the subroutines, is presented.

```
SUBROUTINE constant
REAL*8 v(4),u(3),x,l
COMMON/SMS/v,x,u,l
v(1) =x/l
v(2) =1-v(1)
v(3) =v(1)*v(2)
v(4) =v(1)*u(1)+v(2)*u(2)+v(3)*u(3)
END
SUBROUTINE gradfi(i,gi)
REAL*8 v(4),u(3),x,l,gi
COMMON/SMS/v,x,u,l
gi =2*v(4)*v(i)
END
```

4. Finite element example

In this section, the automatically generated code is compared with the carefully manually written code. However, a manually written code is never totally objective and consequently comparisons are also not objective. The code used here follows the conventional finite element procedure as described by Zienkiewicz and Taylor [25], where the code written for the finite element environment FEAP [21] for similar elements can be found.

The use of the SMS system will be introduced in the case of a three dimensional, eight nodes, isoparametric element. Large logarithmic strains and the Neo-Hook's hyperelastic material law [17] are considered, which makes the developed element geometrically and material nonlinear. Classical, displacement based variational formulation

Table 2
Comparison of the efficiency and code size

Method	Normalized time	Code size (kB)	Number of num. oper.
Hand coded	1	22	—
1 subset	1.94	15	98 427
3 subsets	0.57	11	33 610
24 subsets	0.53	60	30 622

is used. The element has 24 degrees of freedom. Details of the element formulation and full input for the SMS system and MATHEMATICA can be found in [10]. Herein, only the code size and the efficiency are compared.

The code was automatically generated for a number of subsets 1, 3, and 24 in a way as described in the previous section. In Table 2, the numerical efficiency of the solutions and the amount of the resulting program code in bytes are compared. Efficiency is assessed by a direct measurement of the execution time needed to evaluate the nodal force vector and the stiffness matrix. Since all elements are incorporated into FEAP and compiled with the same compiler, the results represent quite an objective measure of efficiency. As can be expected, more explicit formulas mean also more efficient code. Considerably improved efficiency from the number of subsets 1 to 3 is the consequence of partially separable functions. Displacement fields u, v and w are indeed discretized independently. From 3 to 24 only minor improvement can be observed. It is a consequence of more explicit formulas, without loops, while the number of numerical operations remains almost the same.

As opposed to efficiency, more subsets do not necessarily lead to larger code. The amount of the generated code is influenced by two processes: more subsets increase the amount of the code but at the same time more explicit formulas can be additionally simplified, which decreases the code size. As a result of both processes, the number of subsets 3 has the shortest code.

If both criteria, code size and efficiency, are taken into account, then the code generated by partitioning the unknown parameters into three subsets has the best overall performance. It is interesting to notice that the manually written code has similar structure.

5. Recent applications of the SMS system

The new package has already been successfully used for the development of the new 2D and 3D elements based on a modified enhanced strain method [10]. Among other problems, the well known hour-glassing problem of enhanced strain elements in the presence of large deformations was successfully solved with the help of the new system [11]. It was also applied to develop a 'spline' finite strip element for nonlinear analysis of the prismatic shell structures with the non uniform cross section [12]. In all those cases the symbolic approach significantly improves the element formulation, not only from the point of view of efficiency and reliability of the automatically generated program code, but also from the point of view of efficiency and reliability of the finite element formulation itself.

However, the high abstract description of the problem does not automatically mean more compact and efficient numerical code. Conventional FE technology is based on matrix algebra. It can be represented as a complicated sequence of vector and matrix operations as addition, multiplication, inversion, etc. Mathematical tools such as matrix algebra and tensor algebra are instruments which normally help the user to handle the problem of uncontrollable expression growth. With their help we can also detect important relations between the formulas which can dramatically reduce the entire derivation and improve efficiency of the final numerical code. These tools are basically not needed when the continuum problem is described on a high abstract level with the SAC systems. On the other hand, matching common sub-expressions and some higher relations, as described, can hardly be compared with the powerful mathematical tools. Thus, at the critical points of the derivation some structural knowledge of the problem might still be needed.

References

[1] T.Y. Chang, H.Q. Tan, D. Zheng, M.W. Yuan, Application of symbolic method to hybrid/mixed finite elements and computer implementation, Comput. Struct. 35 (1990) 293–299.

[2] P. Fritzson, D. Fritzson, The need for high-level programming support in scientific computing applied to mechanical analysis, Comput. Struct. 45 (1992) 387–395.

[3] E. Gallopoulos, E. Houstis, J. Rice, Problem-solving environments for computational science, IEEE Comput. Sci. Eng. 1 (1994) 11–23.

[4] G. Gonnet, New results for random determination of equivalence of expression, in: B.W. Char (Ed.), Proc. 1986 ACM Symp. on Symbolic and Algebraic Comp., Waterloo, Ont., 1986, pp. 127–131.

[5] A. Griewank, On Automatic Differentiation, Mathematical Programming: Recent Developments and Applications, Kluwer Academic Publisher, Amsterdam, 1989, pp. 83–108.

[6] J.A. Hulzen, Code optimization of multivariate polynomial schemes: a pragmatic approach, in: J.A.V. Hulzen (Ed.), IEUROCAL'83 Proc., Lecture Notes in Computer Science, vol. 162, Springer, Berlin, 1983.

[7] C.-Y. Yang, An algebraic-expressed finite element model for symbolic computation, Comput. Struct. 52 (5) (1994) 1069–1077.

[8] E. Kant, Synthesis of mathematical modeling software, IEEE Software 10 (1993) 30–41.

[9] E. Kant, A.S-H. Yau, R. Liska, S. Steinberg, A problem solving environment for generating certifiably correct programs to solve evolution equations, preprint.

[10] J. Korelc, Symbolic approach in computational mechanics and its application to the enhanced strain method, Doctoral Dissertation, Institut of Mechanics, TH Darmstadt, Germany, 1996.

[11] J. Korelc, P. Wriggers, Consistent gradient formulation for a stable enhanced strain method for large deformations, Eng. Comput. 13 (1) (1996) 103–123.

[12] J. Korelc, J. Banovec, Spline finite strip analysis of shells using longitudinal connections between strips, ZAMM Z. angew. Math. Mech. 75 (1995) 263–264.

[13] L. Leff, D.Y.Y. Yun, The symbolic finite element analysis system, Comput. Struct. 41 (1991) 227–231.

[14] W.A. Martin, Determining the equivalence of algebraic expressions by hash coding, in: S.R. Petrick (Ed.), Proc. 2nd Symp. on Symbolic and Algebraic Manipulation, Los Angeles, CA, ACM, NY, 1971, pp. 305–310.

[15] A.K. Noor, C.M. Andersen, Computerized symbolic manipulation in nonlinear finite element analysis, Comput. Struct. 13 (1981) 379–403.

[16] J. Nagabhushanam, C.J. Srinivas, G.H. Gaonkar, Symbolic generation of elemental matrices for finite element analysis, Comput. Struct. 42 (1992) 375–380.
[17] R.W. Ogden, Non-linear Elastic Deformations, Ellis Horwood, Chichester, UK, 1984.
[18] N. Sharma, Synthesis of sequential and parallel programs for finite element analysis, Ph.D. Dissertation, Kent State University, Kent, OH, 1992.
[19] H.J. Stetter, Tools for scientific computing, ZAMM Z. angew. Math. Mech. 73 (1993) 335–348.
[20] The ABAQUS Users's Manual, Hibbitt, Karlsson and Sorensen, 1080 Main Street Pawtucket, RI, USA.
[21] R.L. Taylor, FEAP – A finite element analysis program, Description and Users Manual, University of California, Berkeley, 1990.
[22] S. Wolfram, Mathematica: A System for Doing Mathematics by Computer, Addison-Wesley, Reading, MA, 1991.
[23] P.S. Wang, Symbolic computation and parallel software, Technical Report ICM-9109-12, Department of Mathematics and Computer Science, Kent State University, USA, 1991.
[24] P.S. Wang, Finger: a symbolic system for automatic generation of numerical programs in finite element analysis, J. Symbol Comput. 2 (1986) 305–316.
[25] O.C. Zienkiewicz, R.L. Taylor, The Finite Element Method, vol. 1, 4th ed., McGraw-Hill, London, 1989.