Rewriting Strategies for a Two-Scale Method: Application to Combined Thin and Periodic Structures

B. Yang*, W. Belkhir†‡, R. N. Dhara*, A. Giorgetti†‡ and M. Lenczner*
FEMTO-ST
* Département Temps-Fréquence, 26 chemin de l'Epitaphe, 25030 Besançon, France
† Département Informatique des Systèmes Complexes, 16 route de Gray, 25030 Besançon, France
‡ INRIA Nancy Grand-Est, CASSIS project, 54600 Villers-lès-Nancy, France
{bin.yang,walid.belkhir,rajnarayan.dhara,alain.giorgetti}@femto-st.fr, michel.lenczner@utbm.fr

***Abstract*—Multiphysics models of large arrays of micro- and nanosystems are too complex to be efficiently simulated by existing simulation software. Fortunately, asymptotic methods such as those based on two-scale convergence are applicable to homogenization of thin or periodic (i.e. array) structures. They generate simpler models tractable to simulation, but their application is long and requires a mathematical expertise.**

Our goal is to provide engineers with an implementation of this mathematical tool inside a modeling software. We follow therefore a multidisciplinary approach which combines a generalization and formalization effort of mathematical asymptotic methods, together with rewriting-based formal transformation techniques from computer science. This paper describes this approach, illustrates it with an example and presents the architecture of the software under design.

***Keywords*-Partial differential equations, two-scale derivation, symbolic computation.**

I. Introduction

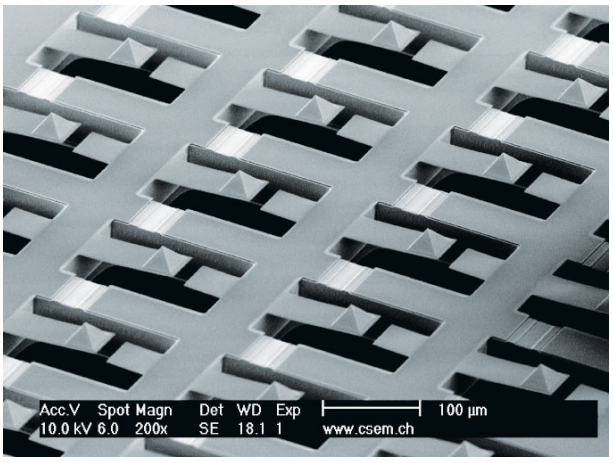


Fig. 1. An array of micro-levers for parallel AFM applications. A courtesy of A. Meister from CSEM Switzerland.

The context of this work is the design of microsystem array architectures, including microcantilevers, micromirrors, droplet ejectors, micromembranes, microresistors, biochips, to cite only a few. A typical example is an array of atomic force microscopes (AFM), as the one shown in Figure 1. AFM arrays have many applications, such as surface imaging in materials science, fundamental physics and biology, nano-object manipulation, or nanolithography. The numerical simulation of large microsystem arrays based on classical methods like the Finite Element Method (FEM) is prohibitive for today's computers (at least in a time compatible with the time scale of a designer). The calculation of a reasonably complex cell of a three-dimensional microsystem requires about 1000 degrees of freedoms which leads to about 10 000 000 degrees of freedoms for a 100×100 array.

A solution consists in approximating the model by a multiscale method. An asymptotic model is derived from a system of PDEs by taking into account that at least one parameter is very small. For a periodic structure the parameter is the small ratio of a cell size to the global size. It is thickness for a thin structure. The resulting model is another system of PDEs, obtained by taking the mathematical limits of the nominal model, in some energy sense, when the small parameters are going to zero. The resulting PDEs can be implemented in a simulation software such as the finite element based simulator COMSOL (Multiphysics Finite Element Analysis Software, official site http://www.comsol.com). Their simulation turns out to be dramatically faster while the approximation maintains a good level of precision. Since the method is rigorous and systematic, it also offers the advantage of a high reliability.

The literature in this field is vast and a large number of techniques have been developed for a large variety of geometric features and physical phenomena. However, none of them have been implemented in a systematical approach to render it available to engineers as a design tool. In fact, each published paper focus on a special case regarding geometry or physics, and very few works are considering a general picture. By contrast our aim is to treat the problem of systematic application of asymptotic methods by to implement them is a software constructing approximated models. This approach will cover many situations from a small number of bricks. It combines mathematical and computer science tools. The mathematical tool is the *two-scale transform* originally introduced in [1], [2], [3] to transform models combining periodic and thin structures. It is also referred as the *unfolding*

method. We have extended its domain of application to cover in the same time homogenization of periodic media, see for instance [4], and asymptotic analysis for thin domains, see [5]. The computer science tools include term rewriting [6], [7], [8], λ-calculus and types [9], [10], [11]. Compared to other techniques, ours requires more modular calculations and avoids any non-constructive proof.

Our software will aid to design multiphysics systems with high contrast in equation coefficients and complex geometries, including thin structures, periodic structures, with possibly many nested scales, and combinations of these. The produced simulations should be light enough to be inserted into parameter identification and optimization loops. The software is written in the symbolic computation language Maple™.

In the paper we present each aspect of our method as well as our first results. It is organized as follows. Section II introduces the mathematical tools used in model derivation and a running example. Section III details our design methodology, the architecture of our software and its theoretical foundations. Section IV presents a model derivation performed by the software and its validation.

II. Mathematical tools used for model derivation

To date, the program covers only the elliptic second order PDE,

$$-\sum_{i,j=1}^{d} \frac{\partial}{\partial x_i}(a_{ij}(x)\frac{\partial}{\partial x_j}u(x)) = f(x),$$

posed in a region Ω in $\mathbb{R}^d$ consisting of arbitrary number of subregions each of which may have a periodic structure, be thin or combining these two features. The coefficients a_{ij} of the equation may also be periodic. The boundary conditions can be Dirichlet conditions $u = g$ on a part Γ_1 of the boundary $\partial\Omega$ for a given g, and Neumann conditions

$$\sum_{i,j=1}^{d} n_j a_{ij}(x)\frac{\partial}{\partial x_j}u = h$$

for a given h on the other part Γ_2 of the boundary with outward unit vector $(n_1, .., n_d)$. This partial differential equation appears in a number of physical models such as the stationary diffusion equation, the equation of electrostatics or a model for elastic beam in torsion. We assume that all small parameters of the problem are expressed in terms of a single one denoted by ε.

For the sake of illustration, the remainder of this presentation is based on the example of thermal equilibrium of the AFM array shown in Figure 1, connected to a thin bar and placed in a vacuum. The complete structure is represented in Figure 2(a). The bar thickness is in the same range as the cell sizes. A heat source is placed in each cantilever to model the presence of actuations by thermo-elastic coupling. The heat flux at all boundaries is vanishing ($h = 0$), except at the bar end and at the left and bottom sides of the cantilever array where a temperature is imposed at $g = 0°C$. Modeling is done in two dimensions, so that the matrix a represents the two-dimensional thermal conductivity and f the two-dimensional density of thermal source.

Here, we do not detail the mathematical steps of construction of asymptotic model, but we just review the mathematical objects that are manipulated in order to explain, in Section III, how they are treated in terms of symbolic transformations. Several proofs have been published [12], [2], [3], [13], [14]. Here we adapt the proofs of the paper [13] where an effort has been made to formulate proofs in a modular form and to avoid any non-computational steps. The steps in this formal method are rigorously specified at a high level of generality, that make them independent of the domain geometry and applicable to other equations. The model derivation starts from the weak formulation associated to the above classical formulation boundary value problem. Find $u \in V$ such that

$$\sum_{i,j=1}^{d} \int_\Omega a_{ij}\frac{\partial u}{\partial x_i}\frac{\partial v}{\partial x_j}dx = \int_\Omega fvdx \qquad (1)$$

for all $v \in V$ where V represents the set of admissible functions $V = \{u \mid u = 0 \text{ on } \Gamma_1\}$. We observe that we have dropped the question of function regularity that should be taken into account in the set of admissible functions because this has not yet been taken into account in the implementation.

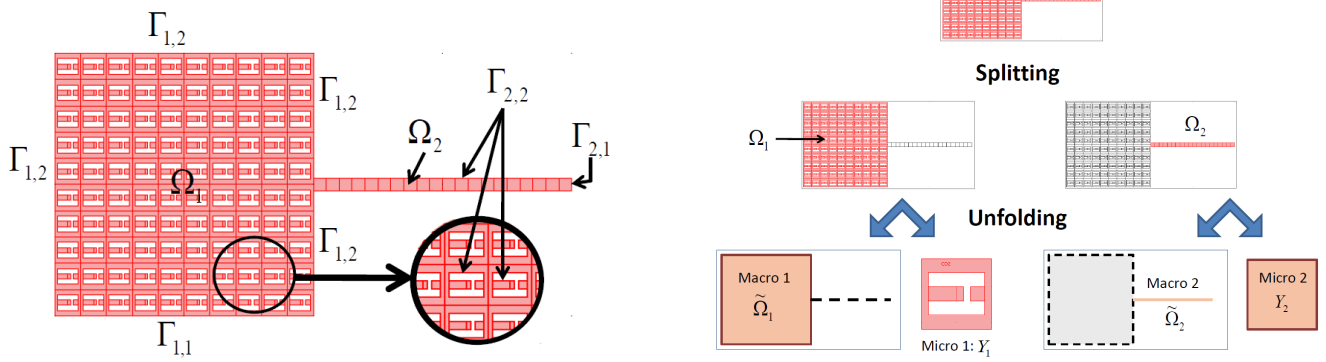
The weak formulation (1) requires the definition of regions as Ω and their boundaries as Γ_1 and Γ_2, of variables as $x = (x_1, .., x_d)$, of functions as $u(x)$ and $v(x)$, of set of functions as V, and then of operations associated with them. Beyond simple operations such as addition or multiplication between functions, it is required to define formally the integral over a region and the sum over a set of indices.

The usual operations on variational formulation require to use the extension to multidimensional domains of the rule of integration by parts. The so-called *Green formula* holds for sufficiently regular functions u and v defined in a domain $\Theta \subset \mathbb{R}^d$,

$$\int_\Theta u\ \frac{\partial v}{\partial x_i}\ dx = \int_{\partial\Theta} u\ v\ (n_x)_i\ ds(x) - \int_\Theta v\frac{\partial u}{\partial x_i}\ dx$$

for any $i = 1, ..., d$ where n_x represents the outward pointing unit normal of the hyper-surface volume element $ds(x)$ on the boundary $\partial\Theta$.

On the other hand, the model derivation relates to the two-scale transform whose principle is shown in Figure 2(b). The derivation starts with a phase of region splitting into several subregions (Ω_1 and Ω_2 in the example) requiring different treatments. The derivation continues with the unfolding step. In that step, the physical region is partitioned into a family of periodic cells, even if it does not undergo a simplification to the periodicity, and is transformed (by a process not detailed here) into the product of so-called *macroscopic* and *microscopic* regions. The macroscopic (resp. microscopic) region associated to a region Ω_i is denoted hereby $\widetilde{\Omega}_i$ (resp. Y_i). In Figure 2(b) parts removed from the initial model are indicated in gray or by doted lines.



(a) Geometry and boundary conditions for an array of cantilevers connected to a beam

(b) Flowchart representing the two-scale transformation of the region Ω

Fig. 2. Running example

The microscopic region contains all the information on the microstructure of the subregion while the macroscopic one is only used to locate the position of the cells; in the case of a thin structure its dimension is that of the neutral axis or plane. Figure 2(b) illustrates this construction process on the example where Ω_1 is a periodic region and Ω_2 a thin region. Their image pairs are denoted by $(\widetilde{\Omega}_1, Y_1)$ and $(\widetilde{\Omega}_2, Y_2)$ respectively. By composition with the above operation, a function defined on a subregion is transformed into a function defined on the product of its macroscopic and microscopic regions. This defines the linear operator T named the *two-scale transform*. For demonstration purposes, we also define the adjoint T^* of T. Since T is a linear operator which transforms a function defined on a subdomain Ω_1 into a product $\widetilde{\Omega}_1 \times Y_1$, conversely its adjoint for the L^2-inner product is defined from $\widetilde{\Omega}_1 \times Y_1$ to Ω_1, by the equality

$$\int_{\Omega_1} T^* w \; v \; dx = \int_{\widetilde{\Omega}_1 \times Y_1} w \; Tv \; dxdy$$

for w and v respectively defined in $\widetilde{\Omega}_1 \times Y_1$ and Ω_1.

To derive the approximate model, we require the concept of approximation of a function v by another function w in the sense of the small parameter ε, that we denote by $v = w + O(\varepsilon)$, together with the algebraic rules on $O(\varepsilon)$. Finally, the linearity of some operators has to be used repeatedly.

III. Software Architecture

This section presents our software design methodology and its theoretical foundations. It will be named MEMSALab, for "MEMS Arrays Laboratory". The software will transform a nominal multiphysic model, composed of a geometry and PDEs, into a multiscale (MS) asymptotic model approximating the initial one. This transformation depends on specifications about periodic and thin parts of the model, which are formulated by the designer.

The software will implement the sequence of modelling steps depicted in Figure 3. The software will be composed of modules. The Data Structure Constructor is in charge of importing the nominal model from a finite element analysis and simulation software package of physics and engineering applications, e.g. COMSOL. Conversely, a Geometry & Equation Generator will export the derivated multiscale model towards COMSOL after being adapted so that the simulation with COMSOL can be done effectively. Finally the results of this simulation will be brought back to the physical domain by means of a Inverse MS Transform.

The modelling step named Multiscale Data Structure Constructor is essentially devoted to the extension of the internal data structure with additional multiscale data. Its input is a symbolic representation of the geometry produced by the Data Structure Constructor from the numerical data from COMSOL. Data structures are detailed in Section III-A. The multiscale data structures are designed and implemented to be complete, flexible and able to describe complex geometries, but without redundancies. They describe the mathematical objects (e.g. functions, variables ...) and their properties, the geometrical regions and their boundaries, the admissible functional spaces, and the algebraic properties of the two-scale operators. The purpose of the modelling step named Multiscale (MS) Model Constructor is to automatically construct the asymptotic multiscale model from the multiscale data structures. Yet this purpose is not completely fulfilled since it requires a unification of multiscale methods. However two parts of this modelling step are already built and tested, namely a symbolic transformation package and a type-checker. Sections III-B and III-C respectively detail these two basic blocks and their theoretical foundations.

A. Data structures

Four kinds of data structures with increasing complexity have been defined.

The simplest ones are formal expressions in the host language Maple. These formal expressions are composed of symbols representing mathematical operations (such as sum, product, ...) or operators (such as partial derivation or integration) that are applied in a hierarchical way on similar expressions

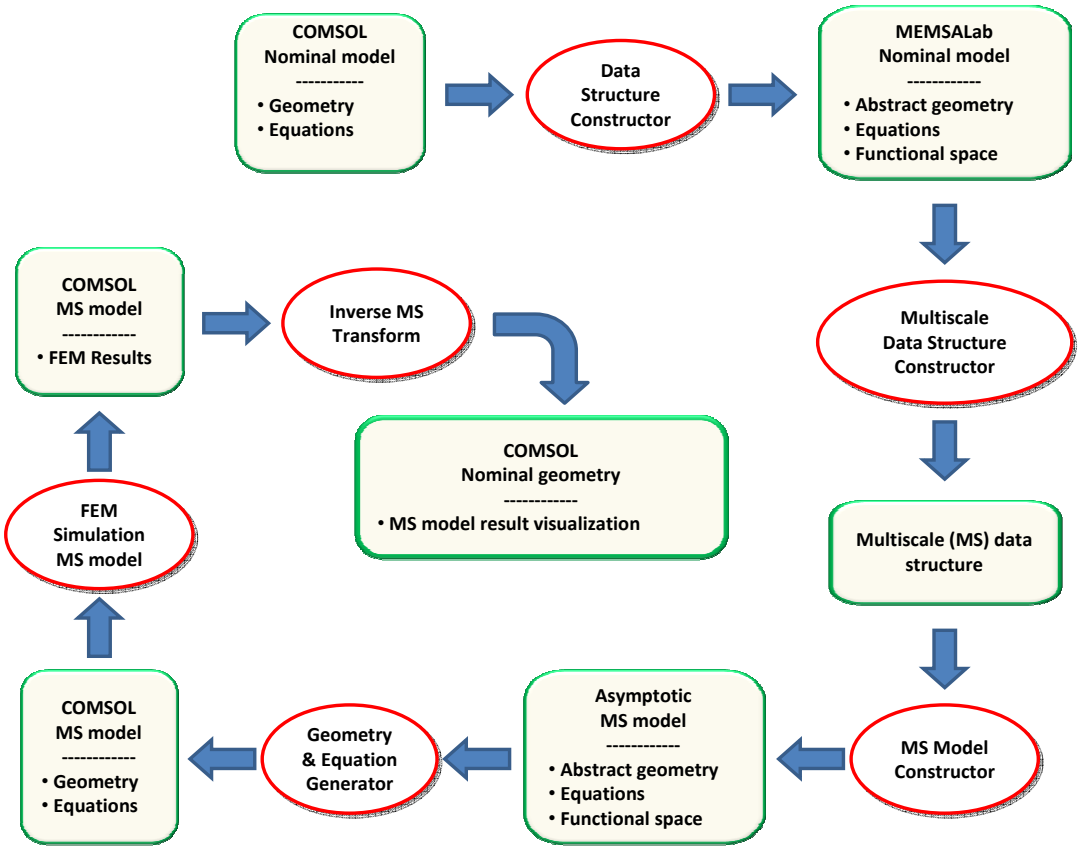


Fig. 3. Flowchart of the modelling steps

and on atomic symbols, such as region and (mathematical) variable names. For instance, the above weak formulation is represented by a single Maple expression with $=$ as top symbol. Most of these symbols are usefully predefined in Maple, but we adapt some of them and define new symbols (e.g. the two-scale operators). For instance, the Maple-defined symbol `int` for integration is not general enough to be used. We introduce a distinct integration symbol `Integral` for integration over a region and encode its semantics by equalities (data structures of the fourth kind, see below for details).

The second kind of data structures describes geometrical regions and their boundaries, mathematical functions and admissible functional spaces. They are stored in Maple arrays. Functional spaces are also formalized in the multiscale data structures and can be used and modified by the Multiscale Model Constructor.

Data structures of the third level are formal equalities (pairs of Maple expressions) encoding equational assertions such as boundary hypotheses. The fourth level of data structures represents the algebraic properties of the new operators (two-scale operators, integration over a region, etc) as systems of oriented equalities, a.k.a. rewriting strategies, as detailed in Section III-B.

B. Symbolic transformation package

Compared to other techniques, our multiscale method requires more modular calculations, avoids any non-constructive proof and intensively relies on equational reasoning. The classical way to automate equational reasoning is to consider mathematical equalities as rewrite rules. The rewrite rule $t \to u$ orients the equality $t = u$ from left to right and states that every occurrence of an instance of t can be replaced with the corresponding instance of u. Consequently symbolic computation with equalities is reduced to a series of term rewritings.

Term rewriting provides a theoretical and computational framework which is very useful to express, study and analyze a wide range of complex dynamic systems [15], [16]. It is characterized by the repeated transformation of a data object such as a word, term or graph. Transformations are combinations of rules specifying how to transform an object into another

one when it follows a specific pattern. Rule application can be restricted by additional conditions. Rules can be combined by specifying strategies controlling the order and the way rules are repeatedly applied. Term rewriting is used in formal semantics to describe the meaning of programming languages and more generally in computer science to describe program transformations and to perform automated reasoning. It is central in systems where the notion of *rule* is explicit such as expert systems, algebraic specifications, etc.

The computer algebra system Maple™ is widely used in the mathematical community and we have retained it for a prototypal implementation of the algorithmic aspects of multiscale methods. The Maple language is appropriate to combine function-based and rule-based symbolic transformations. Unfortunately, its rewriting function does not provide the user with control on how and where rules must be applied.

Algebraic computation and term rewriting are two research domains with strong similarities. Both are separately well-studied but there are only few works about the combination of algebraic computation and term rewriting [7], [17], [18], [19]. We have implemented a Maple package, named `symbtrans`, extending Maple with rule-based programming, where rules, strategies and usual Maple functions can be freely combined [19]. The transformation language provided by the `symbtrans` package is deliberately an adaptation for Maple of popular strategy languages such as ρ-log [8] or TOM [20].

The language of `symbtrans` offers the most common strategy constructors, as detailed in the following subsections. The combination of those strategies offers a simple and compact way for expressing sequences of symbolic derivations.

1) Rewriting at the top: In `symbtrans` the Maple™ statement `ruleName:=[l,r]` declares the rewrite rule $l \to r$ as a pair of terms and assigns it the name `ruleName`. The term l is called the *pattern* of the rule. The function `Transform` turns the pair `ruleName` into the partial function `Transform(ruleName)`. Given a term t, the rewrite rule $l \to r$ is applied on t at the top by the function application `Transform(ruleName)`(t). If the rule cannot be applied i.e. if t does not match its pattern l, then the exception `"Fail"` is raised. This is the standard *rewriting at the top* or *top rewriting* strategy.

Example 1. *Consider the property $\int v + w\ dx = \int v\ dx + \int w\ dx$ of linearity of the integral. The rewrite rule corresponding to its application from left to right can be defined with `symbtrans` as the pair*

```
IntegralLinearity := [
 Integral(A_ + B_, C_),
 Integral(A, C) + Integral(B, C)];
```

A convention in the package is that variable names in rule patterns end with "_" in order to distinguish them from constants. In order to apply the `IntegralLinearity` rule at the top to the term

```
t := Integral(v(x)+w(x),x);
```

we write `Transform(IntegralLinearity)(t)`. The resulting term is

```
Integral(v(x),x) + Integral(w(x),x)
```

2) Traversal strategies: A subterm of t that can be transformed by a strategy s is called a *redex* of t for s. The strategy `Outermost`(s) applies the strategy s once to all the redexes of t for s that are the closest ones to the root of t, i.e. to the largest subterms of t on which s succeeds. In other words the strategy `Outermost` traverses the term t down from its root and tries to apply s to each traversed subterm. If the strategy s succeeds on some subterm t' of t, then it is not applied to the proper subterms of t'. In particular, `Outermost`(s) fails iff s fails on all the subterms of t.

Example 2. *Let*

$$t = \underbrace{2 + \int v(x)\ dx + 3(\underbrace{\int w(x) + g(x)\ dx}_{r_1})\ dx}_{r_0}$$

be a term with two redexes r_0 and r_1 for the rule of integral linearity. Then `Outermost(IntegralLinearity)(t)` gives the expression

$$2 + \int v(x)\ dx + \int 3(\int w(x) + g(x)\ dx)\ dx,$$

since the rule `IntegralLinearity` is only applied to the outermost redex r_0 of t.

The strategy `Innermost`(s) works similarly, but in the opposite direction, i.e. it traverses a term t up from its smallest subterms and tries to apply the strategy s once to the smallest redexes of t for s.

Example 3. *For the term t of example 2, the expression `Innermost(IntegralLinearity)(t)` gives the expression*

$$2 + \int v(x)\ dx + 3(\int w(x)\ dx + \int g(x)\ dx)\ dx,$$

since the rule `IntegralLinearity` has been applied to the innermost redex r_1 of t.

3) Fixed-point computation: Applying a strategy repeatedly to a term, until it is no more applicable, is called a *fixed-point computation*. It is a strong and useful strategy, but presents a risk of non termination. The `symbtrans` package provides the strategy combinator `STNormalizer` for this strategy. Its functionality is illustrated by the following example.

Example 4. *The theory of convergence corresponds to the*

following rewrite rules:

$$O(\varepsilon) + O(\varepsilon) \to O(\varepsilon), \tag{2}$$

$$\sum_{i=1}^{n} O(\varepsilon) \to O(\varepsilon), \tag{3}$$

$$\int_{\omega} O(\varepsilon)\, ds \to O(\varepsilon), \text{ and} \tag{4}$$

$$z \times O(\varepsilon) \to O(\varepsilon) \tag{5}$$

for a term z bounded with respect to ε. A choice between two or more strategies, where the next strategy is applied only if the current one fails, can be constructed by the strategy combinator `LeftChoice`*. The strategy combinators* `STNormalizer` *and* `LeftChoice` *can be combined as follows*

```
ConvergenceStrategy :=
  STNormalizer(
    LeftChoice([
      Outermost(OEpsilonSum),
      Outermost(OEpsilonSUM),
      Outermost(OEpsilonIntegral),
      Outermost(OEpsilonConst)
    ])
  );
```

where `OEpsilonSum`*,* `OEpsilonSUM`*,* `OEpsilonIntegral` *and* `OEpsilonConst` *are* `symbtrans` *rules which respectively encode the rewriting rules (2), (3), (4) and (5). The result* `ConvergenceStrategy` *is a powerful strategy that reduces $O(\varepsilon)$ terms as much as possible. In the present case it can be shown that it always terminates and thus can be systematically applied after each tranformation step.*

4) Procedural programming: It is possible to combine rewrite rules and procedural programming in `symbtrans`. That is, the right-hand side of a rewrite rule may contain calls of Maple predefined functions or of functions defined by the user. This feature is not available in pure rewriting languages, e.g. in Maude [21]. But it is available in rewriting languages built upon a host language, e.g. ρ-log [8] which is built on Mathematica and Tom [20] which is built on Java.

Example 5. *The code*

```
e := [L_, DelayEval -> nops(L)];
```

is a rule declaration whose right-hand side contains a call of the Maple™ function `nops(L)` *that returns the number of arguments of* `L`*. In particular, it returns the size of* `L` *when* `L` *is a list.* `DelayEval` *is a reserved keyword in* `symbtrans`*, whose presence is necessary when the right-hand side of the rewrite rule contains function calls.*

5) Conditional rewriting: Rewrite rules can be conditional. A conditional rule is a rewrite rule $l \to r$ and a condition c on the variables in l. In `symbtrans` a conditional rule is declared as a list `[`l`,`r`,`c`]` of three elements, where the third element is the condition. Again the `DelayEval` mechanism is required in the condition when it contains function calls.

Example 6. *The conditional rewrite rule*

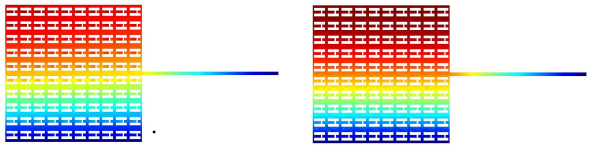
```
[L(A_*X_),A*L(X),DelayEval->IsScalar(A)]
```

corresponds to the linearity of the operator L. It replaces $L(A\,x)$ with $A\,L(x)$ if A is a scalar.

C. Type-checker

This section shows the interest of a typing discipline for the formalization of multiscale methods. In computer science type systems complete grammatical descriptions of programming languages with additional contextual conditions of *well-formedness*. For example, the fact that a mathematical expression under an integration sign has to be integrable is a contextual information that is better detected by well-formedness rules than by grammatical rules. The set of rules that relate the type of a mathematical expression with the one of its sub-expressions is called a *type system*. The reader is invited to see [11], [10] for more information on type theory. We propose a first type system in which region names (Ω, Ω_1, Ω_2, $\widetilde{\Omega}_1$, $\widetilde{\Omega}_2$, Y_1 and Y_2 in the example) are considered as *type constants*, i.e. atomic type names. Another type constant is $\mathbb{R}$ for real numbers. The notation $e : A$ asserts that e is an expression of type A. The type $A \to B$ denotes the set of functions whose domain is A and co-domain is B. Moreover, any Cartesian product of region names also forms a type. For instance, the functions w defined on $\widetilde{\Omega}_1 \times Y_1$ in Section II are expressions of type $\widetilde{\Omega}_1 \times Y_1 \to \mathbb{R}$. It is denoted by $w : \widetilde{\Omega}_1 \times Y_1 \to \mathbb{R}$.

For any region Ω, any variable x such that $x : \Omega$ and any expression e such that $e : A$ or $e : A \to B$, where A is a Cartesian product composed of other regions than Ω, the expression $\int_\Omega e\, u\, dx$ (where u is any expression) can be rewritten as $\int_\Omega u\, dx$ and the expression $\partial e/\partial x$ as 0. Such simplification rules of major importance are made possible thanks to the typing information. The type checking process requires that the user declares the type of all the mathematical objects used in the proof. These objects are the constants, variables, functions, $O(\varepsilon)$ terms, etc. Their type is stored in the data structures described in Section III-A. For all the possible software inputs the types of the standard mathematical operators such as derivation and integration are predefined in the type system. For each region Ω we can define the type of the two-scale transform T_Ω of functions defined on Ω as $(\Omega \to \mathbb{R}) \to (((\nu(\Omega) \times \mu(\Omega)) \to \mathbb{R})$ where $\nu(\Omega)$ (resp. $\mu(\Omega)$) denotes the macroscopic (resp. microscopic) region of Ω. But it is more generic and compact to consider a single *polymorphic* two-scale transform $T : \Pi\alpha.(\alpha \to \mathbb{R}) \to (((\nu(\alpha) \times \mu(\alpha)) \to \mathbb{R})$, where $\Pi\alpha$ is the *universal quantification* over the *type variable* α [10]. In the present case, this variable can only be replaced by a region name and this replacement produces an instance of this polymorphic operator. This replacement is called the *universal application* (or *extraction*), that is, if t is a term of type $\Pi\alpha.U$, and β is a type, then $t(\beta)$ is a term of type



(a) Nominal model (b) Two-scale model

Fig. 4. Results for u^0 by FEM simulations

$U[\alpha := \beta]$, where $U[\alpha := \beta]$ is the substitution of α by β in U. The polymorphic adjoint of the polymorphic two-scale transform T is $T^* : \Pi\alpha.(((\nu(\alpha) \times \mu(\alpha)) \to \mathbb{R}) \to (\alpha \to \mathbb{R})$.

For multiscale methods, regularity conditions could additionnally be checked by a richer type system. Regularity conditions for a mathematical function (i.e. derivability and integrability) are usually described by its membership to an adequate functional space.

IV. Validation

When applying our software to Eq. (1), we specify that the two-scale transforms to be used are T^1 and T^2 corresponding to Fig. 2(b). The two-scale model provided by the software includes the microscopic equation posed in the Y_1 cell with the coefficient $a^1 = T^1 a$, the function space $W^1 = \{v$ defined in Y_1 v is Y_1-periodic$\}$ and the variational formulation:

Find $w_i^1 \in W^1$ such that

$$\sum_{k,\ell=1}^{2} \int_{Y_1} a_{k\ell}^1 \frac{\partial w_i^1}{\partial y_k} \frac{\partial v}{\partial y_\ell} dy = -\sum_{k,\ell=1}^{2} \int_{Y_1} a_{k\ell}^1 \delta_{ik} \frac{\partial v}{\partial y_\ell} dy$$

for all $v \in W^1$. It also includes a problem posed in the microscopic cell of the thin part Y_2 even if the latter has not the periodicity feature. This problem is written in a similar manner not detailed here and leads to the definition of a^2 and w_i^2. It follows the definition of homogenized thermal diffusion coefficient and density of heat source per unit area in the first macroscopic part $\widetilde{\Omega}_1$

$$a_{ij}^H = \sum_{k,\ell=1}^{2} \frac{1}{|Y_1|} \int_{Y_1} a_{k\ell}^1 (\delta_{ik} + \frac{\partial w_i^1}{\partial y_k})(\delta_{j\ell} + \frac{\partial w_j^1}{\partial y_\ell}) dy$$

and $f^H = \frac{1}{|Y_1|} \int_{Y_1} T^1 f dy$. Similar formula yields a^H and f^H in $\widetilde{\Omega}_2$ the second macroscopic part. The two-scale model also includes the macroscopic problem posed on $\widetilde{\Omega}_1 \cup \widetilde{\Omega}_2$ the macroscopic region with $V^0 = \{v$ defined in $\widetilde{\Omega}_1 \cup \widetilde{\Omega}_2 \mid v = 0$ on $\Gamma_1 \cap (\widetilde{\Omega}_1 \cup \widetilde{\Omega}_2)\}$ the macroscopic admissible function space and the variational formulation: Find $u^0 \in V^0$ such that

$$\sum_{i,j=1}^{2} \int_{\widetilde{\Omega}_1} a_{ij}^H \frac{\partial u^0}{\partial \widetilde{x}_i^1} \frac{\partial u^0}{\partial \widetilde{x}_j^1} d\widetilde{x}^1 + \int_{\widetilde{\Omega}_2} a_{11}^H \frac{\partial u^0}{\partial \widetilde{x}_1^2} \frac{\partial u^0}{\partial \widetilde{x}_1^2} d\widetilde{x}_1^2 = \int_{\widetilde{\Omega}_1} f^H v^0 d\widetilde{x}^1 + \int_{\widetilde{\Omega}_2} f^H v^0 d\widetilde{x}_1^2$$

for all $v \in V^0$. The final approximation in the physical region Ω is

$$u \approx B^1(u^0 + \varepsilon \sum_{i=1}^{2} (y_i + \overline{w}_i^1) \frac{\partial u^0}{\partial \widetilde{x}_i^1}) \text{ in } \Omega_1,$$

where

$$\overline{w}_i^1 = w_i^1 - \frac{1}{|Y_1|} \int_{Y_1} w_i^1 dy$$

and B^1 is the approximate inverse two-scale transform defined for any Y_1-periodic function $v(\widetilde{x}^1, y)$ by

$$(B^1 v)(x) = v(\widetilde{x}^1, \frac{x - \varepsilon/2}{\varepsilon}).$$

A similar approximation is built in Ω_2. Now we compare the results obtained with the nominal model and the simplified model, see Fig. 4. To obtain sufficient accuracy of calculations with the software COMSOL we used meshes with 459,840 elements to model the complete problem, of 7664 elements for both microscopic problems and with 305 elements for the macroscopic problem. The relative quadratic error

$$\left(\int_\Omega |u - u^0|^2 dx \right)^{1/2} / \left(\int_\Omega |u|^2 dx \right)^{1/2}$$

between u, the model nominal solution, and u^0, this of the homogenized model, is 6.85%. The computing time on a laptop for the three problems are respectively 3.734s, 0.046s and 0.015s, giving a ratio of computation time of the nominal model and simplified model of 0.029.

V. Conclusion

We presented the mathematical tools involved in the two-scale approximation of models combining thin and periodic structures, the basic principles of a new software dedicated to the generation of multiscale models, and their illustration on an example of MEMS array. Each mathematical tool of the method, namely the integral and PDE calculus, the algebraic rules on $O(\varepsilon)$, and the algebraic properties of the two-scale operators are expressed in the software by means of the notion of strategy. Thanks to the notion of strategy the formal derivation of an approximated model is close to the mathematical one in terms of size and nature of steps. Among other advantages the symbolic transformation language smoothly integrates with standard Maple functions. Consequently, the Maple programmer learns it quickly and is free to mix procedural- and rule-based programming styles. Moreover all the features of her development environment (such as refactoring, code completion, dependency analyses, etc) are preserved for free.

The symbolic transformation package and the type-checker have been successfully applied together to a couple of models. In particular, the asymptotic model from the running example has been produced from Eq. (1) with the assistance of these tools.

To conclude, we underline that there are a number of interests with our approach. Clearly, the final model is "correct-by-construction", human errors are avoided, and the model

derivation effort is dramatically reduced. In the point of view of a MEMS designer, this software will generate simplified analytical and numerical models corresponding to a design and to a set of requested simplifications. Since the computation will be fast enough, a variety of simplifications can be tested and the one satisfying the best tradeoff between simulation time and precision can be retained.

REFERENCES

[1] M. Lenczner, "Homogénéisation d'un circuit électrique," *C. R. Acad. Sci. Paris Sér. II b*, vol. 324, no. 9, pp. 537–542, 1997.

[2] J. Casado-Díaz, "Two-scale convergence for nonlinear Dirichlet problems in perforated domains," *Proc. Roy. Soc. Edinburgh Sect. A*, vol. 130, no. 2, pp. 249–276, 2000.

[3] D. Cioranescu, A. Damlamian, and G. Griso, "Periodic unfolding and homogenization," *C. R. Math. Acad. Sci. Paris*, vol. 335, no. 1, pp. 99–104, 2002.

[4] A. Bensoussan, J.-L. Lions, and G. Papanicolaou, *Asymptotic analysis for periodic structures*, ser. Stud. in Math. and its Appl. Amsterdam: North-Holland Publishing Co., 1978, vol. 5.

[5] P. G. Ciarlet, *Mathematical elasticity. Vol. I*, ser. Studies in Mathematics and its Applications. Amsterdam: North-Holland Publishing Co., 1988, vol. 20.

[6] Terese, *Term Rewriting Systems*, ser. Cambridge Tracts in Theor. Comp. Sci. Cambridge Univ. Press, 2003, vol. 55.

[7] H. Cirstea and C. Kirchner, "The rewriting calculus — Part I *and* II," *Logic Journal of the Interest Group in Pure and Applied Logics*, vol. 9, no. 3, pp. 427–498, May 2001.

[8] M. Marin and F. Piroi, "Rule-based programming with Mathematica," in *In Sixth Mathematica Symposium (IMS 2004)*, 2004, pp. 1–6.

[9] J.-L. Krivine, *λ-calculus, types and models*. Ellis Horwood, 1993.

[10] J.-Y. Girard, P. Taylor, and Y. Lafont, *Proofs and types*. New York, NY, USA: Cambridge University Press, 1989.

[11] H. Geuvers, "Introduction to type theory," in *Language Engineering and Rigorous Software Development*, ser. LNCS. Springer, 2009, vol. 5520, pp. 1–56.

[12] M. Lenczner and G. Senouci-Bereksi, "Homogenization of electrical networks including voltage-to-voltage amplifiers," *Math. Models Methods Appl. Sci.*, vol. 9, no. 6, pp. 899–932, 1999.

[13] M. Lenczner and R. C. Smith, "A two-scale model for an array of AFM's cantilever in the static case," *Mathematical and Computer Modelling*, vol. 46, no. 5-6, pp. 776–805, 2007.

[14] D. Cioranescu, A. Damlamian, and G. Griso, "The periodic unfolding method in homogenization," *SIAM Journal on Mathematical Analysis*, vol. 40, no. 4, pp. 1585–1620, 2008.

[15] P. Baldan, C. Bertolissi, H. Cirstea, and C. Kirchner, "A rewriting calculus for cyclic higher-order term graphs," *Mathematical Structures in Computer Science*, 2006.

[16] H. Cirstea, G. Faure, and C. Kirchner, "A rho-calculus of explicit constraint application," *The journal of Higher-Order and Symbolic Computation*, 2005.

[17] S. Fèvre and D. Wang, "Combining algebraic computing and term-rewriting for geometry theorem proving," in *Proceedings of the International Conference on Artificial Intelligence and Symbolic Computation*. London, UK: Springer-Verlag, 1998, pp. 145–156.

[18] R. Bündgen, "Combining computer algebra and rule based reasoning," in *Integrating Symbolic Mathematical Computation and Arti Intelligence. Proceedings of AISMC-2, volume 958 of LNCS*. Springer, 1995, pp. 209–223.

[19] W. Belkhir, A. Giorgetti, and M. Lenczner, "A Symbolic Transformation Language and its Application to a Multiscale Method," submitted for publication.

[20] E. Balland, P. Brauner, R. Kopetz, P. Moreau, and A. Reilles, "Tom: Piggybacking Rewriting on Java," in *RTA*, 2007, pp. 36–47.

[21] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, Eds., *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, ser. LNCS, vol. 4350. Springer, 2007.