Towards an automatic tool for multi-scale model derivation

Walid Belkhir, Nicolas Ratier, Duy Duc Nguyen, Bin Yang, Michel Lenczner, Frédéric Zamkotsian, Horatiu Cirstea

Towards an automatic tool for multi-scale model derivation

Walid Belkhir[1,2], Nicolas Ratier[3], Duy Duc Nguyen[1], Bin Yang[1], Michel Lenczner[1], Frédéric Zamkotsian[4] and Horatiu Cirstea [5]

[1]FEMTO-ST, Time and Frequency Department, University of Franche-Comté, 25000, Besançon, France
[2]INRIA Nancy - Grand Est, CASSIS project, 54600 Villers-lès-Nancy,France.
[3]FEMTO-ST, Time and Frequency Department, Technical University of Belfort-Monbéliard, 90010 Belfort, France.
[4]LAM-CNRS, 38 rue Frédéric Joliot-Curie 13388, Marseille, France.
[5]University of Lorraine - LORIA, Campus scientifique - BP 239 - 54506 Vandœuvre-lès-Nancy, France.

November 2, 2015

Abstract

This paper reports recent advances in the development of a symbolic asymptotic modeling software package, called MEMSALab, which will be used for automatic generation of asymptotic models for arrays of micro and nanosystems. More precisely, a model is a partial differential equation and an asymptotic method approximate it by another partial differential equation which can be numerically simulated in a reasonable time. The challenge consists in taking into account a wide range of different physical features and geometries e.g. thin structures, periodic structures, multiple nested scales etc. The main purpose of this software is to construct models incrementally so that model features can be included step by step. This idea, conceptualized under the name "*by-extension-combination*", is presented in detail for the first time.

1 Introduction

Many systems encountered in micro or nano-technologies are governed by differential or partial differential equations (PDEs) that are too complex to be directly simulated with general software. In a number of cases, the complexity of the simulation is due to a combination of many factors as several space scales or time scales, large coefficient heterogeneity or large aspect ratios. Many methods have been developed to overcome these difficulties, and in particular the asymptotic methods, also called perturbation techniques, constitute an active field of research in all fields of physics and mathematics for more than a century. Their application is based on a case-by-case approach so they are implemented only in specialized software. We adopt an alternate approach by developing a software package called `MEMSALab` (for MEMS Array Lab) whose aim is to incrementally derive

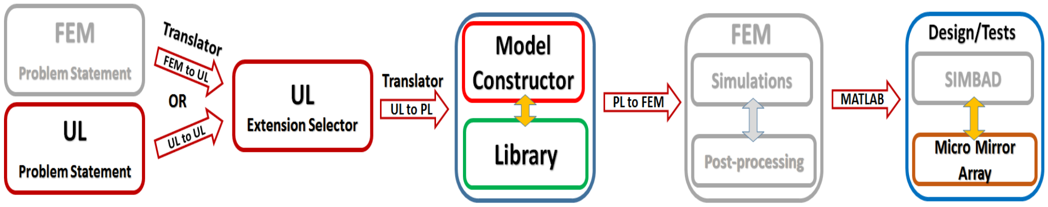

Figure 1: Flow of a `MEMSALab` Application.

asymptotic models for input equations by taking into account their own features e.g. the scalar valued or vector valued solution, different estimates on the solutions and sources, thin structures, periodic structures, multiple nested scales etc. The resulting model can be directly exploited in simulation software.

Our approach of the software development is two-fold. On one hand we develop computer science concepts and tools allowing the software implementation and on the other hand we derive and implement asymptotic models to anticipate the introduction of related modeling concepts in the software library. This paper is written in this spirit, it reports our latest advances in the development of the kernel of `MEMSALab`. The technique of model derivation relies on an asymptotic method taking into account the small ratio between the sizes of a cell and of the whole array [LS07]. It is not detailed since it is relatively long and technical, however we present some key facts giving an idea of the features playing a role in the derivation.

In [BGL14] we presented a transformation language implemented as a Maple package. It relies on the paradigm of rule-based programming and rewriting strategies as well as their combination with standard Maple code. We used this language to encode "by hand" the homogenized model of the stationary heat equation with periodic coefficients. Then, in [YBL14b] we introduced a theoretical framework for computer-aided derivation of multi-scale models. It relies on a combination of an asymptotic method with term rewriting techniques. In the framework [YBL14b] a multi-scale model derivation is characterized by the features taken into account in the asymptotic analysis. Its formulation consists in a derivation of a *reference proof* associated to a *reference model*, and in a set of *extensions* to be applied to this proof until it takes into account the wanted features. The reference model covers a very simple case i.e. the periodic homogenization model of a scalar second-order elliptic equation posed in a one-dimensional domain. The related reference proof is a series of derivations that turn a partial differential equation into another one. An extension transforms the tree structure of the proof as long as the corresponding feature is taken into account, and many extensions are composed to generate a new extension. The composition of several existing elementary extensions instead of the development of new extension transformations has the advantage of reducing the development effort by avoiding doing complex changes manually. This method has been applied to generate a family of homogenized models for second order elliptic equations with periodic coefficients that could be posed in multi-dimensional domains, with possibly multi-domains and/or thin domains. However, it is limited to extension not operating on the same positions of the three.

This limitation is due to an unsufficient formalization of the concept of extension. The present paper fills the gap, so the "*by-extension-combination*" method specifies what is meant by extension, also called generalization, and how it is implemented in terms of added context and/or parametrization. The clear statement allows defining rigorously the combination of extensions. Some key implementation aspects are discussed. The symbolic transformation language, also called "*Processing Language*", previously written in the Maple package is now in Ocaml to gain in development flexibility, to reduce the programming errors and to take advantage of a free environment. A "*User Language*" is now available for the specification of the proofs and the extensions but is not detailed here since it is not a key ingredient of the by-extension-combination method.

The general picture of the approach is shown in Figure 2.

At the level 1 there is the input PDE that corresponds to the reference model.

At the level 2 there is the proof of reference that, when applied to the input PDE gives an approximated model, which is another PDE.

At the level 3 there are extensions of proofs. The application of an extension to a proof gives a new proof that captures a new feature. The application of the resulting new proof to the input PDE gives another approximated PDE that covers the new feature.

At the level 4 Two (or many) extensions, each of which comes with a new feature, can be combined to generate a new extension that covers the new feature. Then, the resulting extension is applied to the reference proof, and the resulting proof is applied to the input PDE.

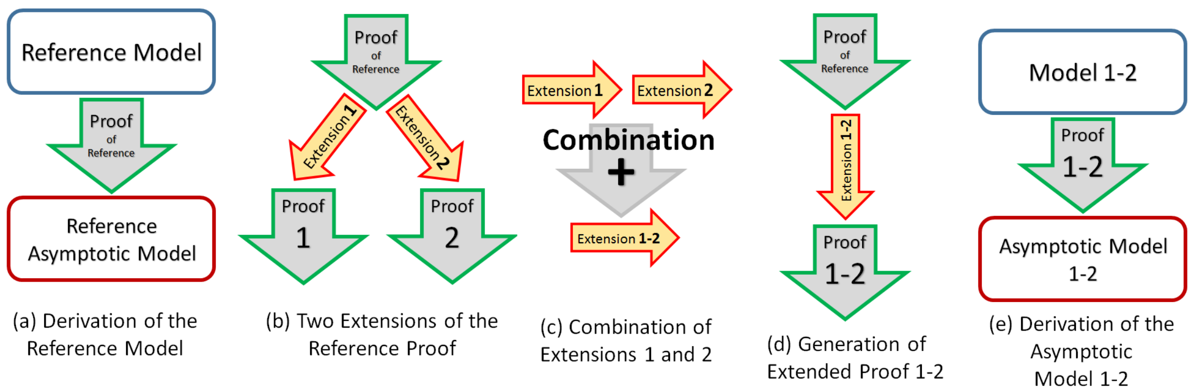

Figure 2: General picture of the extension-combination approach in MEMSALab.

Finally, we mention that our purpose is not to fully formalize the multi-scale model proofs as with a proof assistant e.g. Coq [BCHPM04], but to devise a methodology for an incremental construction of complex model proofs, as well as a tool that comes with such methodology. It is worth mentioning that the concept of proof reuse by means of abstraction/generalization and modification of formal proofs was investigated in many works e.g. [BL04]. Although the notion of unification is at the heart of our formalism as well as the works on the reuse of formal proofs by generalization, these works do not consider the combination of proofs. Finally, this approach is new at least in the community of multi-scale methods where asymptotic models are not derived by computer-aided combinations.

1.1 Organization of the paper

The paper is organized as follows: Section 2 introduces preliminary definitions. Namely, term rewriting and strategies. In section 3 we introduce the ideas behinds the by-extension-combination method. We formalize the extension as a transformation that preserves the mathematical semantics. In section 4 we show how to implement the extension operators as rudimentary operations, namely adding contexts at some positions of the input term, and parametrisation which consists in replacing some subterms by rewriting variables. We call this class of extension the position-based extensions. Then, we show how to combine position-based extensions. In section 5 we show how to implement the extension operators as rewriting strategies and we define their combination. In other words we define a subclass of rewriting strategies which is closed under combination. In section **??** we explain the principles of the user language in which one writes its proofs and extensions. In section 6 we present the reference proof and its extension model as well as the scripts written in the user language and the outputs.

Contents

2 Preliminaries

In this section we introduce concepts which will be used to formulate the extension mechanisms for the multi-scale model derivations. Precisely, we define the notion of term rewriting together with its underlying concepts: terms over a first order (many-sorted) signature, substitutions, rewriting rules and strategies. Then, we describe the concept of second-order rules and strategies operating on first order rewriting rules and strategies, and finally a grammar of mathematical expressions and proofs used in examples in the rest of the paper.

2.1 Term Rewriting

Definition 1 (Terms) *Let $\mathcal{F} = \cup_{n\geq 0}\mathcal{F}_n$ be a set of symbols called* function symbols*, each symbol f in $\mathcal{F}_n$ has an* arity *which is the index of the set $\mathcal{F}_n$ it belongs to, it is denoted $arity(f)$. Elements of arity zero are called* constants *and often denoted by the letters $a, b, c, \ldots$ It is always assumed that there is at least one constant. Occasionally, prefix or postfix notation for $\mathcal{F}_1$ and infix notation for $\mathcal{F}_2$ may be used. $\mathcal{F}$ is often called a set of ranked function symbols or a (unsorted or mono-sorted)* signature*. Given a (denumerable) set $\mathcal{X}$ of* variable *symbols, the set of (first-order)* terms *$\mathcal{T}(\mathcal{F},\mathcal{X})$ is the smallest set containing $\mathcal{X}$ and such that $f(t_1,\ldots,t_n)$ is in $\mathcal{T}(\mathcal{F},\mathcal{X})$ whenever $arity(f) = n$ and $t_i \in \mathcal{T}(\mathcal{F},\mathcal{X})$ for $i \in [1..n]$. Let the function symbol $\Box \notin \mathcal{F}$ with arity zero, the set $\mathcal{T}_\Box(\mathcal{F},\mathcal{X})$ of "contexts", denoted simply by $\mathcal{T}_\Box$, is made with terms with symbols in $\mathcal{F}\cup\mathcal{X}\cup\{\Box\}$ which includes exactly one occurence of $\Box$. Evidently, $\mathcal{T}_\Box(\mathcal{F},\mathcal{X})$ and $\mathcal{T}(\mathcal{F},\mathcal{X})$ are two disjoint sets.*

We denote by $\mathcal{V}ar(t)$ the set of variables occurring in t. The set of variable-free terms, called ground *terms, is denoted $\mathcal{T}(\mathcal{F})$. Terms that contain variables are said* open.

Notice that the set of $\mathcal{V}ar(t)$ variables of a term t as well as the notion of ground term depends on the set of variables the terms are defined on. For example, $\texttt{Reg}(\Omega, d)$ is open if $\mathcal{X} = \{\Omega\}$ and $\mathcal{F} = \{\texttt{Reg}, d\}$. It is closed if $\mathcal{X} = \emptyset$ and $\mathcal{F} = \{\Omega, \texttt{Reg}, d\}$.

Example 2 *Let $\mathcal{X} = \emptyset$ and $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{F}_2$ where $\mathcal{F}_0 = \{x, \Omega\}$, $\mathcal{F}_1 = f$ and $\mathcal{F}_3 = \texttt{Integral}$. Then, $\texttt{Integral}(\Omega, f(x), x)$ is a term in in $\mathcal{T}(\mathcal{F})$ and thus in $\mathcal{T}(\mathcal{F},\mathcal{X})$. It corresponds to the mathematical expression $\int_\Omega f(x)\,dx$. Notice that both x and Ω are function symbols of arity zero, i.e. they are constants in the rewriting sense while x is a variable in the mathematical sense.*

To make clear the distinction between the various types of variables, the mathematical variables will be denoted by the letters $x, y, z, \ldots$ however the rewriting variables will be denoted by the capital letters $X, Y, Z, \ldots$.

Definition 3 *Let t be a term in $\mathcal{T}(\mathcal{F},\mathcal{X})$.*

1. *The set of positions of the term t, denoted by $\mathcal{P}os(t)$, is a set of strings[1] of positive integers such that:*
   - *If $t = X \in \mathcal{X}$, then $\mathcal{P}os(t) = \{\epsilon\}$, where ϵ denotes the empty string.*

[1] A string is an element of $\mathbb{N}^\omega = \{\epsilon\} \cup \mathbb{N} \cup (\mathbb{N}\times\mathbb{N}) \cup (\mathbb{N}\times\mathbb{N}\times\mathbb{N}) \cup \cdots$. Given two strings $p = p_1p_2\ldots p_n$ and $q = q_1q_2\ldots q_m$, the *concatenation* of p and q, denoted by $p \cdot q$ or simply pq, is the string $p_1p_2\ldots p_nq_1q_2\ldots q_m$. Notice that $(\mathbb{N}^\omega, \cdot)$ is a monoid with ϵ as the identity element.

- *If* $t = f(t_1, ..., t_n)$ *then*

$$\mathcal{P}os(t) = \{\epsilon\} \cup \bigcup_{i=1}^{n} \{ip \mid p \in \mathcal{P}os(t_i)\}.$$

We denote the set of the positions of a subterm r *in a term* t *by* $\mathcal{P}os(t, r)$. *The position* ϵ *is called the root position of term* t*, and the function or variable symbol at this position is called root symbol of* t.

2. *The prefix order defined as*

$$p \leq q \text{ iff there exists } p' \text{ such that } pp' = q \tag{1}$$

is a partial order on positions. If $p' \neq \epsilon$ *then we obtain the strict order* $p < q$. *We write* $(p \parallel q)$ *iff* p *and* q *are incomparable with respect to* $\leq$. *The binary relations defined by*

$$p \sqsubset q \quad \text{iff} \quad \left(p < q \text{ or } p \parallel q\right) \tag{2}$$

$$p \sqsubseteq q \quad \text{iff} \quad \left(p \leq q \text{ or } p \parallel q\right) \tag{3}$$

are total relations on positions.

3. *For any* $p \in \mathcal{P}os(t)$ *we denote by* $t|_p$ *the subterm of* t *at position* p:

$$\begin{aligned} t|_\epsilon &= t, \\ f(t_1, ..., t_n)|_{iq} &= t_i|_q. \end{aligned}$$

4. *For any* $p \in \mathcal{P}os(t)$ *we denote by* $t[s]_p$ *the term obtained by replacing the subterm of* t *at position* p *by* s:

$$\begin{aligned} t[s]_\epsilon &= s, \\ f(t_1, ..., t_n)[s]|_{iq} &= f\left(t_1, ..., t_i[s]_q, ..., t_n\right) \end{aligned}$$

We sometimes use the notation $t[s]_p$ *just to emphasizes that the term* t *contains* s *as subterm at position* p.

5. *For any* $u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ *and* $\tau \in \mathcal{T}_\Box(\mathcal{F}, \mathcal{X}) \cup \mathcal{T}(\mathcal{F}, \mathcal{X})$, *the notation* $\tau[u]$ *has two different meanings depending on* τ,

$$\begin{aligned} \tau[u] &= \tau[u]_{\mathcal{P}os(t,\Box)} \text{ for } \tau \in \mathcal{T}_\Box(\mathcal{F}, \mathcal{X}) \\ &= \tau \text{ for } \tau \in \mathcal{T}(\mathcal{F}, \mathcal{X}). \end{aligned}$$

Definition 4 (Substitution) *A substitution is a mapping* $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ *such that* $\sigma(X) \neq X$ *for only finitely many* X*s. The (finite) set of variables that* σ *does not map to themselves is called the domain of* σ:

$$\mathcal{D}om(\sigma) \stackrel{def}{=} \{X \in \mathcal{X} \mid \sigma(X) \neq X\}.$$

If $\mathcal{D}om(\sigma) = \{X_1, ..., X_n\}$ *then we write* σ *as:*

$$\sigma = \{X_1 \mapsto \sigma(X_1), ..., X_n \mapsto \sigma(X_n)\}.$$

The range of σ *is* $\mathcal{R}an(\sigma) := \{\sigma(X) \mid X \in \mathcal{D}om(\sigma)\}$

A substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X}))$ *uniquely extends to an endomorphism* $\widehat{\sigma} : \mathcal{T}(\mathcal{F}, \mathcal{X}) \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ *defined by:*

1. $\widehat{\sigma}(X) = \sigma(X)$ *for all* $X \in \mathsf{Dom}(\sigma)$*,*

2. $\widehat{\sigma}(X) = X$ *for all* $X \notin \mathsf{Dom}(\sigma)$*,*

3. $\widehat{\sigma}(f(t_1, \ldots, t_n)) = f(\widehat{\sigma}(t_1), \ldots, \widehat{\sigma}(t_n))$ *for* $f \in \mathcal{F}$*.*

In what follows we do not distinguish between a substitution and its extension.

The composition $\sigma\gamma$ *of two substitutions* σ *and* γ *is defined by* $\sigma\gamma\,(X) \stackrel{def}{=} \sigma\,(\gamma\,(X))$*, for all* $X \in \mathsf{Dom}(\gamma)$*.*

Definition 5 *A term* u *is subsumed by a term* t *if there is a substitution* σ *s.t.* $\sigma(t) = u$*. A substitution* σ *is subsumed by a substitution* γ*, where* $\mathsf{Dom}(\sigma) = \mathsf{Dom}(\gamma)$*, iff for every variable* $X \in \mathsf{Dom}(\sigma)$*, the term* $\sigma(X)$ *is subsumed by the term* $\gamma(X)$*.*

Definition 6 (Rewrite rule) *A* $\mathcal{T}(\mathcal{F}, \mathcal{X})$ rewrite rule *over a signature* $\mathcal{F}$ *is a a pair* $(l, r) \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}, \mathcal{X})$*, denoted by* $l \twoheadrightarrow r$ *or* $l \rightrightarrows r$*, such that* $\mathcal{V}ar\,(r) \subseteq \mathcal{V}ar\,(l)$*. Usually,* $l \neq X$ *with* $X \in \mathcal{X}$*.* l *is called the left-hand side (lhs) of the rewrite rule and* r *the right-hand side (rhs). A term rewriting system (TRS) is a set of rewrite rules.*

We simply say rewrite rule when $\mathcal{X}$ and $\mathcal{F}$ are clear from the context.

Definition 7 (Term rewriting) *We say that* $u \in \mathcal{T}\,(\mathcal{F})$ rewrites into *a term* $v \in \mathcal{T}\,(\mathcal{F})$ *w.r.t. a rewrite rule* $l \twoheadrightarrow r$*, which is also denoted* $u \longrightarrow v$*, iff there exist (i) a position* $p \in \mathcal{P}os(u)$ *and (ii) a ground substitution* σ *with* $\mathsf{Dom}(\sigma) = \mathcal{V}ar\,(l)$ *such that* $u_{|p} = \sigma(l)$ *and* $v = u\,[\sigma(r)]_p$*. We can use the notation* $u \stackrel{l\twoheadrightarrow r,\sigma,p}{\longrightarrow} v$ *to make explicit the corresponding rewrite rule, position and substitution respectively.*

We say that $u \in \mathcal{T}\,(\mathcal{F})$ rewrites into *a term* $v \in \mathcal{T}\,(\mathcal{F})$ *w.r.t. a rewrite system* $\mathcal{R}$*, which is also denoted* $u \longrightarrow_{\mathcal{R}} v$*, iff there exist (i) a position* p*, (ii) a ground substitution* σ*, and (iii) a rewrite rule* $l \twoheadrightarrow r \in \mathcal{R}$ *such that* $u \stackrel{l\twoheadrightarrow r,\sigma,p}{\longrightarrow} v$*.*

$\longrightarrow^*_{\mathcal{R}}$ *denotes the reflexive transitive closure of the relation* $\longrightarrow_{\mathcal{R}}$*.*

Example 8

$$\sin(X)^2 \to 1 - \cos(X)^2, \text{ and } 1 - \cos(X)^2 \to \sin(X)^2,$$

where sin, cos, 1*, and '*$-$*'* $\in \mathcal{F}$ *and* $X \in \mathcal{X}$*.*

Definition 9 (Unification problem, unifier, complete and minimal set of unifiers) *Let* t_i, u_i *for* $i = 1, ..., n$ *be sorted terms.*

- *A unification problem is a finite set of potential equations* $E = \{t_1 \doteq u_1, \ldots, t_n \doteq u_n\}$*.*
- *A unifier of* E *is a substitution* σ *which is a solution of* E*, i.e. of* $\sigma(t_i) = \sigma(u_i)$ *and* $\mathrm{sort}(t_i) = \mathrm{sort}(u_i)$ *for all* i*. If* E *admits a solution, then it is called solvable.*
- *For a given unification problem* E*, a (possibly infinite) set* $\{\sigma_1, \sigma_2, \ldots\}$ *of unifiers of* E *is complete iff each solution of* E *is subsumed by some unifiers* σ_i*. The set of unifiers is minimal if none of its substitutions subsumes another one.*

The existence of a complete and minimal solution of a unification problem is ensured by the following proposition.

Proposition 10 (See [BN99]) *Each solvable unification problem E has a complete and minimal singleton solution set $\{\sigma\}$. The solution σ is called the most general unifier of E, and it is denoted by* $\mathrm{mgu}(E)$*.*

For the sake of completness of the presentation, we recall an algorithm of unification.

Algorithm 11 (Unification)

$$
\begin{aligned}
E \cup \{t \doteq t\} &\rightsquigarrow E \\
E \cup \{f(t_1, \ldots, t_n) \doteq f(u_1, \ldots, u_n)\} &\rightsquigarrow E \cup \{t_1 \doteq u_1, \ldots, t_n \doteq u_n\} \\
E \cup \{f(t_1, \ldots, t_n) \doteq g(u_1, \ldots, u_m)\} &\rightsquigarrow \mathtt{fail} \quad \textit{if } g \neq f \\
E \cup \{f(t_1, \ldots, t_n) \doteq X\} &\rightsquigarrow E \cup \{X \doteq f(t_1, \ldots, t_n)\} \\
E \cup \{X \doteq t\} &\rightsquigarrow E[X := t] \cup \{X \doteq t\} \\
&\quad\ \textit{if } X \notin \mathcal{V}ar(t) \textit{ and } X \in \mathcal{V}ar(E) \textit{ and} \\
E \cup \{X \doteq f(X_1, \ldots, X_n)\} &\rightsquigarrow \mathtt{fail} \\
&\quad\ \textit{if } X \in \mathcal{V}ar(f(X_1, \ldots, X_n)) \textit{ or}
\end{aligned}
$$

2.1.1 A Rewriting Strategy Language

We define the syntax of our strategy language as well as its semantics.

Definition 12 (strategies) *The syntax of a (first order) strategy s over terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is given by*

$$s \ ::= l \to r \ \mid \ \eta(s) \ \mid \ s; s \ \mid \ s \oplus s \ \mid \ s^\star \ \mid \ \mathtt{Some}(s) \ \mid$$

where l, r, u, w, m, v are terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$.

The strategy (i.e. the rewriting rule) $l \to r$ consists in the application of $l \to r$ to the top. While applied to a given term t, we compute a substitution σ such that $t = \sigma(l)$. Either there is no such substitution, and in this case the application of $l \to r$ to t fails. Or, there exists a unique substitution σ, and in this case the result of the application of $l \to r$ to t is the term $\sigma(r)$.

The strategy constructor η stands for the *identity as fail*. That is, for a given strategy s, when the strategy $\eta(s)$ is applied to an input term t, then the strategy s is applied to s and if this application fails, then the final result is t. That is, in case of failure, η behaves like the identity strategy.

The strategy constructor ; stands for the *composition* of two strategies. That is, the strategy $s_1; s_2$ consists of the application of s_1 followed by the application of s_2.

The strategy constructor $\oplus$ stands for *left choice*. The strategy $s_1 \oplus s_2$ applied s_1. If this application fails then s_2 is applied. Hence, $s_1 \oplus s_2$ fails when applied to a given term t iff both s_1 and s_2 fail when applied to t. Notice that the constructor $\oplus$ is associative.

The strategy constructor * stands for the *iteration*. The strategy $s^\star$ consists in the iteration of the application of s until a fixed-point is reached.

The strategy $\mathtt{Some}(s)$ applied s to all the immediate subterms of the input term, it fails iff s fails on all the subterms. This strategy will be used to build more complex traversal strategies.

The semantics of a strategy is a mapping from $\mathcal{T}(\mathcal{F},\mathcal{X})\cup\mathbb{F}\to\mathcal{T}(\mathcal{F},\mathcal{X})\cup\mathbb{F}$, where $\mathbb{F}\notin\mathcal{F}\cup\mathcal{X}$ is a particular constant that denotes the failure of the application of a strategy to a term.

$$(l\to r)(t)\overset{def}{=}\begin{cases}\sigma(r) & \text{if } \exists\sigma \text{ s.t. } \sigma(l)=t,\\ \mathbb{F} & \text{otherwise.}\end{cases}$$

$$(\eta(s))(t)\overset{def}{=}\begin{cases}s(t) & \text{if } s(t)\neq\mathbb{F}\\ t & \text{otherwise.}\end{cases}$$

$$(s_1;s_2)(t)\overset{def}{=}\begin{cases}s_2(s_1(t)) & \text{if } s_1(t)\neq\mathbb{F}\\ \mathbb{F} & \text{otherwise.}\end{cases}$$

$$(s_1\oplus s_2)(t)\overset{def}{=}\begin{cases}s_1(t) & \text{if } s_1(t)\neq\mathbb{F}\\ s_2(t) & \text{otherwise.}\end{cases}$$

$$\begin{aligned}s^\star(t)&\overset{def}{=}s^n(t) \text{ where } n \text{ is the least integer s.t. } s^{n+1}(t)=s^n(t)\\ &\quad\text{and } s^n(t)=\underbrace{s(s(\dots s(t)))}_{n \text{ times}}\end{aligned}$$

$$(\texttt{Some}(s))(t)\overset{def}{=}\begin{cases}\mathbb{F} & \text{if } t=f(t_1,\dots,t_n) \text{ and } \forall i\in[n]\ s(t_i)=\mathbb{F}\\ f(\eta(s)(t_1),\dots,\eta(s)(t_n)) & \text{if } t=f(t_1,\dots,t_n) \text{ and } \exists i\in[n]\ s(t_i)\neq\mathbb{F}\\ \mathbb{F} & \text{if } ar(t)=0\end{cases}$$

Besides, we need some *traversal* strategies. They explore the structure of the term they are applied on. We provide next two traversal strategies: `InnerMost` and `BottomUp`.

The strategy `InnerMost`(s) is very common in symbolic computation. It applies the strategy s once to all the inner most redexes of t for s, i.e. to the largest number subterms of t that are far from the root and on which s succeeds. In other words the strategy `InnerMost` traverses the term t up from its root and tries to apply s to each traversed subterm once. If the strategy s succeeds on some subterm t' of t, then it is not applied to t. As a consequence, `InnerMost`(s) fails if and only if s fails on all the subterms of t.

The strategy `BottomUp`(s) tries to apply the strategy s to all the subterms of any term t, at any depth, by starting from the leaves of t and going up to the root of t.

The strategies `InnerMost` and `BottomUp` are defined as follows:

$$\texttt{InnerMost}(s)\overset{def}{=}\texttt{Some}(\texttt{InnerMost}(s))\ \oplus\ s$$

$$\texttt{BottomUp}(s)\overset{def}{=}\texttt{Some}(\texttt{BottomUp}(s))\ ;s$$

2.1.2 The processing language

We describe the grammar of expressions in which the problem is processed leaving out other structures as proofs, lemmas, propositions, etc.

$$
\begin{aligned}
\mathcal{E} &::= \texttt{Plus}(\mathcal{E},\mathcal{E}) \;\mid\; \texttt{Mult}(\mathcal{E},\mathcal{E}) \;\mid \\
&\quad\;\; \texttt{Minus}(\mathcal{E}) \;\mid\; \texttt{Inverse}(\mathcal{E}) \;\mid\; \texttt{Power}(\mathcal{E},\mathcal{E}) \;\mid\; \mathcal{F} \\
\mathcal{F} &::= \texttt{Fun}(f,[\mathcal{I};\ldots;\mathcal{I}],[\mathcal{V};\ldots;\mathcal{V}],[(\mathcal{R},\mathcal{E});\ldots;(\mathcal{R},\mathcal{E})],K) \;\mid \\
&\quad\;\; \texttt{Oper}(A,[\mathcal{I};\ldots;\mathcal{I}],[\mathcal{E};\ldots;\mathcal{E}],[\mathcal{V};\ldots;\mathcal{V}],[\mathcal{V};\ldots;\mathcal{V}] \\
&\qquad\qquad [d;\ldots;d]) \;\mid \\
&\quad\;\; \mathcal{V} \;\mid\; \texttt{MathCst}(d) \;\mid\; \texttt{Nil} \\
\mathcal{V} &::= \texttt{MathVar}(x,[\mathcal{I};\ldots;\mathcal{I}],\mathcal{R}) \\
\mathcal{R} &::= \texttt{Reg}(\Omega,[\mathcal{I};\ldots;\mathcal{I}],[d,\ldots,d],[\mathcal{R};\ldots;\mathcal{R}],\mathcal{R},\mathcal{E}) \;\mid\; \texttt{Nil}, \\
\mathcal{I} &::= \texttt{Ind}(i,[d,\ldots,d])
\end{aligned}
$$

It describes mathematical expressions built up by the arithmetic operations "+" (`Plus`), "·" (`Prod`), etc as well as the mathematical function constructor `Fun` and the operator constructor `Oper`. The latter allows one to build expressions for mathematical operators such as the integration operator $\int$, the derivative operator ∂, the summation operator $\sum$, the multi-scale operators T, B, etc. Besides, a mathematical expression can contain mathematical variables (`MathVar`), regions (`Reg`) and discrete variables (`Ind`).

We shall sometimes write and depict lists in the prefix notation using the constructor `list` and `nil` (empty list). For instance, if e_1 and e_2 are two expressions, we shall write $\texttt{list}(e_1, \texttt{list}(e_2, \texttt{nil}))$ instead of $[e_1; e_2]$. The symbol `Nil` in the grammar above represents an "empty expression".

The user language is based on shortcuts avoiding the repetitions. Examples of the short-cut terms are given bellow.

$$
\begin{array}{llll}
\underline{\Omega} & \equiv \texttt{Reg}(\Omega, d), & \underline{u_{\underline{i}}}(\underline{x}) & \equiv \texttt{Fun}(u, [\underline{i}], (\underline{u}(\underline{x}), \underline{i}), \\
\underline{x} & \equiv \texttt{Var}(x, [], \underline{\Omega}), & u_{\underline{i}\underline{j}}(\underline{x}) & \equiv \texttt{Fun}(u, [\underline{i}, \underline{j}], (\underline{u}(\underline{x}), \underline{i}), \\
\underline{y} & \equiv \texttt{Var}(y, [], \underline{\Omega}), & \frac{\partial \underline{u}(\underline{x})}{\partial \underline{x}} & \equiv \texttt{Oper}(\texttt{Deriv}, \underline{u}(\underline{x}), [\underline{x}]), \\
\underline{i} & \equiv \texttt{Index}(i, \texttt{Set}(I, \{1:n\})), & \frac{\partial u_{\underline{i}}(\underline{x})}{\partial \underline{x}_{\underline{j}}} & \equiv \texttt{Oper}(\texttt{Deriv}, \underline{u}(\underline{x}), [\underline{x}]), \\
\underline{j} & \equiv \texttt{Index}(i, \texttt{Set}(I, \{1:n\})), & \int \underline{u}(\underline{x})\ d\underline{x} & \equiv \texttt{Oper}(\texttt{Integral}, \underline{u}(\underline{x}), [\underline{x}]), \\
\underline{u}(\underline{x}) & \equiv \texttt{Fun}(u, [], [\underline{x}], \texttt{unknown}), & \int \underline{u}(\underline{x}, \underline{y})\ d\underline{x} & \equiv \texttt{Oper}(\texttt{Integral}, \underline{u}(\underline{x}, \underline{y}), [\underline{x}]), \\
\underline{u}(\underline{x}, \underline{y}) & \equiv \texttt{Fun}(u, [], [\underline{x}, \underline{y}], \texttt{unknown}), & \sum_{\underline{i}} \underline{u_{\underline{i}}}(\underline{x}) & \equiv \texttt{Oper}\left(\texttt{Sum}, \underline{u_{\underline{i}}}(\underline{x}), [\underline{i}]\right), \\
\underline{v}(\underline{x}) & \equiv \texttt{Fun}(v[], [], [\underline{x}], \texttt{test}), & \sum_{\underline{i}} \underline{u_{\underline{i}\underline{j}}}(\underline{x}) & \equiv \texttt{Oper}\left(\texttt{Sum}, \underline{u_{\underline{i}\underline{j}}}(\underline{x}), [\underline{i}]\right).
\end{array}
$$

3 Principle of the Extension-Combination Method

In this section we introduce the ideas behind the notion of the extension-combination method.

The idea of the extension can be viewed as a generalization of a proof. It is based on two concepts: *mathematical equivalence* and *parametrisation*. Consider the expression $\partial_x v$ that we want to generalize to $\partial_{x_i} v$ where $i \in \{1, \ldots, n\}$. We proceed in two steps. First a mathematical equivalence consists in introduction a discrete variable i, ranging from 1 to 1, to the expression $\partial_x v$, yielding the expression $\partial_{x_i} v$, where $i \in \{1, \ldots, 1\}$. Notice that this transformation does not change the mathematical meaning. Secondly, the step of parametrisation consists in replacing the upper bound 1 by a variable n, yielding the expression $\partial_{x_i} v$, where $i \in \{1, \ldots, n\}$. We propose

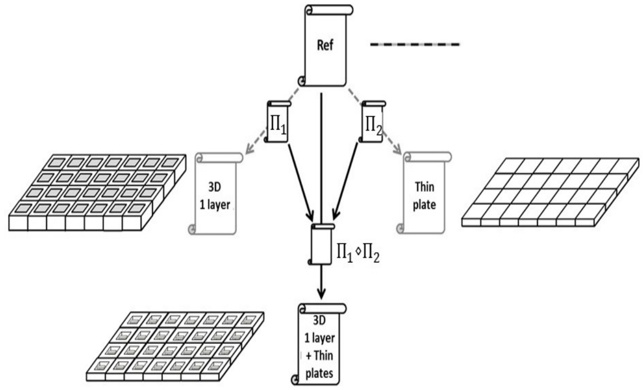

Figure 3: By-extension-combination principle illustrated on a reference proof (top). Left: a one-layer periodic problem. Right: a thin layer with homogeneous coefficients. The combination of these two extensions yields a thin layer with periodic coefficients (bottom).

next an implementation of the notion of generalization by the by-extension-combination method, where for the moment we do not distinguish between the two phases of *mathematical equivalence* and *parametrisation*. This method relies on three key principles. Firstly, we introduce a reference model together with its derivation. This derivation is called the reference proof, it is depicted on the top of Figure 3. It is based on the derivation approach of [LS07] and was implemented and presented in details in [YBL14b]. Although the reference model covers a very simple case, its proof is expressed in a sufficiently general way. A number of basic algebraic *properties* are formulated as *rewriting rules*, they are considered as the building blocks of the proofs. The full derivation of the model is formulated as a sequence of applications of these rules. The proof of some properties is also performed by a sequence of applications of mathematical rules when the others are admitted e.g. the Green rule.

Then, an *elementary extension* is obtained by an application of an elementary transformation, called also an *extension operator*, to the reference proof. In Figure 3 the extension operators are Π_1 and Π_2. They respectively cover the extension to the 3-D setting and the thinness setting. We notice that, in practice, when a single feature is taken into account, only a small change occurs in a relatively long proof. In other words, while considering an elementary extension, most of the existing rules could be reused by operating a small change on them, and, on the other hand, only a small number of new rules has to be manually introduced.

Finally, we make possible the combination of two extension operators to produce a new extension operator that takes into account the features covered by each initial extension operator. In the example of Figure 3, the combination of the extension operators Π_1 and Π_2 is the extension operator $\Pi_1 \diamond \Pi_2$. By iterating this process, many extension operators can be combined together giving rise to complex extensions that cover many features.

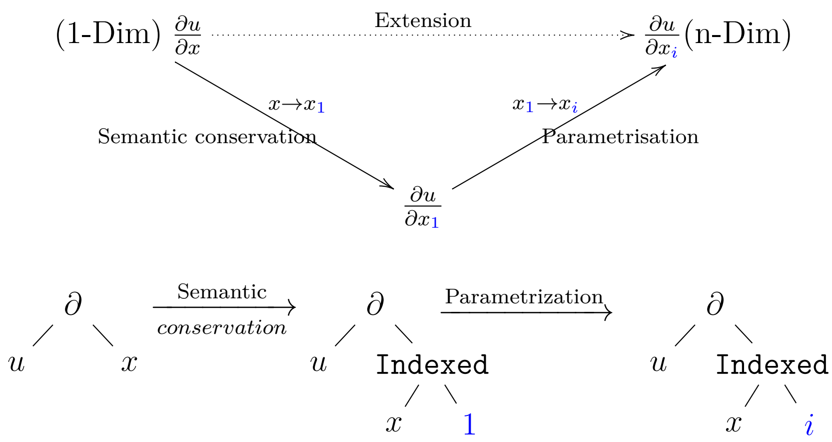

Figure 5: Schematic description of the notion of extension

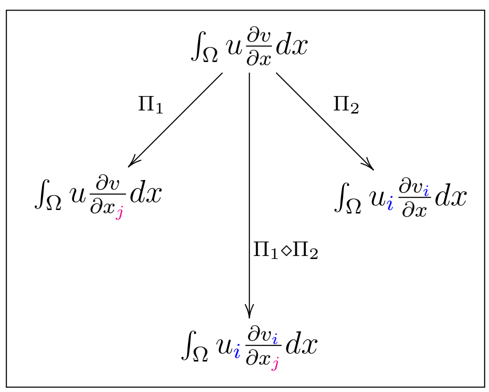

Figure 4: An example of the by-extension-combination method applied to the mathematical expression $\int_\Omega u \frac{\partial v}{\partial x} dx$ that corresponds to the left hand side of Green formula Eq. (74), where Π_1 stands for the extension operator of the *multi-dimension setting*, Π_2 stands for the extension operator to the *vector-valued setting*, and $\Pi_1 \diamond \Pi_2$ stands for the combination of Π_1 and Π_2.

Figure 4 shows how the extension operators and their combination operate on the mathematical expression $\int_\Omega u \frac{\partial v}{\partial x} dx$, which is the left hand side of Green formula Eq. (74).

3.1 Extensions as Second Order Strategies

The mathematical equivalence between FO-strategies is defined through an equational system $\mathcal{R}$ made with a set of SO-rules and a list of position where to apply it. In the example, $\mathcal{R}$ is made with the single SO-rule $R := x \rightrightarrows \sum_{i=1}^{1} x_i$ for the equivalence between two different expressions of a variable. In general, we say that two FO-strategies s_1 and s_2 are mathematically equivalent with respect to $\mathcal{R}$, written $s_1 \simeq_{\mathcal{R}} s_2$, if they are syntactically equal[2] modulo application of $\mathcal{R}$ at a set of

[2]The syntactic equality between rewriting rules has always to be done modulo α-equivalence. Two rewriting rules are α-equivalent if they are syntactically identical up to a renaming of their variables. For instance, the rules $f(x) \to g(x)$ and $f(y) \to g(y)$, where x and y are variables, are α-equivalent. Two strategies are α-equivalent if they are syntactically identical up to a renaming of the variables of their rewriting rules. For instance, the strategies `BottomUp`$(f(x) \to g(x))$ and `BottomUp`$(f(y) \to g(y))$ are α-equivalent.

positions and that a SO-strategy S conserves this mathematical equivalence if for all FO-strategy s, $S(s) \simeq_{\mathcal{R}} s$.

Next, a SO-strategy S' is *parametrized* if the right-hand side part of some of its rewriting rules contains FO-variables which are not in its left-hand side part as for instance the rule $S' := \sum_{i=1}^{1} \rightrightarrows \sum_{i=1}^{n}$ where n is a FO-variable and therefore a parameter of S'. The idea behind parametrization is to transform FO-strategies with additional FO-variables, so that they represent more general properties. The set of these FO-variables is the set of parameters of S.

Definition 13 (Strategy of extension) *A SO extension strategy $\mathcal{S}$ with respect to an equational system $\mathcal{R}$ of mathematical equivalences is a combination $S; S'$ where S is a SO-strategy conserving the mathematical equivalences and S' a parametrized SO-strategy. When two FO-strategies s_1, s_2 satisfy*

$$s_1 = \mathcal{S}(s_0)$$

we say that s_1 generalize s_0 through $\mathcal{S}$.

In the above example, the SO-rule $\mathcal{R}$ consists in replacing x by $\texttt{Oper}\,(\texttt{Sum}, x, [\underline{i}]))$ and inserting $\tau_1 := 1 : 1$ in the empty list of $\underline{x}$. The replacement is seen as another insertion of the term $\tau_2 := \texttt{Oper}\,(\texttt{Sum}, \perp, [\underline{i}]))$ between the root and $\underline{x}$, the latter being positionned in place of $\perp$. The SO-strategy S consists in the insertion of the terms τ_1 and τ_2 at all positions of $[]$ in $\underline{x}$ and of $\underline{x}$ respectively. The operation of adding terms at some positions is the key operation for extensions and is defined rigorously in the next subsection.

3.2 Position-based extensions and their combination

In sections 4 and 5 we show how to implement the extension operators. In fact there are two equivalent implementations of the extension operators. The first one is in terms of adding contexts at a certain positions of the input term. Despite the fact that this implementation is not practical, it captures the idea of an extension. This implementation will be presented in section 4.

The second implementation consists in implementing an extension operator as a rewriting strategy that searches for patterns then adds contexts at certain positions. This implementation will be presented in section 5. The second implementation is clearly more general than the first but it is equivalent to the first in the sense that, for every input term and a strategy extension operator, we can construct an equivalent position-based extension operator. We shall argue that the operation of combination of strategy-based extensions is sound and complete by relating it to the operation of combination of position-based extensions.

Beside, in both cases we define the operation $\Diamond$ of combination of extension operators in such way, in each case, the class of extension operators is closed under combination. In other words, we define a class of rewriting strategies that comes with an internal combination operation.

4 Position-based extensions and their combination

An extension operator consist in the operation that adds *contexts* or replace terms by rewriting variables for parametrization at given positions of a term. For the sake of shortness, we do not take term replacement into account in the rest of the paper. The context $\tau = \texttt{list}(\square, j)$ depicted in Figure 6 captures the idea that the extension would add a discrete variable to an expression. The application of $\Pi_{(p,\tau)}$ to the term $t = \partial_x v(x)$ at the position of p of the variable x (the parameter of the derivative operator ∂) yields the term $\Pi_{(p,\tau)}(t) = \partial_{x_j} v(x)$. Similarly, Figure 7 illustrates the extension operator $\Pi_{(q,\tau')}$ and its application to the term $t = \partial_x v(x)$ at the position of the function v which yields the term $\Pi_{(q,\tau')}(t) = \partial_x v_i(x)$.

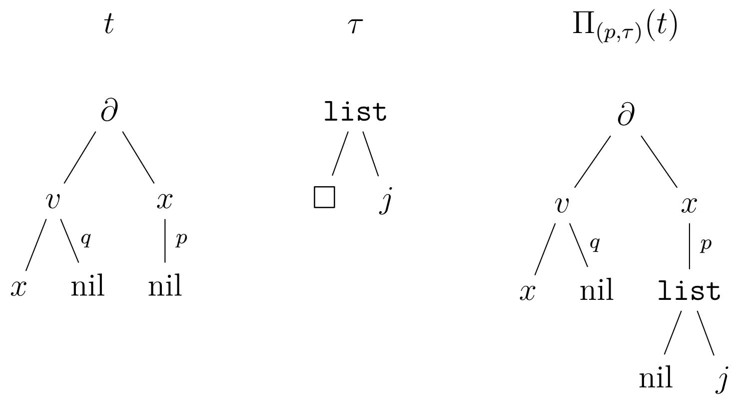

Figure 6: Application of the extension operator $\Pi_{(p,\tau)}$ (with the extension constructor τ) to the term $t = \partial_x v(x)$ at the position p, yielding the term $\partial_{x_j} v(x)$.

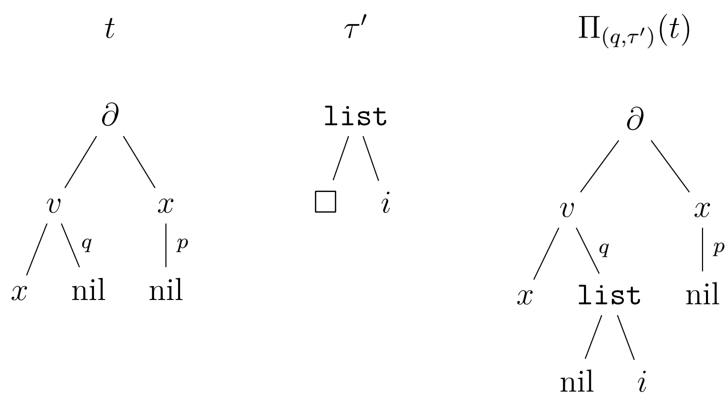

Figure 7: Application of the extension operator $\Pi_{(q,\tau')}$ (with the extension constructor τ') to the term $t = \partial_x v(x)$ at the position q, yielding the term $\partial_x v_i(x)$.

When an extension operator $\Pi_{(p,\tau)}$, where p is a position, is applied to a term t at the position p, the context τ is inserted at the position p of t, and the subterm of t at the position p is inserted at $\Box$. The general schema of the application of an extension operator is depicted in Figure 8.

Figure 9 shows the combination of the two extension operators $\Pi_{(p,\tau)}$ and $\Pi_{(q,\tau')}$. In what follows $\mathcal{P}os$ stands for the set of positions, $t_{|p}$ stands for the subterm of t at the position p, and $t[t']_p$ stands for the replacement of the subterm of t at the position p with the term t'. If τ is a context, we shall denote by $\tau[t]$ the term obtained from τ by replacing the $\Box$ by t. In particular, if τ and τ' are contexts, we call $\tau[\tau']$ the composition of τ and τ'.

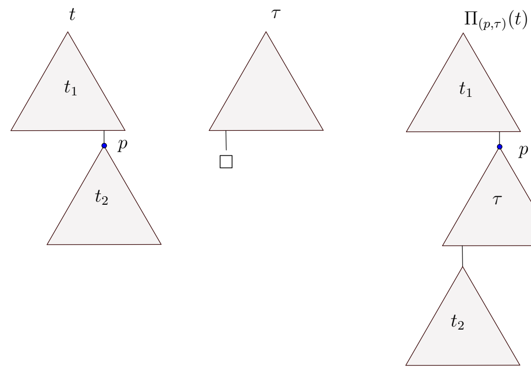

Figure 8: Schematic diagram of the application of an extension operator $\Pi_{(p,\tau)}$ (with a context τ) to a term t at the positon p.

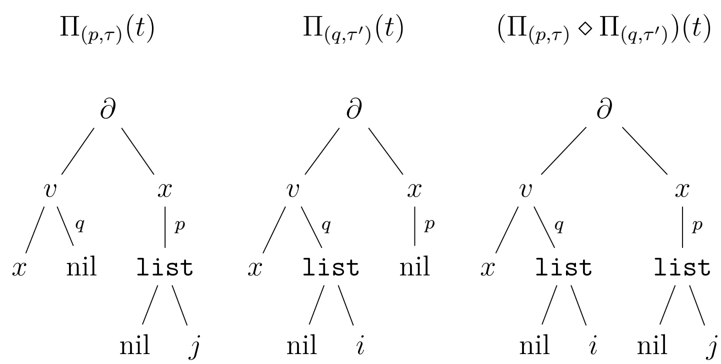

Figure 9: The extension operator $\Pi_{(q,\tau)} \diamond \Pi_{(q,\tau')}$ which is the combination of the two extension operators $\Pi_{(p,\tau)}$ and $\Pi_{(q,\tau')}$, and its application to the term t.

Formally,

Definition 14 (Position-based extension) *A parameter of a position-based extension operator is of the form*

$$\mathcal{P} = [(p_1, \tau_1), \ldots, (p_n, \tau_n)],$$

where p_i are positions and each τ_i is either a context in $\mathcal{T}_\Box$ or a term in $\mathcal{T}$. The empty list parameter will be denoted by $\varnothing$.

Moreover, we impose that the parameters satisfy some constraints to avoid conflicts in the simultaneous operations of insertions and replacements.

Definition 15 (Well-founded position-based extension) *The parameter $\mathcal{P} = [(p_1, \tau_1), ..., (p_n, \tau_n)]$ of a position-based extension is well founded iff*

i.) a position occurs at most one time in $\mathcal{P}$, i.e. $p_i \neq p_j$ for all $i \neq j$.

ii.) if p_i is a position of term insertion (i.e. (p_i, τ_i) is such that $\tau_i \in \mathcal{T}_\Box$) and if p_j is a position of term replacement (i.e. (p_j, τ_j) is such that $\tau_j \in \mathcal{T}$), we have that $p_i \sqsubset p_j$,

iii.) any two positions p_i and p_j of replacement are not comparable, i.e. $p \parallel p'$.

In all what follows we assume that the parameters are well-founded. Let $\Theta(\mathcal{P})$ to be the set of positions at the root of $\mathcal{P}$ if $\mathcal{P}$ is viewed as a tree. For a parameter $\mathcal{P}$, we next define its semantics denoted by $\Pi_{\mathcal{P}}$ which is called an extension operator.

Definition 16 (Position-based extension operator) *The extension operator $\Pi_{\mathcal{P}}$, where $\mathcal{P}$ is a parameter, is inductively defined by:*

$$\begin{cases} \Pi_{\varnothing} & \stackrel{def}{=} \mathbb{F} \\ \Pi_{(p,\tau)} & \stackrel{def}{=} \lambda u.u \mapsto u[\tau[u_{|p}]]_p \\ \Pi_{[(p_1,\tau_1),\ldots,(p_n,\tau_n)]} & \stackrel{def}{=} \Pi_{(p_1,\tau_1)};\ldots;\Pi_{(p_n,\tau_n)} \end{cases}$$

In what follows the combination of two extension operators $\Pi_{\mathcal{P}}$ and $\Pi_{\mathcal{P}'}$, where $\mathcal{P},\mathcal{P}'$ are position-based extension, will be denoted by $\Diamond$, amounts to combine their parameters $\mathcal{P}$ and $\mathcal{P}'$:

$$\Pi_{\mathcal{P}} \Diamond \Pi_{\mathcal{P}} \stackrel{def}{=} \Pi_{\mathcal{P}\Diamond\mathcal{P}'}$$

and thus we shall only define the combination of the parameters of extensions.

Definition 17 (Combination of two position-based extensions) *Let $\mathcal{P} = [(p_1,\tau_1),\ldots,(p_n,\tau_n)]$ and $\mathcal{P}' = [(p'_1,\tau'_1),\ldots,(p'_m,\tau'_m)]$. Let $(p''_1,\tau''_1),\ldots,(p''_r,\tau''_r)$ be such that, for all i, either*

i.) $p''_i = p_j$, for some j, and in this case $\tau''_i = \tau_j$, or

ii.) $p''_i = p'_j$ and $\tau''_i = \tau'_j$, or

iii.) $p''_i = p_j = p'_k$, for some j,k, and in this case

$$\tau''_i = \begin{cases} \tau'_k[\tau_j] & \text{if } \tau'_k \in \mathcal{T}_\square \\ \tau_j & \text{if } \in \tau'_k, \tau_j \in \mathcal{T} \text{ and } \tau'_k = \tau_j \end{cases} \tag{4}$$

We define the combination $\mathcal{P} \diamond \mathcal{P}'$ by

$$\mathcal{P} \diamond \mathcal{P}' = \begin{cases} [(p''_1,\tau''_1),\ldots,(p''_r,\tau''_r)], & \text{if } [(p''_1,\tau''_1),\ldots,(p''_r,\tau''_r)] \text{ is Well-founded,} \\ \varnothing & \text{otherwise} \end{cases}$$

Remark 18 *The following hold.*

1. *The combination of position-based extensions is non commutative since $\tau'_j[\tau_i] \neq \tau_i[\tau'_j]$ and is non associative due to the condition $\tau_i = \tau'_j$, e.g.*

$$\begin{aligned} (\Pi_{(p,f(\square))} \Diamond \Pi_{(p,f(\square))}) \Diamond \Pi_{(p,f(f(\square)))} &= \Pi_{(p,f(f(\square)))} \Diamond \Pi_{(p,f(f(\square)))} = \Pi_{(p,f(f(\square)))} \\ \Pi_{(p,f(\square))} \Diamond (\Pi_{(p,f(\square))} \Diamond \Pi_{(p,f(f(\square)))}) &= \Pi_{(p,f(\square))} \Diamond \Pi_{(p,f(f(f(\square))))} = \Pi_{(p,f(f(f(f(\square)))))}. \end{aligned}$$

2. *The neutral element of the combination operation is $\Pi_{(\varepsilon,\square)}$. i.e. For every extension parameter $\mathcal{P}$, we have that*

$$\Pi_{(\varepsilon,\square)} \Diamond \Pi_{\mathcal{P}} = \Pi_{\mathcal{P}} \Diamond \Pi_{(\varepsilon,\square)} = \Pi_{\mathcal{P}}.$$

3. *The extension operator $\Pi_{(\varepsilon,\square)}$ plays the role of the identity. i.e. for every term $t \in \mathcal{T}$, we have that*

$$\Pi_{(\varepsilon,\square)}(t) = t.$$

5 Strategy-based extensions and their combination

The definitions of position-based extensions and of their combination are satisfactory from the theoritical point of view. However, they are not useful for practical applications, since the positions are generally not accessible and cannot be used on a regular basis for operations. So, these principles are translated into a framework of classical strategies to form a subclass of extensions that is closed by combination. Extensions are built starting from three kinds of simple strategies of navigation and then using inductive definitions yields *strategy-based* extensions, or simply, extensions. The two formulations are in fact equivalent. We extend the definition of the combination operator $\Diamond$ from position-based extensions to strategy-based extensions. The subsection 5.1 completes Section 2 by introducing the key concept of anti-patterns.

5.1 Positive and negative patterns

In order to carry on the combination of extensions we need to consider negative patterns. For instance, the negative patter $\neg a$, where a is a constant, represents all the closed terms which are different than a. The reason of the consideration of negative patterns follows naturally when one wants to combine two strategies, say s_1 and s_2:

$$s_1 = (u_1 \to u_1) \,;\, s_1'$$
$$s_2 = (u_2 \to u_2) \,;\, s_2'$$

where u_1, u_2 are (positive) patterns and s_1', s_2' are strategies. The wanted combination consists of three elements. *(i.)* either u_1 and u_2 can be unified, yielding the pattern denoted by $u_1 \wedge u_2$, and in this case combine s_1' and s_2'. *(ii.)* either we have u_1 but not u_2 and is this case we consider s_1', *(iii.)* either we have u_2 but not u_1 and is this case we consider s_2'. Formally, the resulting strategy can be written as

$$\begin{aligned} s_1 \Diamond s_2 = & (u_1 \wedge u_2 \to u_1 \wedge u_2) \quad ; s_1' \Diamond s_2' \ \oplus \quad (5)\\ & (u_1 \wedge \neg u_2 \to u_1 \wedge \neg u_2) \,;\, s_1' \quad \oplus \quad (6)\\ & (u_2 \wedge \neg u_1 \to u_2 \wedge \neg u_1) \,;\, s_2' \quad \oplus \quad (7)\end{aligned}$$

Definition 19 (Positive and negative patterns) *A pattern is defined by the following grammar:*

$$\mathcal{U} ::= x \ \mid \ f(\mathcal{U}, \ldots, \mathcal{U}) \ \mid \ \neg\mathcal{U} \ \mid \ \mathcal{U} \wedge \mathcal{U} \ \mid \ \mathcal{U} \vee \mathcal{U} \quad (8)$$

where $x \in \mathcal{X}$ is a variable and f is a functional symbol from $\mathcal{F}$. The set of patterns is denoted by $\mathcal{T}(\mathcal{X}, \mathcal{F})$ or simply by $\mathcal{T}$. A positive pattern (resp. negative pattern) is a pattern that does not contains (resp. that contains) the symbol $\neg$. The set of positive patterns (resp. negative patterns) is denoted by $\mathcal{T}^+$ or simply $\mathcal{T}^+(\mathcal{X}, \mathcal{F})$ (resp. $\mathcal{T}^-$ or simply $\mathcal{T}^-(\mathcal{X}, \mathcal{F})$).

If u is a positive pattern in $\mathcal{T}^+$ and u' is a pattern in $\mathcal{T}$ and p is a position in u, we shall denote by $u[u']_{||p}$ the pattern $u[u' \wedge u_{|p}]_{|p}$, i.e. we "add" u' at the position p of u, or more precisely, we insert the conjunction $u_{|p} \wedge u'$ at the position p of u.

The semantics of a pattern is given by its *unfolding*. The unfolding of a pattern t, denoted by $[\![t]\!]$, is the set of all terms that can be obtained from t by instantiating the variables:

Definition 20 (Unfolding of a pattern) *The unfolding of a pattern is a function* $[\![\cdot]\!] : \mathcal{T} \rightarrow 2^{\mathcal{T}^+}$ *that associates to each pattern a (possibly empty) set of positive patterns, it is inductively defined by*

$$[\![u]\!] \stackrel{def}{=} \bigcup_{\sigma} \{\sigma(u)\} \ \textit{if } u \in \mathcal{X} \cup \mathcal{F}^0 \tag{9}$$

$$[\![f(t_1, \ldots, t_n)]\!] \stackrel{def}{=} \bigcup \{f(t'_1, \ldots, t'_n) \ | \ t'_i \in [\![t_i]\!]\} \ \textit{if } f \in \mathcal{F}^n \tag{10}$$

$$[\![f(t_1, \ldots, t_n)]\!] \stackrel{def}{=} \emptyset \ \textit{if } \exists i \in [1, n] \ \textit{s.t.} \ [\![t_i]\!] = \emptyset, \ \textit{where } f \in \mathcal{F}^n \tag{11}$$

$$[\![u \wedge v]\!] \stackrel{def}{=} [\![u]\!] \cap [\![v]\!] \tag{12}$$

$$[\![u \vee v]\!] \stackrel{def}{=} [\![u]\!] \cup [\![v]\!] \tag{13}$$

$$[\![\neg u]\!] \stackrel{def}{=} \mathcal{T}^+ \setminus [\![u]\!] \tag{14}$$

Remark 21 *Notice that*

1. *If* x *is a variable in* $\mathcal{X}$ *and* f *is a functional symbol, then* $[\![x]\!] = \mathcal{T}^+$ *and* $[\![\neg x]\!] = \emptyset$ *and* $[\![f(\neg x)]\!] = \emptyset$.

2. $[\![t]\!] = \emptyset$ *iff* t *contains* $\neg x$ *where* x *is a variable.*

3. $[\![\neg f(a)]\!] \neq [\![f(\neg a)]\!]$.

4. *If* $t \in \mathcal{T}^+$ *then* $|\ [\![t]\!]\ | \geq 1$.

5. $|\ [\![t]\!]\ | = 1$ *iff* $t \in \mathcal{T}^+$ *and* $Var(t) = \emptyset$.

6. *If* $t \in \mathcal{T}^-$ *and* t *does not contain double negations (i.e. subterms of the form* $\neg\neg u$*), then either* $[\![t]\!] = \emptyset$ *or* $|\ [\![t]\!]\ | = \infty$.

7. *There is a linear time algorithm that checks whether* $[\![u]\!] = \emptyset$*, for any pattern* u.

8. *If the set of constants and functional symbols is finite* ($|\mathcal{F}| \leq \infty$)*, then a negative pattern in* $\mathcal{T}^-$ *can be equivalent to a positive disjunction of patterns from* $\mathcal{T}^+$ *e.g.*

$$[\![\neg a]\!] = [\![\bigvee_{\substack{g_i \in \mathcal{F}^n \\ x_i^j \in \mathcal{X}}} g_i(x_i^1, \ldots, x_i^n) \vee \bigvee_{b_i \in \mathcal{F}^0 \setminus \{a\}} b_i]\!] \tag{15}$$

$$[\![\neg f(a)]\!] = [\![\bigvee_{\substack{g_i \in \mathcal{F}^n \setminus \{f\} \\ x_i^j \in \mathcal{X}}} g_i(x_i^1, \ldots, x_i^n) \vee \bigvee_{b_i \in \mathcal{F}^0 \setminus \{a\}} f(b_i) \vee \bigvee_{b_i \in \mathcal{F}^0} b_i]\!] \tag{16}$$

9.

$$[\![f(u_1, \ldots, u_n) \wedge f(v_1, \ldots, v_n)]\!] = [\![f(u_1 \wedge v_1, \ldots, u_n \wedge v_n)]\!] \tag{17}$$

The DeMorgan laws can be proved:

Lemma 22 *For every patterns $u, v \in \mathcal{T}$, we have that*

$$[\![\neg\neg u]\!] = [\![u]\!] \tag{18}$$
$$[\![\neg(u \wedge v)]\!] = [\![\neg u \vee \neg v]\!] \tag{19}$$
$$[\![\neg(u \vee v)]\!] = [\![\neg u \wedge \neg v]\!] \tag{20}$$

Proof. Immediate from the De Morgan laws in set theory since $\neg$ is interpreted as the complement. ■

Lemma 23 *For any positive patterns $u, u' \in \mathcal{T}^+$ we have that*

$$[\![u \wedge u']\!] = [\![\delta(u)]\!], \text{ where } \delta = \mathtt{mgu}(u, u') \tag{21}$$

Proof. On the one hand we have that

$$\begin{aligned}
[\![u \wedge u']\!] &\stackrel{def}{=} [\![u]\!] \cap [\![u']\!] && (22)\\
&\stackrel{def}{=} \{\sigma(u) \ \mid \ \sigma \in \xi\} \cap \{\sigma'(u') \ \mid \ \sigma' \in \xi\} && (23)\\
&= \bigcup_{\sigma,\sigma'} \{t \in \mathcal{T}^+ \ \mid \ t = \sigma(u) = \sigma'(u')\} && (24)\\
&= \bigcup_{\sigma} \{t \in \mathcal{T}^+ \ \mid \ t = \sigma(u) = \sigma(u')\} && (25)
\end{aligned}$$

On the other hand, we have that

$$[\![\delta(u)]\!] \stackrel{def}{=} \bigcup_{\lambda \in \xi} \{\lambda(\delta(u)) \ \mid \ \delta = \mathtt{mgu}(u, u')\} \tag{26}$$

The inclusion (25) $\subseteq$ (26) follows from the fact that δ is the most general unifier of u and u' and hence, if $\sigma(u) = \sigma(u')$, then σ is subsumed by δ (See Definition 5) in the sense that exists a substitution λ such that $\lambda \circ \delta = \sigma$, i.e. $\lambda(\delta(u)) = \sigma(u)$.

■

5.2 Extension operators as strategies

Instead of considering only positions, we can enrich the definition of the extension operators to incorporate both positions and nested searching patterns. The grammar of the parameters of (strategy-based) extension operators follows.

Definition 24

$$\textit{(I)} \quad \begin{cases} \mathcal{P} & ::= (\theta, \tau) \ \mid \ (\theta, \mathcal{P}) \ \mid \ [\mathcal{P}, \ldots, \mathcal{P}] \ \mid \mathtt{IM}(\mathcal{P}) \\ \theta & ::= p \ \mid \ u \end{cases}$$

where p is a position, τ is a context in $\mathcal{T}_\Box$ or a positive term in $\mathcal{T}^+$, u is a pattern in $\mathcal{T}$, and IM *is an unary constructor.*

The semantics of $\mathcal{P}$, denoted by $\Pi_{\mathcal{P}}$, is formally given in definition 26. The semantics of the position based extensions, i.e. extension whose parameter is of the form $[(p_1, \tau_1), \ldots, (p_n, \tau_n)]$ where p_i are positions, was already given in section 4. Intuitively, the semantics of $(u, \mathcal{P})$, is a strategy that checks if the pattern u matches with the input term, if it is the case we apply $\Pi_{\mathcal{P}}$ to the input term, otherwise, it fails. The semantics of $\mathtt{IM}(u, \mathcal{P})$ is a strategy that locates all the subterms of the input term that match with u by the `InnerMost` strategy, and at each subterms we apply the strategy $\Pi_{\mathcal{P}}$. The semantics of $[(u_1, \mathcal{P}_1), \ldots, (u_n, \mathcal{P}_n)]$ is the left choice between the strategies $\Pi_{(u_1,\mathcal{P}_1)}, \ldots, \Pi_{(u_n,\mathcal{P}_n)}$.

We generalize next the condition of well-foundedness to strategy-based extensions.

Definition 25 (Well-founded (strategy-based) extensions) *A an extension is well-founded iff*

i.) *all its list subparameters are either of the form*

(a) $[(p_1, \mathcal{P}_1), \ldots, (p_n, \mathcal{P}_n)]$, *where all* p_i *are positions,*

(b) *or* $[(u_1, \mathcal{P}_1), \ldots, (u_n, \mathcal{P}_n)]$, *where all* u_i *are patterns,*

(c) *or* $[\mathtt{IM}(u_1, \mathcal{P}_1), \ldots, \mathtt{IM}(u_n, \mathcal{P}_n)]$, *where all* u_i *are patterns.*

ii.) *all its subparameters of the form* $[(p_1, \tau_1), \ldots, (p_n, \tau_n)]$, *are well-founded according to definition 15, where* p_i *are positions and* τ_i *are terms in* $\mathcal{T}^+ \cup \mathcal{T}_{\Box}$. *And,*

iii.) *for all its subparameters of the form* $[(u_1, \mathcal{P}_1), \ldots, (u_n, \mathcal{P}_n)]$, *where each* u_i *is a pattern in* $\mathcal{T}$ *and* $\mathcal{P}_i$ *is either an extension or a term in* $\mathcal{T}^+ \cup \mathcal{T}_{\Box}$, *we have that* u_i *is of the form* u'_i *or* $u'_i \wedge U'_i$ *where* $u'_i \in \mathcal{T}^+$ *and* $U'_i \in \mathcal{T}^-$. *And,*

iv.) *for all its subparameters of the form* $[(u_1 \wedge U_1, \mathcal{P}_1), \ldots, (u_n \wedge U_n, \mathcal{P}_n)]$, *where each* u_i *is a positive pattern in* $\mathcal{T}^+$, u_i *is a negative pattern in* $\mathcal{T}^-$ *and* $\mathcal{P}_i$ *is either an extension or a term in* $\mathcal{T}^+ \cup \mathcal{T}_{\Box}$, *we have that for every* $i, j = 1, \ldots, n$ *with* $i \neq j$,

$$\forall p \in \mathcal{P}os(u_i), \quad [\![u_i[u_j \wedge U_j]_{||p} \wedge U_i]\!] = \emptyset.$$

In what follows we assume that the extensions are well-founded. The semantics of the an extension as a strategy follows.

Definition 26 *The semantics of an extension* $\mathcal{P}$, *denoted by* $\Pi_{\mathcal{P}}$, *is defined inductively on* $\mathcal{P}$ *as* *follows.*

$$\Pi_{(\theta,\tau)} \stackrel{def}{=} \begin{cases} \lambda t.t \mapsto t[\tau[t_{|\theta}]]_\theta & \textit{if } \theta \in \mathcal{P}os \\ u \to \tau[u] & \textit{if } \theta = u \in \mathcal{T} \end{cases}$$

$$\Pi_{(\theta,\mathcal{P})} \stackrel{def}{=} \begin{cases} \lambda t.t \mapsto t[t']_\theta & \textit{if } \theta \in \mathcal{P}os \\ \quad \textit{where } t' = \Pi_{\mathscr{P}}\left(t_{|\theta}\right) & \\ (u \to u\ ); \Pi_{\mathscr{P}} & \textit{if } \theta = u \in \mathcal{T} \end{cases}$$

$$\Pi_{[\mathscr{P}_1,\ldots,\mathscr{P}_n]} \stackrel{def}{=} \begin{cases} \Pi_{\mathscr{P}_1}; \ldots; \Pi_{\mathscr{P}_n} & \textit{if } \Theta(\mathscr{P}_n) \neq \emptyset \\ \Pi_{\mathscr{P}_1} \oplus \ldots \oplus \Pi_{\mathscr{P}_n} & \textit{otherwise} \end{cases}$$

$$\Pi_{\mathtt{IM}(\mathscr{P})} \stackrel{def}{=} \mathtt{InnerMost}(\Pi_{\mathscr{P}})$$

We shall write $\mathcal{P} \equiv \mathcal{Q}$ iff $\Pi_{\mathcal{P}} = \Pi_{\mathcal{Q}}$.

The following properties are immediate from the definition of the semantics of extension operators. For any positions p, p', any patterns $\mathcal{U}, \mathcal{U}'$ and any parameter $\mathcal{P}$, we have that

$$\begin{aligned} (\epsilon, \mathcal{P}) &\equiv \mathcal{P}, \\ (p, (p', \mathcal{P})) &\equiv (pp', \mathcal{P}), \\ (\mathcal{U}, (\mathcal{U}', \mathcal{P})) &\equiv (\mathcal{U} \wedge \mathcal{U}', \mathcal{P}), \\ \mathrm{IM}(\mathcal{U}, (\mathcal{U}', \mathcal{P})) &\equiv \mathrm{IM}(\mathcal{U} \wedge \mathcal{U}', \mathcal{P}) \end{aligned}$$

5.2.1 Combination of strategy-based extension

In this section we define the operation of combination of extensions. Before that we give the definition of the depth of an extension, which corresponds its longest path starting from the root if it is viewed as a tree; and the definition of its width.

Definition 27 (Depth of an extension) *The depth of an extension operator $\Pi_{\mathcal{P}}$, denoted by $\Delta(\Pi_{\mathcal{P}})$ is the depth of tree-like structure of $\mathcal{P}$, denoted simply by $\Delta(\mathcal{P})$, that is,*

$$\Delta(\mathcal{P}) = \begin{cases} 0 & \textit{if } \mathcal{P} = (\theta, \tau) \\ 1 + \Delta(\mathcal{P}') & \textit{if } \mathcal{P} = (\theta, \mathcal{P}') \\ 1 + max(\Delta(\mathcal{P}_1), \ldots, \Delta(\mathcal{P}_n)) & \textit{if } \mathcal{P} = (\theta, [\mathcal{P}_1, \ldots, \mathcal{P}_n]) \end{cases}$$

Definition 28 (Width of an extension) *The width of an extension $\Pi_{\mathcal{P}}$, denoted by $|\Pi_{\mathcal{P}}|$, or simply by $|\mathcal{P}|$, is defined by*

$$|\mathcal{P}| = \begin{cases} 1 & \textit{if } \mathcal{P} = (\theta, \tau) \textit{ or } \mathcal{P} = (\theta, \mathcal{P}') \textit{ or } \mathcal{P} = \mathtt{IM}(\mathcal{P}'),\ \textit{where } \theta \in \mathcal{T} \cup \mathcal{P}os, \tau \in \mathcal{T}_\square \cup \mathcal{T}^+ \\ n & \textit{if } \mathcal{P} = [\mathcal{P}_1, \ldots, \mathcal{P}_n] \end{cases}$$

Definition 29 (Combination of extensions) *The combination of two extension, $\Pi_{\mathcal{P}}$ and $\Pi_{\mathcal{P}'}$ as follows.*

$$\Pi_{\mathcal{P}} \Diamond \Pi_{\mathcal{P}'} = \Pi_{\mathcal{P} \Diamond \mathcal{P}'} \tag{27}$$

where the combination $\mathcal{P} \Diamond \mathcal{P}'$ of parameters is inductively defined by:

$\Delta(\mathcal{P}) = \Delta(\mathcal{P}') = 0$ **and** $|\mathcal{P}| = |\mathcal{P}'| = 1$**.** *We disinguihs three cases depending on the type of $\mathcal{P}$ and $\mathcal{P}'$.*

Case (i). *If the symbol at the root of $\mathcal{P}$ and $\mathcal{P}'$ is a position, i.e. $\Theta(\mathcal{P}) \neq \emptyset$ and $\Theta(\mathcal{P}') \neq \emptyset$, i.e. $\mathcal{P} = (p, \tau)$ and $\mathcal{P}' = (p', \tau')$ then this case is similar to the combination given in Definition 17 for position-based extensions.*

Case (ii). *If the symbol at the root of $\mathcal{P}$ and $\mathcal{P}'$ is a pattern, i.e. $\mathcal{P} = (U, \tau)$ and $\mathcal{P}' = (U', \tau')$, then*

$$\begin{aligned}
(U,\tau)\Diamond(U',\tau') \stackrel{def}{=} \Big[\, & (U \wedge U', \tau'[\tau])\,, && (28)\\
& (U \wedge \neg U', \tau)\,, && (29)\\
& (\neg U \wedge U', \tau') \Big] && (30)\\
& && (31)
\end{aligned}$$

Case (iii). *If the symbol at the root of $\mathcal{P}$ and $\mathcal{P}'$ is* `IM`*, i.e. $\mathcal{P} = IM(u \wedge U, \tau)$ and $\mathcal{P}' = \mathtt{IM}(u' \wedge U', \tau')$, then*

$$\mathcal{P} \Diamond \mathcal{P}' \stackrel{def}{=} \tag{32}$$

$$\begin{cases}
\mathtt{IM}\Big[\Big(u\,[u' \wedge U']_{|p} \wedge U, [(\epsilon,\tau),(p,\tau')]\Big)\,, & \\
\quad \Big(u\,[\neg\,(u' \wedge U')]_{|p} \wedge U, [(\epsilon,\tau)]\Big)\,, & \text{if } \exists! p \text{ s.t. } [\![u\,[u' \wedge U']_{|p} \wedge U]\!] \neq \emptyset \\
\quad (\neg(u \wedge U), \mathtt{IM}\,(u' \wedge U', \tau'))\Big] & \text{and } u_{|p} \notin \mathcal{X} \\
\mathtt{IM}\Big(u\,[u' \wedge U']_{|p} \wedge U, [(\epsilon,\tau),(p,\mathtt{IM}(u' \wedge U', \tau'))]\Big) & \text{if } \forall p [\![u\,[u' \wedge U']_{|p} \wedge U]\!] \neq \emptyset \\
 & \text{and } u_{|p} \in \mathcal{X} \\
\text{"Symmetric cases"} & \\
\mathtt{IM}\,([\mathcal{P},\mathcal{P}']) & \text{if } \exists p \text{ s.t. } [\![u\,[u' \wedge U']_{|p} \wedge U]\!] \neq \emptyset \\
 & \text{and } \exists q \text{ s.t. } [\![u\,[u' \wedge U']_{|q} \wedge U]\!] \neq \emptyset \\
\varnothing & \text{otherwise}
\end{cases} \tag{33}$$

$\Delta(\mathcal{P}) = \Delta(\mathcal{P}') \geq 1$ **and** $|\mathcal{P}| = |\mathcal{P}'| = 1$**.** *Again, we disinguihs three cases depending on the symbol at the root of $\mathcal{P}$ and $\mathcal{P}'$.*

Case (i). *If the symbol at the root of $\mathcal{P}$ and $\mathcal{P}'$ is a position, i.e. $\mathcal{P} = (p, \mathcal{Q})$ and $\mathcal{P} =$*

$(p', \mathcal{Q}')$, *then*

$$(p, \mathcal{Q}) \lozenge (p', \mathcal{Q}') \overset{def}{=} \begin{cases} [(p, \mathcal{Q}), (p', \mathcal{Q}')] & \text{if } p' < p \\ [(p', \mathcal{Q}'), (p, \mathcal{Q})] & \text{if } p < p' \\ (p, \mathcal{Q} \lozenge \mathcal{Q}') & \text{if } p = p' \end{cases} \tag{34}$$

Case (ii). *If the symbol at the root of $\mathcal{P}$ and $\mathcal{P}'$ is a pattern, i.e.* $\mathcal{P} = (U, \mathcal{Q})$ *and* $\mathcal{P}' = (U', \mathcal{Q}')$*, then*

$$(U, \mathcal{Q}) \lozenge (U', \mathcal{Q}') \overset{def}{=} \Big[(U \wedge U', \mathcal{Q} \lozenge \mathcal{Q}'), \tag{35}$$

$$(U \wedge \neg U', \mathcal{Q}), \tag{36}$$

$$(\neg U \wedge U', \mathcal{Q}') \Big] \tag{37}$$

$$\tag{38}$$

Case (iii). *If the symbol at the root of $\mathcal{P}$ and $\mathcal{P}'$ is* IM*, i.e.*

$$\begin{cases} \mathcal{P} = \mathtt{IM}(u \wedge U, \mathcal{Q}), \text{ and} \\ \mathcal{P}' = \mathtt{IM}(u' \wedge U', \mathcal{Q}'), \end{cases}$$

then

$$\mathcal{P} \lozenge \mathcal{P}' \overset{def}{=} \tag{39}$$

$$\begin{cases} \mathtt{IM}\Big[\Big(u\left[u' \wedge U'\right]_{|p} \wedge U, (\epsilon, \mathcal{Q}) \lozenge (p, \mathcal{Q}')\Big), & \\ \quad \Big(u\left[\neg\left(u' \wedge U'\right)\right]_{|p} \wedge U, \mathcal{Q}\Big), & \text{if } \exists! p \text{ s.t. } [\![u\left[u' \wedge U'\right]_{|p} \wedge U]\!] \neq \emptyset \\ \quad (\neg(u \wedge U), \mathtt{IM}\left(u' \wedge U', \mathcal{Q}'\right)) \Big] & \text{and } u_{|p} \notin \mathcal{X} \\ \mathtt{IM}\Big(u\left[u' \wedge U'\right]_{|p} \wedge U, (\epsilon, \mathcal{Q}) \lozenge (p, \mathtt{IM}(u' \wedge U', \mathcal{Q}'))\Big) & \text{if } \exists! p \text{ s.t. } [\![u\left[u' \wedge U'\right]_{|p} \wedge U]\!] \neq \emptyset \\ & \text{and } u_{|p} \in \mathcal{X} \\ \text{"Symmetric cases"} & \\ \mathtt{IM}\left([\mathcal{P}, \mathcal{P}']\right) & \text{if } \forall p [\![u\left[u' \wedge U'\right]_{|p} \wedge U]\!] \neq \emptyset \\ & \text{and } \forall q [\![u\left[u' \wedge U'\right]_{|q} \wedge U]\!] \neq \emptyset \\ \varnothing & \text{otherwise} \end{cases} \tag{40}$$

$\Delta(\mathcal{P}) = \Delta(\mathcal{P}') \geq 0$ and ($|\mathcal{P}| \geq 2$ or $|\mathcal{P}'| \geq 2$). *The definition is by induction on* $|\mathcal{P}'|$.

$$[\mathcal{Q}_1, \ldots, \mathcal{Q}_n] \otimes \mathcal{Q} = \begin{cases} [\mathcal{Q}_1, \ldots, \mathcal{Q}_i \lozenge \mathcal{Q}, \ldots, \mathcal{Q}_n] & \text{if } \exists i \text{ s.t. } Q_i \lozenge Q \neq \varnothing \\ [\mathcal{Q}_1, \ldots, \mathcal{Q} \lozenge \mathcal{Q}_i, \ldots, \mathcal{Q}_n] & \text{if } \exists i \text{ s.t. } Q \lozenge Q_i \neq \varnothing \\ [\mathcal{Q}_1, \ldots, \mathcal{Q}_n, \mathcal{Q}] & \text{otherwise} \end{cases} \tag{41}$$

and,

$$\mathcal{P}\lozenge([\mathcal{Q}'_1,\ldots,\mathcal{Q}'_n]) = (\mathcal{P}\lozenge\mathcal{Q}'_1)\lozenge[\mathcal{Q}'_2,\ldots,\mathcal{Q}'_n] \tag{42}$$

Remark 30 *The following is not hard to prove.*

- *If* $\Pi_{\mathcal{P}}$ *and* $\Pi_{\mathcal{Q}}$ *are well-founded, so is* $\Pi_{\mathcal{P}}\lozenge\Pi_{\mathcal{Q}}$*.*
- *The neutral element of the combination operation is* $\Pi_{(\varepsilon,\square)}$*. i.e. for every extension parameter* $\mathcal{P}$*, we have that*

$$\Pi_{(\varepsilon,\square)}\lozenge\Pi_{\mathcal{P}} = \Pi_{\mathcal{P}}\lozenge\Pi_{(\varepsilon,\square)} = \Pi_{\mathcal{P}}.$$

- *For every extension parameter* $\mathcal{P}$*, we have that*

$$\Pi_{\mathcal{P}}\lozenge\Pi_{\mathcal{P}} = \Pi_{\mathcal{P}}$$

5.3 A correction criterion of the combination of strategy-based extension operators

The expressions of the combinations of strategy based extensions have been given without justification. Since a same extension can be expressed in different ways, it is mandatory that the combination of two extensions, whatever their forms, yields equivalent extensions. The combination of position-based extensions are defined in a clear and nonambiguous manner, so we choose it as the reference. The first lemma of this section shows that any strategy-based extension can be expressed as a position-based extension. Then, we introduce a correction criteria of the combination rules of strategy-based combinations, its application to the above rules being left for a future work.

Lemma 31 (Strategy-based extensions as position-based extensions) *Then there exists a function* $\Psi : \zeta \times \mathcal{T}^+ \to \zeta^\star$ *that associates to each extension operator* $\mathcal{P}$ *in* ζ *and positive pattern* t *in* $\mathcal{T}^+$ *a pattern-free extension operator* $\mathcal{Q}$ *in* $\zeta^\star$*, denoted by* $\Psi_{(\mathcal{P},t)}$*, such that*

$$\Pi_{\mathcal{P}}(t) = \Pi_{\mathcal{Q}}(t) \tag{43}$$

Proof. Out of $\mathcal{P}$ and t, we shall construct a pattern-free extension operator $\mathcal{Q}$ of the form

$$\mathcal{Q} = [(p_1,\tau_1),\ldots,(p_n,\tau_n)] \tag{44}$$

where p_i are positions in t and τ_i are contexts with $n \geq 0$, such that $\Pi_{\mathcal{P}}(t) = \Pi_{\mathcal{Q}}(t)$. The proof is by induction $\Delta(\mathcal{P})$, the depth of $\mathcal{P}$.

Basic case: $\Delta(\mathcal{P}) = 0$**.** We distinguish three cases depending on the type of $\mathcal{P}$.

Case (i). $\eta(\mathcal{P}) = Posi$. In this case we define

$$\mathcal{Q} \overset{def}{=} \mathcal{P} \tag{45}$$

Case (ii). $\eta(\mathcal{P}) = Patt$. In this case $\mathcal{P}$ is of the form

$$\mathcal{P} = [(u_1 \wedge U_1,\tau_1),\ldots,(u_n \wedge U_n,\tau_n)] \tag{46}$$

where u_i (resp. U_i) are positive (resp. negative) patterns and τ_i are contexts. Thus, according to definition **??** of the semantics of extension operators, we have that

$$\Pi_{\mathcal{P}} = \bigoplus_i (u_i \to \tau_i[u_i] \quad \text{if } U_i) \tag{47}$$

And since there is a unique $k \in [1, n]$ such that the conditional rule $u_k \to \tau_k[u_k] \quad \text{if } U_k$ can be applied to t, then

$$\begin{aligned} \Pi_{\mathcal{P}}(t) &= (u_k \to \tau_k[u_k] \quad \text{if } U_k)\,(t) &(48)\\ &= \tau_k[\sigma(u_k)] \quad \text{where } \sigma(u_k) = t \text{ and } \sigma \in [\![U_k]\!] &(49)\\ &= \tau_k[t] &(50)\end{aligned}$$

On the other hand, we define $\mathcal{Q}$ as follows:

$$\mathcal{Q} \stackrel{def}{=} [(\epsilon, \tau_k)] \tag{51}$$

Therefore,

$$\begin{aligned} \Pi_{\mathcal{Q}}(t) &= \tau_k[t] &(52)\\ &= \Pi_{\mathcal{P}}(t) &(53)\end{aligned}$$

Case (iii). $\eta(\mathcal{P}) = \mathtt{IM}$. In this case $\mathcal{P}$ is of the form

$$\mathcal{P} = \mathtt{IM}[(U_1, \tau_1), \ldots, (U_n, \tau_n)] \tag{54}$$

where $[\![U_i]\!] \cap [\![U_j]\!] = \emptyset$ for all $i \neq j$. Let

$$\begin{cases} [p_1^i, \ldots, p_{n_i}^i] = \lambda^\star(U_i), & \text{and} \\ \phi(p_1^i) = \tau_i \end{cases} \tag{55}$$

for all $i \in [n]$. And let

$$[q_1, \ldots, q_m] = \text{Max} \bigcup_{i \in [n]} \{\lambda^*(U_i)\} \tag{56}$$

We define $\mathcal{Q}$ by

$$\mathcal{Q} = [(q_1, \tau'_1), \ldots, (q_m, \tau'_m)] \tag{57}$$

where $\tau'_j = \phi(q_j)$, for all $j \in [m]$.

Finally, the proof for $\Delta(\mathcal{P}) \geq 1$ is simply made by induction which does not involve any additional difficulty.

■

Criterion 32 (Correctness of combinations of strategy-based extensions) *The combination of two extension operators $\mathcal{P}$ and $\mathcal{P}'$ is correct iff*

$$\Psi(\mathcal{P} \Diamond \mathcal{P}', t) = \Psi(\mathcal{P}, t) \Diamond \Psi(\mathcal{P}', t) \tag{58}$$

for any positive term $t \in \mathcal{T}^+$.

6 Mathematical proofs

This section presents the reference proof, already published in [YBL14a], following the technique of [LS07] and applied to a second order elliptic equation posed in a one-dimensional domain. It is made with propositions, that are accepted without proof, and of Lemmas which are proven. Then, an extension of the proof to a multi-dimensional domain is written in the spirit of strategy-based extensions to serve as an illustration of the concept of extension.

6.1 The Reference Proof

This section presents the model derivation used as a reference for the extensions. We recall the framework of the two-scale convergence as presented in [LS07]. The presentation is divided into three subsections. The first one is devoted to basic definitions and properties, stated as *Propositions*. The latter are admitted without proof because they are assumed to be prerequisites, or building blocks, in the proofs. They are used as elementary steps in the two other sections detailing the proof of the convergence of the two-scale transform of a derivative, and the homogenized model derivation. The main statements of these two subsections are also stated as *Propositions* and their proofs are split into numbered blocks called lemmas. Each lemma is decomposed into steps referring to the definitions and propositions. All components of the *reference model* derivation, namely the definitions, the propositions, the lemmas and the proof steps are designed so that to be easily implemented but also to be generalized for more complex models. We quote that a number of trivial mathematical properties are used in the proofs but are not explicitly stated nor cited. However an implementation must take them into account.

6.1.1 Notations, Definitions and Propositions

Note that the functional framework used in this section is not as precise as it should be for a usual mathematical work. The reason is that the complete functional analysis is not covered by our symbolic computation. So fine and precise mathematical statements and justifications cannot be in the focus of this work.

In the sequel, $A \subset \mathbb{R}^n$ is a bounded open set, with measure $|A|$, having a "sufficiently" regular boundary ∂A and with unit outward normal denoted by $n_{\partial A}$. We shall use the set $L^1(A)$ of integrable functions and the set $L^p(A)$, for any $p > 0$, of functions f such that $f^p \in L^1(A)$, with norm $||v||_{L^p(A)} = (\int_A |v|^p \, dx)^{1/p}$. The Sobolev space $H^1(A)$ is the set of functions $f \in L^2(A)$ which gradient $\nabla f \in L^2(A)^n$. The set of p times differentiable functions on A is denoted by $\mathcal{C}^p(A)$, where p can be any integer or ∞. Its subset $\mathcal{C}^p_0(A)$ is composed of functions which partial derivatives are vanishing on the boundary ∂A of A until the order p. For any integers p and q, $\mathcal{C}^q(A) \subset L^p(A)$. When $A = (0, a_1) \times ... \times (0, a_n)$ is a cuboid (or rectangular parallelepiped) we say that a function v defined in $\mathbb{R}^n$ is A-periodic if for any $\ell \in \mathbb{Z}^n$, $v(y + \sum_{i=1}^n \ell_i a_i e_i) = v(y)$ where e_i is the i^{th} vector of the canonical basis of $\mathbb{R}^n$. The set of A-periodic functions which are $\mathcal{C}^\infty$ is denoted by $\mathcal{C}^\infty_\sharp(A)$ and those which are in $H^1(A)$ is denoted by $H^1_\sharp(A)$. The operator tr (we say *trace*) can be defined as the restriction operator from functions defined on the closure of A to functions defined on its boundary ∂A. Finally, we say that a sequence $(u^\varepsilon)_{\varepsilon>0} \in L^2(A)$ converges strongly in $L^2(A)$ towards $u^0 \in L^2(A)$ when ε tends to zero if $\lim_{\varepsilon\to 0} ||u^\varepsilon - u^0||_{L^2(A)} = 0$. The convergence is said to be weak if $\lim_{\varepsilon\to 0} \int_A (u^\varepsilon - u^0) v \, dx = 0$ for all $v \in L^2(A)$. We write $u^\varepsilon = u^0 + O_s(\varepsilon)$ (respectively $O_w(\varepsilon)$), where $O_s(\varepsilon)$ (respectively $O_w(\varepsilon)$) represents a sequence tending to zero strongly (respectively weakly) in

$L^2(A)$. Moreover, the simple notation $O(\varepsilon)$ refers to a sequence of numbers which simply tends to zero. We do not detail the related usual computation rules.
Let u^ε, the solution of a linear boundary value problem posed in Ω,

$$\left\{ \begin{array}{l} -\dfrac{d}{dx}(a^\varepsilon(x)\dfrac{du^\varepsilon(x)}{dx}) = f \text{ in } \Omega \\ u^\varepsilon = 0 \text{ on } \Gamma, \end{array} \right. \tag{59}$$

where the right-hand side

$$||f||_{L^2(\Omega)} \le C, \tag{60}$$

the coefficient $a^\varepsilon \in \mathcal{C}^\infty(\Omega)$ is $\varepsilon\Omega^1$-periodic, and there exist two positive constants α and β independent ε such that

$$0 < \alpha \le a^\varepsilon(x) \le \beta. \tag{61}$$

The weak formulation is obtained by multiplication of the differential equation by a test function $v \in \mathcal{C}^\infty_\Gamma(\Omega)$ and application of the Green formula,

$$\kappa^0 \int_\Omega a^\varepsilon(x)\frac{du^\varepsilon}{dx}\frac{dv}{dx}\ dx = \kappa^0 \int_\Omega f(x)v(x)\ dx. \tag{62}$$

Proposition 33 ***[Interpretation of a weak equality]*** *For $u \in L^2(A)$ and for any $v \in C^\infty_0(A)$,*

$$\text{if } \int_A u(x)\ v(x)\ dx = 0 \text{ then } u = 0$$

in the sense of $L^2(A)$ functions.

Proposition 34 ***[Interpretation of a periodic boundary condition]*** *For $u \in H^1(A)$ and for any $v \in \mathcal{C}^\infty_\#(A)$,*

$$\text{if } \int_{\partial A} u(x)\ v(x)\ n_{\partial A}(x)\ dx = 0 \text{ then } u \in H^1_\sharp(A).$$

In the remainder of this section, only the dimension $n = 1$ is considered, the general definitions being used for the elementary extensions.

Notation 35 ***[Physical and microscopic Domains]*** *We consider an interval $\Omega = \bigcup_{c=1}^{N(\varepsilon)} \Omega^{1,\varepsilon}_c \subset \mathbb{R}$ divided into $N(\varepsilon)$ periodic cells (or intervals) $\Omega^{1,\varepsilon}_c$, of size $\varepsilon > 0$, indexed by c, and with center x_c. The translation and magnification $(\Omega^{1,\varepsilon}_c - x_c)/\varepsilon$ is called the unit cell and is denoted by Ω^1. The variables in Ω and in Ω^1 are denoted by x^ε and x^1.*

The two-scale transform T is an operator mapping functions defined in the physical domain Ω to functions defined in the two-scale domain $\Omega^\sharp \times \Omega^1$ where for the *reference model* $\Omega^\sharp = \Omega$. In the following, we shall denote by Γ, $\Gamma^\sharp$ and Γ^1 the boundaries of Ω, $\Omega^\sharp$ and Ω^1.

Definition 36 ***[Two-Scale Transform]*** *The two-scale transform T is the linear operator defined by*

$$(Tu)(x_c, x^1) = u(x_c + \varepsilon x^1) \tag{63}$$

and then by extension $T(u)(x^\sharp, x^1) = u(x_c + \varepsilon x^1)$ for all $x^\sharp \in \Omega^{1,\varepsilon}_c$ and each c in $1, .., N(\varepsilon)$.

Notation 37 ***[Measure of Domains]*** $\kappa^0 = \frac{1}{|\Omega|}$ *and* $\kappa^1 = \frac{1}{|\Omega^\sharp \times \Omega^1|}$.

The operator T enjoys the following properties.

Proposition 38 ***[Product Rule]*** *For two functions* u, v *defined in* Ω,

$$T(uv) = (Tu)(Tv). \tag{64}$$

Proposition 39 ***[Derivative Rule]*** *If* u *and its derivative are defined in* Ω *then*

$$T\left(\frac{du}{dx}\right) = \frac{1}{\varepsilon}\frac{\partial(Tu)}{\partial x^1}. \tag{65}$$

Proposition 40 ***[Integral Rule]*** *If a function* $u \in L^1(\Omega)$ *then* $Tu \in L^1(\Omega^\sharp \times \Omega^1)$ *and*

$$\kappa^0 \int_\Omega u \ dx = \kappa^1 \int_{\Omega^\sharp \times \Omega^1} (Tu) \ dx^\sharp dx^1. \tag{66}$$

The next two properties are corollaries of the previous ones.

Proposition 41 ***[Inner Product Rule]*** *For two functions* u, $v \in L^2(\Omega)$,

$$\kappa^0 \int_\Omega u \ v \ dx = \kappa^1 \int_{\Omega^\sharp \times \Omega^1} (Tu) \ (Tv) \ dx^\sharp dx^1. \tag{67}$$

Proposition 42 ***[Norm Rule]*** *For a function* $u \in L^2(\Omega)$,

$$\kappa^0 \left\|u\right\|^2_{L^2(\Omega)} = \kappa^1 \left\|Tu\right\|^2_{L^2(\Omega^\sharp \times \Omega^1)}. \tag{68}$$

Definition 43 ***[Two-Scale Convergence]*** *A sequence* $u^\varepsilon \in L^2(\Omega)$ *is said to be two-scale strongly (respect. weakly) convergent in* $L^2(\Omega^\sharp \times \Omega^1)$ *to a limit* $u^0(x^\sharp, x^1)$ *if* Tu^ε *is strongly (respect. weakly) convergent towards* u^0 *in* $L^2(\Omega^\sharp \times \Omega^1)$.

Definition 44 ***[Adjoint or Dual of T]*** *As* T *is a linear operator from* $L^2(\Omega)$ *to* $L^2(\Omega^\sharp \times \Omega^1)$, *its adjoint* T^* *is a linear operator from* $L^2(\Omega^\sharp \times \Omega^1)$ *to* $L^2(\Omega)$ *defined by*

$$\kappa^0 \int_\Omega T^* v \ u \ dx = \kappa^1 \int_{\Omega^\sharp \times \Omega^1} v \ Tu \ dx^\sharp dx^1. \tag{69}$$

The expression of T^* can be detailed, it maps regular functions in $\Omega^\sharp \times \Omega^1$ to piecewise-constant functions in Ω. The next definition introduce an operator used as a smooth approximation of T^*.

Definition 45 ***[Regularization of T*]*** *The operator* B *is the linear continuous operator defined from* $L^2(\Omega^\sharp \times \Omega^1)$ *to* $L^2(\Omega)$ *by*

$$Bv = v(x, \frac{x}{\varepsilon}). \tag{70}$$

The nullity condition of a function $v(x^\sharp, x^1)$ on the boundary $\partial\Omega^\sharp \times \Omega^1$ is transferred to the range Bv as follows.

Proposition 46 ***[Boundary Conditions of Bv]*** *If* $v \in \mathcal{C}^\infty_{\Gamma^\sharp}(\Omega^\sharp; \mathcal{C}^\infty(\Omega^1))$ *then* $Bv \in \mathcal{C}^\infty_\Gamma(\Omega)$.

Proposition 47 ***[Derivation Rule for B]*** *If v and its partial derivatives are defined on $\Omega^\sharp \times \Omega^1$ then*

$$\frac{d(Bv)}{dx} = B(\frac{\partial v}{\partial x^\sharp}) + \varepsilon^{-1} B(\frac{\partial v}{\partial x^1}). \tag{71}$$

The next proposition states that the operator B is actually an approximation of the operator T^* for Ω^1-periodic functions.

Proposition 48 ***[Approximation between T^* and B]*** *If $v(x^\sharp, x^1)$ is continuous, continuously differentiable in $x^\sharp$ and Ω^1-periodic in x^1 then*

$$T^* v = Bv - \varepsilon B(x^1 \frac{\partial v}{\partial x^\sharp}) + \varepsilon O_s(\varepsilon) = B \sum_{\ell=0}^{1} \varepsilon^\ell [v, -x^1 \frac{\partial v}{\partial x^\sharp}]_\ell + \varepsilon O_s(\varepsilon). \tag{72}$$

Conversely,

$$Bv = T^*(v) + \varepsilon T^*(x^1 \frac{\partial v}{\partial x^\sharp}) + \varepsilon O_s(\varepsilon) = T^* \sum_{\ell=0}^{1} \varepsilon^\ell [v, x^1 \frac{\partial v}{\partial x^\sharp}]_\ell + \varepsilon O_s(\varepsilon). \tag{73}$$

Next, the formula of integration by parts is stated in a form compatible with the Green formula used in some extensions. The boundary Γ is composed of the two end points of the interval Ω, and the unit outward normal n_Γ defined on Γ is equal to -1 and $+1$ at the left- and right-endpoints respectively.

Proposition 49 ***[Green Rule]*** *If u, $v \in H^1(\Omega)$ then the traces of u and v on Γ are well defined and*

$$\int_\Omega u \frac{dv}{dx}\ dx = \int_\Gamma tr(u)\ tr(v)\ n_\Gamma\ ds(x) - \int_\Omega v \frac{du}{dx}\ dx. \tag{74}$$

The last proposition is stated as a building block of the homogenized model derivation.

Proposition 50 ***[The linear operator associated to the Microscopic problem]*** *For $\mu \in \mathbb{R}$, there exist $\theta^\mu \in H^1_\sharp(\Omega^1)$ solutions to the linear weak formulation*

$$\int_{\Omega^1} a^0 \frac{\partial \theta^\mu}{\partial x^1} \frac{\partial w}{\partial x^1}\ dx^1 = -\mu \int_{\Omega^1} a^0 \frac{\partial w}{\partial x^1}\ dx^1 \text{ for all } w \in \mathcal{C}^\infty_\sharp(\Omega^1), \tag{75}$$

and $\frac{\partial \theta^\mu}{\partial x^1}$ is unique. Since the mapping $\mu \mapsto \dfrac{\partial \theta^\mu}{\partial x^1}$ from $\mathbb{R}$ to $L^2(\Omega^1)$ is linear,

$$\frac{\partial \theta^\mu}{\partial x^1} = \mu \frac{\partial \theta^1}{\partial x^1}, \tag{76}$$

where θ^1 is solution to (75) for $\mu = 1$,

$$\int_{\Omega^1} a^0 \frac{\partial \theta^1}{\partial x^1} \frac{\partial w}{\partial x^1}\ dx^1 = -\int_{\Omega^1} a^0 \frac{\partial w}{\partial x^1}\ dx^1 \text{ for all } w \in \mathcal{C}^\infty_\sharp(\Omega^1). \tag{77}$$

Moreover, the relation (76) can be extended to any $\mu \in L^2(\Omega^\sharp)$.

For $d = [v, x^1 \frac{\partial v}{\partial x^\sharp}]$ and $d^* = [v, -x^1 \frac{\partial v}{\partial x^\sharp}]$ the next proposition states that d^* is adjoint of d for functions vanishing on $\Gamma^\sharp$.

Definition 51 ***[Adjoint of*** d_i***]*** *For* $u, v \in \mathcal{C}^\infty_{\Gamma^\sharp}(\Omega^\sharp; \mathcal{C}^\infty(\Omega^1))$,

$$\int_{\Omega^\sharp\times\Omega^1} u\ d_i v\ dx^\sharp dx^1 = \int_{\Omega^\sharp\times\Omega^1} d_i^* u\ v\ dx^\sharp dx^1 \text{ for } i \in \{0,1\}.$$

The next propositions are required for extensions only.
For *n-dim* extension.

Proposition 52 ***[Introduction of a Kronecker symbol]*** *For any functions* u *and* θ,

$$\frac{\partial u}{\partial x} + \frac{\partial u}{\partial x}\frac{\partial v}{\partial y} = (1 + \frac{\partial v}{\partial y})\frac{\partial u}{\partial x}.$$

For *n-dim* extension, Fifth block-step 2.1.

Proposition 53 ***[decomposition of a sum over a union]***

$$f = f.$$

For *ScalSol* extension, First Block-step 5.1.

Proposition 54 ***[Renumbering a Double Sum]***

$$c = c.$$

For *ScalSol* extension, First Block-step 6.1.

Proposition 55 ***[Identification of an Asymptotic Expansion]***

$$c = O(\varepsilon) \Longrightarrow c = 0.$$

For *ScalSol* extension, Sixth Block-step 1.1.

Proposition 56 ***[Interpretation of the Constraints]*** *If* v^0 *is defined in* $\Omega^\sharp \times \Omega^1$ *satisfies* $\frac{\partial v^0}{\partial x^1} = 0$ *then there exists a function* λ^0 *independent of* x^1 *such that* $v^0 = \lambda^0$.

6.1.2 Two-Scale Approximation of a Derivative

Here we detail the *reference computation* of the weak two-scale limit $\eta = \lim_{\varepsilon\to 0} T(\frac{du^\varepsilon}{dx})$ in $L^2(\Omega^\sharp \times \Omega^1)$ when

$$\|u^\varepsilon\|_{L^2(\Omega)} \text{ and } \left\|\frac{du^\varepsilon}{dx}\right\|_{L^2(\Omega)} \leq C, \tag{78}$$

C being a constant independent of ε. To simplify the proof, we further do the following assumption.

Assumption 57 ***[Approximation of*** Tu***]*** *There exist* u^0, $u^1 \in L^2(\Omega^\sharp \times \Omega^1)$ *such that for any* $k \in \{0,1\}$

$$T(u^\varepsilon) = \sum_{j=0}^{k} \varepsilon^j u^j + \varepsilon^k O_w(\varepsilon),$$

in particular for $k = 1$,

$$T(u^\varepsilon) = u^0 + \varepsilon u^1 + \varepsilon O_w(\varepsilon)$$

i.e.

$$\int_{\Omega^\sharp\times\Omega^1}(T(u^\varepsilon)-\sum_{j=0}^{k}\varepsilon^j u^j)v\ dx^\sharp dx^1=\varepsilon^k O_w(\varepsilon)\ \textit{for all}\ v\in L^2(\Omega^\sharp\times\Omega^1)$$

and in particular for $k=1$,

$$\int_{\Omega^\sharp\times\Omega^1}(T(u^\varepsilon)-u^0-\varepsilon u^1)v\ dx^\sharp dx^1=\varepsilon O(\varepsilon)\ \textit{for all}\ v\in L^2(\Omega^\sharp\times\Omega^1). \tag{79}$$

We quote that Assumption (79) is not mandatory, it is introduced to simplify the proof since it avoids some non-equational steps. The statement proved in the remaining of the subsection is the following.

Proposition 58 ***[Two-scale Limit of a Derivative]*** *If u^ε is a sequence bounded as in (78) and satisfying (79), then u^0 is independent of x^1,*

$$\tilde{u}^1=u^1-x^1\partial_{x^\sharp}u^0 \tag{80}$$

defined in $\Omega^\sharp\times\Omega^1$ is Ω^1-periodic and

$$\eta=\frac{\partial u^0}{\partial x^\sharp}+\frac{\partial\tilde{u}^1}{\partial x^1}. \tag{81}$$

Moreover, if $u^\varepsilon=0$ on Γ then $u^0=0$ on $\Gamma^\sharp$.

The proof is split into four Lemmas.

Lemma 59 ***[First Block: Constraint on** u^0**]*** *u^0 is independent of x^1.*

Proof. Source term. The weak formulation (62) is transformed into

$$\Psi=\varepsilon\kappa^0\int_\Omega\frac{du^\varepsilon}{dx}Bv\ dx$$

with $v\in\mathcal{C}^\infty_{\Gamma^\sharp}(\Omega^\sharp;\mathcal{C}^\infty_{\Gamma^1}(\Omega^1))$. From the Cauchy-Schwartz inequality and (78), $\lim_{\varepsilon\to0}\Psi=0$.

- **Step 1.** Propositions 49 and 46 $\Longrightarrow$

$$\Psi=-\varepsilon\kappa^0\int_\Omega u^\varepsilon\frac{d(Bv)}{dx}\ dx.$$

- **Step 2.** Proposition 47 and the boundness (78) $\Longrightarrow$

$$\Psi=\kappa^0\int_\Omega u^\varepsilon B(\frac{\partial v}{\partial x^1})\ dx+O(\varepsilon).$$

- **Step 3.** Proposition 48 $\Longrightarrow$

$$\Psi=\kappa^0\int_\Omega u^\varepsilon T^*(\frac{\partial v}{\partial x^1})\ \ dx+O(\varepsilon).$$

- **Step 4.** Definition 44 $\Longrightarrow$

$$\Psi = \kappa^1 \int_{\Omega^\sharp \times \Omega^1} T(u^\varepsilon) \frac{\partial v}{\partial x^1}\ dx + O(\varepsilon).$$

- **Step 5.** Assumption (79)

$$\kappa^1 \int_{\Omega^\sharp \times \Omega^1} (u^0 + O(\varepsilon)) \frac{\partial v}{\partial x^1}\ dx = 0.$$

  Passing to the limit when $\varepsilon \to 0 \Longrightarrow$

$$\kappa^1 \int_{\Omega^\sharp \times \Omega^1} u^0 \frac{\partial v}{\partial x^1}\ dx = 0.$$

- **Step 6.**
  Proposition 49 and $v = 0$ on $\Omega^\sharp \times \Gamma^1 \Longrightarrow$

$$\kappa^1 \int_{\Omega^\sharp \times \Omega^1} \frac{\partial u^0}{\partial x^1}\ v\ dx = 0.$$

- **Step 7.** Proposition 33 $\Longrightarrow$

$$\frac{\partial u^0}{\partial x^1} = 0.$$

■

Lemma 60 ***[Second Block: Two-Scale Limit of the Derivative]*** $\eta = \frac{\partial u^0}{\partial x^\sharp} + \frac{\partial \widetilde{u}^1}{\partial x^1}$.

Proof. Source term. The weak formulation (62) is transformed into

$$\Psi = \kappa^1 \int_{\Omega^\sharp \times \Omega^1} T(\frac{du^\varepsilon}{dx}) v\ dx^\sharp dx^1 \tag{82}$$

with $v \in \mathcal{C}^\infty_{\Gamma^\sharp}(\Omega^\sharp; \mathcal{C}^\infty_{\Gamma^1}(\Omega^1))$.

- **Step 1.** Definition 44 $\Longrightarrow$

$$\Psi = \kappa^0 \int_\Omega \frac{du^\varepsilon}{dx} T^* v\ dx.$$

- **Step 2.** Proposition 48 (to approximate T^* by B), the Green formula (74), Proposition 47, the linearity of integrals, and again Proposition 48 (to approximate B by T^*) $\Longrightarrow$

$$\Psi = -\kappa^0 \int_\Omega u^\varepsilon T^*(\frac{\partial v}{\partial x^\sharp})\ dx - \frac{\kappa^0}{\varepsilon} \int_\Omega u^\varepsilon T^*(\frac{\partial v}{\partial x^1})\ dx - \kappa^0 \int_\Omega u^\varepsilon T^*(x^1 \frac{\partial}{\partial x^\sharp}(\frac{\partial v}{\partial x^1}))\ dx + O(\varepsilon).$$

- **Step 3.** Definition 44 $\Longrightarrow$

$$\begin{aligned} \Psi \quad = \quad & -\kappa^1 \int_{\Omega^\sharp \times \Omega^1} T(u^\varepsilon) \frac{\partial v}{\partial x^\sharp}\ dx^\sharp dx^1 - \frac{\kappa^1}{\varepsilon} \int_{\Omega^\sharp \times \Omega^1} T(u^\varepsilon) \frac{\partial v}{\partial x^1}\ dx^\sharp dx^1 \\ & -\kappa^1 \int_{\Omega^\sharp \times \Omega^1} T(u^\varepsilon) d_1(\frac{\partial v}{\partial x^1})\ dx^\sharp dx^1 + O(\varepsilon). \end{aligned}$$

- **Step 4.** Assumption (79) $\Longrightarrow$

$$\begin{aligned}\Psi &= -\kappa^1\int_{\Omega^\sharp\times\Omega^1} u^0\frac{\partial v}{\partial x^\sharp}\ dx^\sharp dx^1 - \frac{\kappa^1}{\varepsilon}\int_{\Omega^\sharp\times\Omega^1} u^0\frac{\partial v}{\partial x^1}\ dx^\sharp dx^1 - \kappa^1\int_{\Omega^\sharp\times\Omega^1} u^1\frac{\partial v}{\partial x^1}\ dx^\sharp dx^1\\ &\quad -\kappa^1\int_{\Omega^\sharp\times\Omega^1} u^0 d_1(\frac{\partial v}{\partial x^1}) + O(\varepsilon).\end{aligned}$$

- **Step 5.** Proposition 51 of the adjoint of d $\Longrightarrow$

$$\begin{aligned}\Psi &= -\kappa^1\int_{\Omega^\sharp\times\Omega^1} u^0\frac{\partial v}{\partial x^\sharp}\ dx^\sharp dx^1 - \frac{\kappa^1}{\varepsilon}\int_{\Omega^\sharp\times\Omega^1} u^0\frac{\partial v}{\partial x^1}\ dx^\sharp dx^1\\ &\quad -\kappa^1\int_{\Omega^\sharp\times\Omega^1}(d_1^* u^0 + u^1)\frac{\partial v}{\partial x^1}\ dx^\sharp dx^1 + O(\varepsilon).\end{aligned}$$

- **Step 6.** The Green formula (74) $\Longrightarrow$

$$\begin{aligned}\Psi &= \kappa^1\int_{\Omega^\sharp\times\Omega^1} \frac{\partial u^0}{\partial x^\sharp} v\ dx^\sharp dx^1 + \frac{\kappa^1}{\varepsilon}\int_{\Omega^\sharp\times\Omega^1} \frac{\partial u^0}{\partial x^1} v\ dx^\sharp dx^1\\ &\quad +\kappa^1\int_{\Omega^\sharp\times\Omega^1}\frac{\partial(d_1^* u^0 + u^1)}{\partial x^1} v\ dx^\sharp dx^1 + O(\varepsilon).\end{aligned}$$

- **Step 7.** Factoring the common powers in ε $\Longrightarrow$

$$\Psi = \kappa^1\int_{\Omega^\sharp\times\Omega^1}(\frac{\partial u^0}{\partial x^\sharp} + \frac{\partial(d_1^* u^0 + u^1)}{\partial x^1}) v\ dx^\sharp dx^1 + \frac{\kappa^1}{\varepsilon}\int_{\Omega^\sharp\times\Omega^1}\frac{\partial u^0}{\partial x^1} v\ dx^\sharp dx^1 + O(\varepsilon).$$

- **Step 8.** The definition (80) of $\tilde{u}^1$ $\Longrightarrow$

$$\Psi = \kappa^1\int_{\Omega^\sharp\times\Omega^1}(\frac{\partial u^0}{\partial x^\sharp} + \frac{\partial \tilde{u}^1}{\partial x^1}) v\ dx^\sharp dx^1 + \frac{\kappa^1}{\varepsilon}\int_{\Omega^\sharp\times\Omega^1}\frac{\partial u^0}{\partial x^1} v\ dx^\sharp dx^1 + O(\varepsilon).$$

- **Step 9.** Lemma 59, and passing to the limit when $\varepsilon \to 0$ $\Longrightarrow$

$$\kappa^1\int_{\Omega^\sharp\times\Omega^1}\eta\ v\ dx^\sharp dx^1 = \kappa^1\int_{\Omega^\sharp\times\Omega^1}(\frac{\partial u^0}{\partial x^\sharp} + \frac{\partial \tilde{u}^1}{\partial x^1}) v\ dx^\sharp dx^1.$$

- **Step 10.** Proposition 33 $\Longrightarrow$

$$\eta = \frac{\partial u^0}{\partial x^\sharp} + \frac{\partial \tilde{u}^1}{\partial x^1}.$$

■

Lemma 61 ***[Third Block: Microscopic Boundary Condition]*** *$\tilde{u}^1$ is Ω^1-periodic.*

Proof. Source term. In (82), we choose $v \in \mathcal{C}^\infty_{\Gamma^\sharp}(\Omega^\sharp; \mathcal{C}^\infty_\sharp(\Omega^1))$.

- **Step 1.** The steps 1-8 of the second block $\Longrightarrow$

$$\kappa^1\int_{\Omega^\sharp\times\Omega^1}\eta v\ dx^\sharp dx^1 - \kappa^1\int_{\Omega^\sharp\times\Gamma^1}(u^1 - x^1\frac{\partial u^0}{\partial x^\sharp}) v\ n_{\Gamma^1}\ dx^\sharp dx^1 - \kappa^1\int_{\Omega^\sharp\times\Omega^1}\frac{\partial u^1}{\partial x^1} v\ dx^\sharp dx^1 = O(\varepsilon).$$

- **Step 2.** Lemma 60 $\Longrightarrow$

$$\int_{\Omega^\sharp\times\Gamma^1}(u^1 - x^1\frac{\partial u^0}{\partial x^\sharp})v\; n_{\Gamma^1}\; dx^\sharp ds(x^1) = O(\varepsilon). \tag{83}$$

- **Step 3.**
Definition (80) of $\tilde{u}^1$ and Proposition 34 $\Longrightarrow$

$$\tilde{u}^1 \text{ is } \Omega^1\text{-periodic.} \tag{84}$$

■

Lemma 62 ***[Fourth Block: Macroscopic Boundary Condition]*** *u^0 vanishes on $\Gamma^\sharp$.*

Proof. Source term. In (82), we choose $v \in \mathcal{C}^\infty(\Omega^\sharp)$,

- **Step 1.** The steps 1-5 of the second block and $u^\varepsilon = 0$ on $\Gamma \Longrightarrow$

$$\kappa^1\int_{\Omega^\sharp\times\Omega^1}\eta\; v\; dx^\sharp dx^1 = \kappa^1\int_{\Omega^\sharp\times\Omega^1}\frac{\partial u^0}{\partial x^0}v\; dx^\sharp dx^1 - \kappa^1\int_{\Gamma^\sharp\times\Omega^1}u^0 v n_{x^\sharp}\; ds(x^\sharp)dx^1.$$

- **Step 2.** Proposition 33 applied two times $\Longrightarrow$

$$\int_{\Omega^1}\eta\; dx^1 = |\Omega^1|\frac{\partial u^0}{\partial x^0} \text{ and } u^0 = 0 \text{ on } \Gamma^\sharp.$$

■

6.1.3 Homogenized Model Derivation

Here we provide the *reference proof* of the homogenized model derivation. It uses Proposition 58 as an intermediary result.

Proposition 63 ***[Boundness of the Solution]*** *The solution u^ε of (62) satisfies the boundness (78).*

Moreover, we assume that for some functions $a^0(x^1)$ and $f^0(x^\sharp)$,

$$T(a^\varepsilon) = a^0 \text{ and } T(f) = f^0(x^\sharp) + O_w(\varepsilon). \tag{85}$$

The next proposition states the homogenized model and is the main result of the *reference proof*. For θ^1 a solution to the microscopic problem (75) with $\mu = 1$, the homogenized coefficient and right-hand side are defined by

$$a^H = \int_{\Omega^1} a^0\left(1 + \frac{\partial\theta^1}{\partial x^1}\right)^2\; dx^1 \text{ and } f^H = \int_{\Omega^1} f^0\; dx^1. \tag{86}$$

Proposition 64 ***[Homogenized Model]*** *The limit u^0 is solution to the weak formulation*

$$\int_{\Omega^\sharp} a^H\frac{du^0}{dx^\sharp}\frac{dv^0}{dx^\sharp}\; dx^\sharp = \int_{\Omega^\sharp} f^H v^0\; dx^\sharp \tag{87}$$

for all $v^0 \in \mathcal{C}^\infty_{\Gamma^\sharp}(\Omega^\sharp)$.

The proof is split into three lemmas.

Lemma 65 ***[Fifth Block: Two-Scale Model]*** *The couple $(u^0, \widetilde{u}^1)$ is solution to the two-scale weak formulation*

$$\int_{\Omega^\sharp\times\Omega^1} a^0 \left(\frac{\partial u^0}{\partial x^\sharp} + \frac{\partial \widetilde{u}^1}{\partial x^1}\right)\left(\frac{\partial v^0}{\partial x^\sharp} + \frac{\partial v^1}{\partial x^1}\right)\ dx^\sharp dx^1 = \int_{\Omega^\sharp\times\Omega^1} f^0 v^0\ dx^\sharp dx^1 \tag{88}$$

for any $(v^i)_{i=0,1} \in \mathcal{C}^\infty_{\Gamma^\sharp}(\Omega^\sharp, C^\infty_\sharp(\Omega^1))$ such that

$$\frac{\partial v^0}{\partial x^1} = 0. \tag{89}$$

Proof. Source term. We choose test functions $v^0 \in \mathcal{C}^\infty_{\Gamma^\sharp}(\Omega^\sharp)$, $v^1 \in \mathcal{C}^\infty_{\Gamma^\sharp}(\Omega^\sharp, C^\infty_\sharp(\Omega^1))$.

- **Step 1** Posing $v = B(v^0 + \varepsilon v^1)$ in (62) and Proposition 46 $\Longrightarrow$

$$Bv \in \mathcal{C}^\infty_\Gamma(\Omega) \text{ and } \kappa^0 \int_\Omega a^\varepsilon \frac{du^\varepsilon}{dx}\frac{dB(v^0+\varepsilon v^1)}{dx}\ dx = \kappa^0\int_\Omega f\ B(v^0+\varepsilon v^1)\ dx.$$

- **Step 2**

  Propositions 47 $\Longrightarrow$

$$\kappa^0 \int_\Omega a^\varepsilon \frac{du^\varepsilon}{dx} B\left(\frac{\partial v^0}{\partial x^\sharp} + \frac{\partial v^1}{\partial x^1}\right) dx = \kappa^0\int_\Omega f\ B(v^0)dx + O(\varepsilon).$$

  Proposition 48 $\Longrightarrow$

$$\kappa^0 \int_\Omega a^\varepsilon \frac{du^\varepsilon}{dx} T^*\left(\frac{\partial v^0}{\partial x^\sharp} + \frac{\partial v^1}{\partial x^1}\right) dx = \kappa^0\int_\Omega f\ T^*(v^0)dx + O(\varepsilon).$$

- **Step 3** Definition 44 and Proposition 38 $\Longrightarrow$

$$\kappa^1 \int_{\Omega^\sharp\times\Omega^1} T(a^\varepsilon)T(\frac{du^\varepsilon}{dx})\left(\frac{\partial v^0}{\partial x^\sharp} + \frac{\partial v^1}{\partial x^1}\right)\ dx^\sharp dx^1 = \kappa^1\int_{\Omega^\sharp\times\Omega^1} T(f)\ v^0\ dx^\sharp dx^1 + O(\varepsilon). \tag{90}$$

- **Step 4**

  Definitions (85), Lemma 58, and passing to the limit when $\varepsilon \to 0 \Longrightarrow$

$$\int_{\Omega^\sharp\times\Omega^1} a^0 \left(\frac{\partial u^0}{\partial x^\sharp} + \frac{\partial \widetilde{u}^1}{\partial x^1}\right)\left(\frac{\partial v^0}{\partial x^\sharp} + \frac{\partial v^1}{\partial x^1}\right)\ dx^\sharp dx^1 = \int_{\Omega^\sharp\times\Omega^1} f^0 v^0\ dx^\sharp dx^1$$

  which is the expected result.

■

Lemma 66 ***[Sixth Block: Microscopic Problem]*** *$\widetilde{u}^1$ is solution to (75) with $\mu = \dfrac{\partial u^0}{\partial x^\sharp}$ and*

$$\frac{\partial \widetilde{u}^1}{\partial x^1} = \frac{\partial u^0}{\partial x^\sharp}\frac{\partial \theta^1}{\partial x^1}.$$

Proof. Source term. We choose $v^0 = 0$ and $v^1(x^\sharp, x^1) = w(x^1)\varphi(x^\sharp)$ in (88) with $\varphi \in \mathcal{C}^\infty(\Omega^\sharp)$ and $w^1 \in \mathcal{C}^\infty_\sharp(\Omega^1)$.

- **Step 1** Proposition 33, Lemma 59, and the linearity of the integral $\Longrightarrow$

$$\int_{\Omega^1} a^0 \frac{\partial \widetilde{u}^1}{\partial x^1}\frac{\partial w^1}{\partial x^1}\ dx^1 = -\frac{\partial u^0}{\partial x^\sharp}\int_{\Omega^1} a^0 \frac{\partial w^1}{\partial x^1}\ dx^1. \tag{91}$$

- **Step 2** Proposition 50 with $\mu = \dfrac{\partial u^0}{\partial x^\sharp} \Longrightarrow$

$$\frac{\partial \widetilde{u}^1}{\partial x^1} = \frac{\partial u^0}{\partial x^\sharp}\frac{\partial \theta^1}{\partial x^1}$$

  as announced.

■

Lemma 67 ***[Seventh Block: Macroscopic Problem]*** *u^0 is solution to (87).*

Proof. Source term. We choose $v^0 \in \mathcal{C}^\infty_{\Gamma^\sharp}(\Omega^\sharp)$ and $v^1 = \dfrac{\partial v^0}{\partial x^\sharp}\dfrac{\partial \theta^1}{\partial x^1} \in \mathcal{C}^\infty(\Omega^\sharp, C^\infty_\sharp(\Omega^1))$ in (88).

- **Step 1** Lemma 66 $\Longrightarrow$

$$\int_{\Omega^\sharp\times\Omega^1} a^0 \left(\frac{\partial u^0}{\partial x^\sharp} + \frac{\partial \theta^1}{\partial x^1}\frac{\partial u^0}{\partial x^\sharp}\right)\left(\frac{\partial v^0}{\partial x^\sharp} + \frac{\partial \theta^1}{\partial x^1}\frac{\partial v^0}{\partial x^\sharp}\right)\ dx^\sharp dx^1 = \int_{\Omega^\sharp\times\Omega^1} f^0 v^0\ dx^\sharp dx^1. \tag{92}$$

- **Step 2**

  Substep 2.1-n-dim Proposition 52

$$\int_{\Omega^\sharp\times\Omega^1} a^0 \left(1 + \frac{\partial \theta^1}{\partial x^1}\right)\left(1 + \frac{\partial \theta^1}{\partial x^1}\right)\frac{\partial u^0}{\partial x^\sharp}\frac{\partial v^0}{\partial x^\sharp}\ dx^\sharp dx^1 = \int_{\Omega^\sharp\times\Omega^1} f^0 v^0\ dx^\sharp dx^1.$$

- **Step 3** Factoring and definitions (86) $\Longrightarrow$

$$\int_{\Omega^\sharp} a^H \frac{\partial u^0}{\partial x^\sharp}\frac{\partial v^0}{\partial x^\sharp}\ dx^\sharp = \int_{\Omega^\sharp} f^H v^0\ dx^\sharp.$$

■

6.2 Extension to n-dimensional Regions

The concept of extension is illustrated, from the mathematical point of view when the dimension of the physical domain Ω is changed from 1 to any positive integer n and the variables $x \in \Omega$ are indexed as x_i with $i \in I$ a subset of integers with $|I| = n$.

We detail the extensions of the reference proof for each proposition that require a change and eventually add new substeps in the proof of lemmas. All required added contexts τ including a symbol $\Box$ are detailed but also the research patterns θ for each proposition. Insertion positions in matching terms are represented by a tilde over the root of the subterm to be moved. For instance, the term $\tau = h(a, \Box)$ is added in $g(a, \widetilde{f}(b, c))$ yielding $g(a, h(a, f(b, c)))$. Same added terms are often used in many propositions. We do not specify the precise strategies of extensions that are left to the programmer.

6.2.1 Notations, Definitions and Propositions

The boundary value problem (59) is replaced with

$$\begin{cases} -\sum_{i\in I}\sum_{j\in I}\frac{\partial}{\partial x_i}(a_{ij}^{\varepsilon}(x)\frac{\partial u^{\varepsilon}(x)}{\partial x_j}) = f \text{ in } \Omega \\ u^{\varepsilon} = 0 \text{ on } \Gamma. \end{cases} \tag{93}$$

The coefficients a_{ij}^{ε} satisfy the same regularity and periodicity as the coefficient a^{ε} in (59). They are also uniformly bounded and positive in the matrix sense,

$$0 < \alpha|\xi|^2 \leq \sum_{i\in I}\sum_{j\in I} a_{ij}^{\varepsilon}(x)\xi_i\xi_j \leq \beta|\xi|^2 \text{ for all } \xi \in \mathbb{R}^n. \tag{94}$$

The derivation of the weak formulation follows the same lines and yields,

$$\kappa^0 \sum_{i\in I}\sum_{j\in I}\int_{\Omega} a_{ij}^{\varepsilon}\frac{\partial u^{\varepsilon}}{\partial x_j}\frac{\partial v}{\partial x_i}\ dx = \kappa^0\int_{\Omega} f\ v\ dx. \tag{95}$$

The added terms are

$$\tau_{Multidim} = \{\Box_i,\ \Box_j, \Box_\ell, \sum_{\ell\in I}\Box, \Box_k, \delta_{kj} * \Box, \sum_{k\in I}\Box, \sum_{i\in I}\Box\}.$$

In the *Derivative Rule,* $\tau = \{\tau_1\}$ is used with two matching terms $\theta = (x, x^1)$ resulting into x_i and x_i^1.

Proposition 68 ***[Derivative Rule]*** *If u and its derivative are defined in Ω then*

$$T\left(\frac{\partial u}{\partial x_i}\right) = \frac{1}{\varepsilon}\frac{\partial (Tu)}{\partial x_i^1} \textit{ for any } i \in I. \tag{96}$$

In the *Derivation Rule for B*, $\tau = \tau_1$ is used with three matching terms $\theta = (x, x^\sharp, x^1)$ that are changed into x_i, $x_i^\sharp$ and x_i^1.

Proposition 69 ***[Derivation Rule for B]*** *If v and its partial derivatives are defined on $\Omega^\sharp \times \Omega^1$ then for $i \in I$,*

$$\frac{\partial (Bv)}{\partial x_i} = B(\frac{\partial v}{\partial x_i^\sharp}) + \varepsilon^{-1}B(\frac{\partial v}{\partial x_i^1}). \tag{97}$$

In the *Approximation between B and T**, $\tau = (\Box_i, \Box_i, \sum_{i\in I}\Box)$ and $\theta = (x^1 * \frac{\partial v}{\partial \widetilde{x^\sharp}}, \widetilde{x^1} * \frac{\partial v}{\partial x^\sharp}, x^1 \tilde{*} \frac{\partial v}{\partial x^\sharp})$ yielding $\sum_{i\in I} x_i^1 * \frac{\partial v}{\partial x_i^\sharp}$ instead of $x^1\frac{\partial v}{\partial x^\sharp}$.

Proposition 70 ***[Approximation between T* and B]*** *If $v(x^\sharp, x^1)$ is continuous, continuously differentiable in $x^\sharp$ and Ω^1-periodic in x^1 then*

$$T^*v = Bv - \varepsilon B(\sum_{i\in I} x_i^1\frac{\partial v}{\partial x_i^\sharp}) + \varepsilon O_s(\varepsilon) = B\sum_{\ell=0}^{1}\varepsilon^\ell[v, -\sum_{i\in I} x_i^1\frac{\partial v}{\partial x_i^\sharp}]_\ell + \varepsilon O_s(\varepsilon). \tag{98}$$

Conversely,

$$Bv = T^*(v) + \varepsilon T^*(\sum_{i\in I} x_i^1\frac{\partial v}{\partial x_i^\sharp}) + \varepsilon O_s(\varepsilon) = T^*\sum_{\ell=0}^{1}\varepsilon^\ell[v, -\sum_{i\in I} x_i^1\frac{\partial v}{\partial x_i^\sharp}]_\ell + \varepsilon O_s(\varepsilon). \tag{99}$$

In the *Green Rule*, $\tau = \tau_1$ so $\theta = (x, n_\Gamma)$ is changed into $(x_i, n_{\Gamma i})$.

Proposition 71 ***[Green Rule]*** *If u, $v \in H^1(\Omega)$ then the traces of u and v on Γ are well defined and*

$$\int_\Omega u \frac{\partial v}{\partial x_i}\ dx = \int_\Gamma tr(u)\ tr(v)\ n_{\Gamma i}\ ds(x) - \int_\Omega v \frac{\partial u}{\partial x_i}\ dx. \tag{100}$$

In the definition of *the linear operator associated to the Microscopic problem,* x^1 is replaced by a vector $(x_i)_{i\in I}$, $\mu \in \mathbb{R}$ is replaced by a vector $\mu \in \mathbb{R}^{|I|}$ and the microscopic problem is transformed accordingly. Its solution θ^μ is replaced by a function Θ^μ indexed by a vector. The special solution θ^1 is replaced by a family $(\theta^\ell)_{\ell=1,\dots,n}$ solution to the problem with $\mu = e_n$, the n^{th} unit vector of the canonical basis, and the equality $\dfrac{\partial \theta^\mu}{\partial x^1} = \mu \dfrac{\partial \theta^1}{\partial x^1}$ is replaced by $\dfrac{\partial \Theta^\mu}{\partial x^1_j} = \sum_{\ell\in I} \mu_\ell \dfrac{\partial \theta^\ell}{\partial x^1_j}$.

Proposition 72 ***[The linear operator associated to the Microscopic problem]*** *For $\mu \in \mathbb{R}^n$, there exist $\Theta^\mu \in H^1_\sharp(\Omega^1)$ solutions to the linear weak formulation*

$$\sum_{i\in I}\sum_{j\in I}\int_{\Omega^1} a^0_{ij} \frac{\partial \Theta^\mu}{\partial x^1_j}\frac{\partial w}{\partial x^1_i}\ dx^1 = -\sum_{i\in I}\sum_{j\in I}\mu_j \int_{\Omega^1} a^0_{ij}\frac{\partial w}{\partial x^1_i}\ dx^1 \text{ for all } w \in \mathcal{C}^\infty_\sharp(\Omega^1), \tag{101}$$

and the vector $(\dfrac{\partial \Theta^\mu}{\partial x^1_j})_{j\in I}$ is unique. Since the mapping $\mu \mapsto \dfrac{\partial \Theta^\mu}{\partial x^1_j}$ from $\mathbb{R}^n$ to $L^2(\Omega^1)$ is linear then

$$\frac{\partial \Theta^\mu}{\partial x^1_j} = \sum_{\ell\in I}\mu_\ell \frac{\partial \theta^\ell}{\partial x^1_j} \text{ for all } j \in I \tag{102}$$

where θ^ℓ is solution to (101) for $\mu = e_\ell$ (so $\mu_j = \delta_{\ell j}$)

$$\sum_{i\in I}\sum_{j\in I}\int_{\Omega^1} a^0_{ij}\frac{\partial \theta^\ell}{\partial x^1_j}\frac{\partial w}{\partial x^1_i}\ dx^1 = -\sum_{i\in I}\int_{\Omega^1} a^0_{i\ell}\frac{\partial w}{\partial x^1_i}\ dx^1 \text{ for all } w \in \mathcal{C}^\infty_\sharp(\Omega^1). \tag{103}$$

Moreover, the relation (102) can be extended to any $\mu \in (L^2(\Omega^\sharp))^n$.

In *Introduction of a Kronecker symbol*, using $\tau = (\square_i, \square_i, \square_k, \delta_{kj} * \square, \sum_{k\in I}\square)$, the matching terms $\theta = (\frac{\partial u}{\partial \widetilde{x}} + \frac{\partial u}{\partial x}\frac{\partial \theta}{\partial y}, \frac{\partial u}{\partial x} + \frac{\partial u}{\partial x}\frac{\partial \theta}{\partial \widetilde{y}}, \frac{\partial u}{\partial \widetilde{x}}\frac{\partial \theta}{\partial y}, \frac{\partial u}{\partial x}\widetilde{+}\frac{\partial u}{\partial x}\frac{\partial \theta}{\partial y}, u\widetilde{+}v)$ with $u, v, x, y \in \mathcal{X}^1$ are changed into $(\frac{\partial u}{\partial x_j} + \frac{\partial u}{\partial x}\frac{\partial \theta}{\partial y}, \frac{\partial u}{\partial x} + \frac{\partial u}{\partial x}\frac{\partial \theta}{\partial y_j}, \frac{\partial u}{\partial x_k}\frac{\partial \theta}{\partial y}, \delta_{kj}\frac{\partial u}{\partial x} + \frac{\partial u}{\partial x}\frac{\partial \theta}{\partial y}, \sum_{k\in I} u + \sum_{k\in I} v)$.

Proposition 73 ***[Introduction of a Kronecker symbol]*** *For any functions u and θ and any indices k, j varying in I,*

$$\frac{\partial u}{\partial x_j} + \sum_{k\in I}\frac{\partial u}{\partial x_k}\frac{\partial v}{\partial y_j} = \sum_{k\in I}(\delta_{kj} + \frac{\partial v}{\partial y_j})\frac{\partial u}{\partial x_k}.$$

6.2.2 Two-Scale Approximation of a Derivative

Here we seek the limit of the components $\eta_i = \lim_{\varepsilon\to 0} T(\frac{\partial u^\varepsilon}{\partial x_i})$ in $L^2(\Omega^\sharp \times \Omega^1)$ for $i \in I$ when

$$\|u^\varepsilon\|_{L^2(\Omega)} \text{ and } \left\|\frac{\partial u^\varepsilon}{\partial x_i}\right\|_{L^2(\Omega)} \leq C, \tag{104}$$

C being a constant independent of ε.

Proposition 74 ***[Two-scale Limit of a Derivative]*** *If u^ε is a sequence bounded as in (104) and satisfying (79), then u^0 is independent of x^1,*

$$\tilde{u}^1 = u^1 - \sum_{i\in I} x_i^1 \frac{\partial u^0}{\partial x_i^\sharp} \tag{105}$$

defined in $\Omega^\sharp \times \Omega^1$ is Ω^1-periodic, and

$$\eta_i = \frac{\partial u^0}{\partial x_i^\sharp} + \frac{\partial \tilde{u}^1}{\partial x_i^1} \text{ for all } i \in I. \tag{106}$$

Moreover, if $u^\varepsilon = 0$ on Γ then $u^0 = 0$ on $\Gamma^\sharp$.

Lemma 75 ***[First Block: Constraint on u^0]*** *u^0 is independent of x^1.*

Proof extension. Source term. Using $\tau = \tau_1$, $\theta = \frac{\partial u^\varepsilon}{\partial \widetilde{x}}$ and the source term (95) instead of (62) yields a rule creating the source term

$$\Psi = \varepsilon\kappa^0 \int_\Omega \frac{\partial u^\varepsilon}{\partial x_i} Bv \; dx \text{ for each } i \in I.$$

- **Step 1-7** are unchanged replacing Propositions 48 and 49 with Propositions 70 and 71 $\Longrightarrow$

$$\frac{\partial u^0}{\partial x_i^1} = 0.$$

■

Lemma 76 ***[Second Block: Two-Scale Limit of the Derivative]*** *$\eta_i = \frac{\partial u^1}{\partial x_i^1}$ for all $i \in I$.*

Proof extension. The initial term is replaced by

$$\Psi = \kappa^1 \int_{\Omega^\sharp\times\Omega^1} T(\frac{\partial u^\varepsilon}{\partial x_i}) v \; dx^\sharp dx^1 \text{ for each } i \in I. \tag{107}$$

- **Step 1** and **Step 3-6** are unchanged. In **Step 2** Propositions 48 and 49 are replaced with Propositions 70 and 71 $\Longrightarrow$

$$\eta_i = \frac{\partial u^1}{\partial x_i^1}.$$

■

Lemma 77 ***[Third Block: Microscopic Boundary Condition]*** *$\tilde{u}^1$ is Ω^1-periodic.*

Proof extension. The initial term is replaced by (107). Then, the steps are the same except that Lemma **??** replaces Lemma 76 and Definition (**??**) of $\tilde{u}^1$ replaces Definition (105). ■

Lemma 78 ***[Fourth Block: Macroscopic Boundary Condition]*** *u^0 vanishes on $\Gamma^\sharp$.*

Proof extension.

- **Step 1.** The steps 1-5 of the proof of Lemma 60 are replaced with those of Lemma 76 $\Longrightarrow$

$$\int_{\Gamma^\sharp \times \Omega^1} u^0 v \; n_{\Gamma^\sharp i} \; ds(x^\sharp) dx^1 = 0.$$

- **Step 2** is unchanged $\Longrightarrow$

$$u^0 = 0 \text{ on } \Gamma^\sharp.$$

■

6.2.3 Homogenized Model Derivation

Proposition 79 ***[Boundness of the Solution]*** *The solution u^ε of (95) satisfies the boundness (104).*

Moreover, we assume that there exists $a^0(x^1)$ and $f^0(x^\sharp)$ such that

$$T(a^\varepsilon) = a^0 \text{ and } T(f) = f^0 + O_w(\varepsilon). \tag{108}$$

In the statement of the homogenized model, the expression of the homogenized coefficients (86) is extended on the form of the matrix of coefficients,

$$a^H_{k\ell} = \sum_{i\in I} \sum_{j\in I} \int_{\Omega^1} a^0_{ij} \left(\delta_{kj} + \frac{\partial \theta^{e_k}}{\partial x^1_j} \right) \left(\delta_{\ell i} + \frac{\partial \theta^{e_\ell}}{\partial x^1_i} \right) \; dx^1 \tag{109}$$

where δ is the Kronecker symbol.

Proposition 80 ***[Homogenized Model]*** *The limit u^0 is solution to the weak formulation*

$$\sum_{k\in I} \sum_{\ell\in I} \int_{\Omega^\sharp} a^H_{k\ell} \frac{\partial u^0}{\partial x^\sharp_k} \frac{\partial v^0}{\partial x^\sharp_\ell} \; dx^\sharp = \int_{\Omega^\sharp} f^H v^0 \; dx^\sharp \tag{110}$$

for all $v^0 \in \mathcal{C}^\infty_{\Gamma^\sharp}(\Omega^\sharp)$.

The extension of its derivation consists in extensions of Lemma 65-67.

Lemma 81 ***[Fifth Block: Two-Scale Model]*** *The couple $(u^0, \tilde{u}^1)$ is solution to the two-scale weak formulation*

$$\sum_{i\in I} \sum_{j\in I} \int_{\Omega^\sharp \times \Omega^1} a^0_{ij} \left(\frac{\partial u^0}{\partial x^\sharp_j} + \frac{\partial \tilde{u}^1}{\partial x^1_j} \right) \left(\frac{\partial v^0}{\partial x^\sharp_i} + \frac{\partial v^1}{\partial x^1_i} \right) \; dx^\sharp dx^1 = \int_{\Omega^\sharp \times \Omega^1} f^0 v^0 \; dx^\sharp dx^1 \tag{111}$$

for any $(v^i)_{i=0,1} \in \mathcal{C}^\infty_{\Gamma^\sharp}(\Omega^\sharp, C^\infty_\sharp(\Omega^1))$ such that

$$\frac{\partial v^0}{\partial x^1} = 0. \tag{112}$$

Proof extension. Source term. Unchanged.

- **Step 1** The initial term i.e. the weak formulation (62) is replaced by (95) $\Longrightarrow$

$$Bv \in \mathcal{C}^\infty_\Gamma(\Omega) \text{ and } \kappa^0 \sum_{i\in I}\sum_{j\in I}\int_\Omega a^\varepsilon_{ij}\frac{\partial u^\varepsilon}{\partial x_j}\frac{\partial B(v^0+\varepsilon v^1)}{\partial x_i}\ dx = \kappa^0\int_\Omega f\ B(v^0+\varepsilon v^1)\ dx.$$

- **Step 2**

  Proposition 69 instead of 47

$$\kappa^0 \sum_{i\in I}\sum_{j\in I}\int_\Omega a^\varepsilon_{ij}\frac{\partial u^\varepsilon}{\partial x_j}B\left(\frac{\partial v^0}{\partial x^\sharp_i}+\frac{\partial v^1}{\partial x^1_i}\right)dx = \kappa^0\int_\Omega f\ T^*(v^0)dx + O(\varepsilon).$$

  Proposition 70 instead of48 $\Longrightarrow$

$$\kappa^0 \sum_{i\in I}\sum_{j\in I}\int_\Omega a^\varepsilon_{ij}\frac{\partial u^\varepsilon}{\partial x_j}T^*\left(\frac{\partial v^0}{\partial x^\sharp_i}+\frac{\partial v^1}{\partial x^1_i}\right)dx = \kappa^0\int_\Omega f\ T^*(v^0)dx + O(\varepsilon).$$

- **Step 3** is unchanged $\Longrightarrow$

$$\kappa^1 \sum_{i\in I}\sum_{j\in I}\int_{\Omega^\sharp\times\Omega^1} T(a^\varepsilon_{ij})T(\frac{\partial u^\varepsilon}{\partial x_j})\left(\frac{\partial v^0}{\partial x^\sharp_i}+\frac{\partial v^1}{\partial x^1_i}\right)\ dx^\sharp dx^1 = \kappa^1\int_{\Omega^\sharp\times\Omega^1} T(f)\ v^0\ dx^\sharp dx^1 + O(\varepsilon). \tag{113}$$

- **Step 4**

  Lemma 58 is replaced with its extension i.e. Lemma 74 $\Longrightarrow$

$$\sum_{i\in I}\sum_{j\in I}\int_{\Omega^\sharp\times\Omega^1} a^0_{ij}\left(\frac{\partial u^0}{\partial x^\sharp_j}+\frac{\partial \widetilde{u}^1}{\partial x^1_j}\right)\left(\frac{\partial v^0}{\partial x^\sharp_i}+\frac{\partial v^1}{\partial x^1_i}\right)\ dx^\sharp dx^1 = \int_{\Omega^\sharp\times\Omega^1} f^0 v^0\ dx^\sharp dx^1.$$

■

Lemma 82 ***[Sixth Block: Microscopic Problem]*** *$\widetilde{u}^1$ is solution to (101) with $\mu_j = \dfrac{\partial u^0}{\partial x^\sharp_j}$ and*

$$\frac{\partial \widetilde{u}^1}{\partial x^1_j} = \sum_{\ell\in I}\frac{\partial u^0}{\partial x^\sharp_\ell}\frac{\partial \theta^{e_\ell}}{\partial x^1_j}.$$

Proof extension. Source term. The initial term (88) is replaced by (111).

- **Step 1** Using Lemma 75 the extension of Lemma 59 $\Longrightarrow$

$$\sum_{i\in I}\sum_{j\in I}\int_{\Omega^1} a^0_{ij}\frac{\partial \widetilde{u}^1}{\partial x^1_j}\frac{\partial w^1}{\partial x^1_i}\ dx^1 = -\sum_{i\in I}\sum_{j\in I}\frac{\partial u^0}{\partial x^\sharp_j}\int_{\Omega^1} a^0_{ij}\frac{\partial w^1}{\partial x^1_i}\ dx^1. \tag{114}$$

- **Step 2** Proposition 72 the extension of Proposition 50 with $\mu_j = \frac{\partial u^0}{\partial x_j^\sharp}$ instead of $\mu = \frac{\partial u^0}{\partial x^\sharp}$ $\Longrightarrow$

$$\frac{\partial \widetilde{u}^1}{\partial x_j^1} = \sum_{k\in I} \frac{\partial u^0}{\partial x_k^\sharp} \frac{\partial \theta^{e_k}}{\partial x_j^1}.$$

■

Lemma 83 ***[Seventh Block: Macroscopic Problem]*** *u^0 is solution to (110).*

Proof extension. Source term Using $\tau = \{\tau_3, \tau_1, \tau_4\}$, $\theta = (\widetilde{\frac{\partial v}{\partial x^\sharp}}, \widetilde{\frac{\partial v}{\partial x^1}}, \theta^1, x\widetilde{*}y)$ with $v, x, y \in \mathcal{X}^0$ in the extension, the test function $v^1 = \frac{\partial v^0}{\partial x^\sharp}\frac{\partial \theta^1}{\partial x^1}$ is replaced with $v^1 = \sum_{\ell\in I} \frac{\partial v^0}{\partial x_\ell^\sharp}\frac{\partial \theta^{e_\ell}}{\partial x_i^1}$ in the extension (111) of (88).

- **Step 1** Lemma 82 as an extension of Lemma 66 $\Longrightarrow$

$$\sum_{i\in I}\sum_{j\in I}\int_{\Omega^\sharp\times\Omega^1} a_{ij}^0 \left(\frac{\partial u^0}{\partial x_j^\sharp} + \sum_{k\in I}\frac{\partial u^0}{\partial x_k^\sharp}\frac{\partial \theta^{e_k}}{\partial x_j^1}\right)\left(\frac{\partial v^0}{\partial x_i^\sharp} + \sum_{\ell\in I}\frac{\partial v^0}{\partial x_\ell^\sharp}\frac{\partial \theta^{e_\ell}}{\partial x_i^1}\right) dx^\sharp dx^1 = \int_{\Omega^\sharp\times\Omega^1} f^0 v^0 \; dx^\sharp dx^1. \tag{115}$$

- **Step 2**

  Substep 2.1-n-dim Proposition 73 and factoring the sums $\sum_{k\in I}\sum_{\ell\in I}$ $\Longrightarrow$

$$\sum_{i\in I}\sum_{j\in I}\sum_{k\in I}\sum_{\ell\in I}\int_{\Omega^\sharp\times\Omega^1} a_{ij}^0\left(\delta_{kj} + \frac{\partial\theta^{e_k}}{\partial x_j^1}\right)\left(\delta_{\ell i} + \frac{\partial\theta^{e_\ell}}{\partial x_i^1}\right)\frac{\partial u^0}{\partial x_k^\sharp}\frac{\partial v^0}{\partial x_\ell^\sharp}\; dx^1 dx^\sharp = \int_{\Omega^\sharp\times\Omega^1} f^0 \; v^0 \; dx^\sharp dx^1.$$

- **Step 3** the sums $\sum_{i\in I}\sum_{j\in I}$ are permuted with the integral $\int_{\Omega^\sharp}$ and Definition (109) of a^H is used instead of (86) $\Longrightarrow$

$$\sum_{k\in I}\sum_{\ell\in I}\int_{\Omega^\sharp} a_{k\ell}^H \frac{\partial u^0}{\partial x_k^\sharp}\frac{\partial v^0}{\partial x_\ell^\sharp}\; dx^\sharp = \int_{\Omega^\sharp} f^H v^0 \; dx^\sharp.$$

■

7 Implementation of the reference proof in the User Language

The implementation of the reference proof in the User Language follows the mathematical formulation in Section 6.1. However, it starts with usual mathematical rules as expansions or factorizations in Section 7.1. The implementations of the propositions is in Section 7.2 when the seven lemma proofs, called blocks, are in Sections 7.3-7.9. Each proof is made with rule definitions and with a list of steps using these rules as well as predefined general rules. In addition, each proof operates on a so-called "input source term" specified in the mathematical part of the document.

7.1 Usual mathematical rules

The following code represents the mathematical properties used to simplify formula. For example a term $t = a + 0$ is simplified into $t = a$ by application of the strategy *simplify_math*.

```
Rule
% ==============================================================================
%        "simplify_minus" : remove a - a in an expression
% ==============================================================================
   simplify_minus1 : a_ - a_ + b_ → b_
   simplify_minus2 : a_ - a_ → 0
% ==============================================================================
%        "mult_with_inverse" : remove a/a in an expression
% ==============================================================================
   mult_with_inverse1 : a_/a_ → 1            if a_ ≠ 0
   mult_with_inverse2 : -a_/a_ → -1          if a_ ≠ 0
   mult_with_inverse3 : a_/-a_ → -1          if a_ ≠ 0
   mult_with_inverse4 : -a_/-a_ → 1          if a_ ≠ 0
   mult_with_inverse5 : a_•b_/a_ → b_        if a_ ≠ 0
   mult_with_inverse6 : -a_•b_/a_ → -1•b_    if a_ ≠ 0
   mult_with_inverse7 : a_•b_/-a_ → -1•b_    if a_ ≠ 0
   mult_with_inverse8 : -a_•b_/-a_ → b_      if a_ ≠ 0
% ==============================================================================
%        "mult_with_inv_power" : remove a•a^(-1) in an expression
% ==============================================================================
   mult_with_inv_power1 : a_•a_^(-1) → 1        if a_ ≠ 0
   mult_with_inv_power2 : -a_•a_^(-1) → -1      if a_ ≠ 0
   mult_with_inv_power3 : a_•-a_^(-1) → -1      if a_ ≠ 0
   mult_with_inv_power4 : -a_•-a_^(-1) → 1      if a_ ≠ 0
   mult_with_inv_power5 : a_•a_^(-1)•b_ → b_    if a_ ≠ 0
% ==============================================================================
%        "simplify_math" : 1. Plus with 0
%                          2. Multiply with 0
%                          3. Integrate of 0
%                          4. Multiply with inverse
%                          5. Simplify minus
%                      %   6. Multiply with a^(-1)
% ==============================================================================
   plus_0 : 0 + a_ → a_
   mult_0 : 0•a_ → 0
% ==============================================================================
%        "simplify_sign" :   1.
%                            2.
%                            3.
%        "expan_sign" :
% ==============================================================================
   simplify_sign1 : -(-(a_)) → a_
   simplify_sign2 : ∫ -a_ đx_  → -∫ a_ đx_
```

```
  simplify_sign2 : ∫ -a_ đx_  → -∫ a_ đx_
  simplify_sign3 : 0 = -a_ → 0 = a_
  simplify_sign4 : -a_ = 0  → a_ = 0
  simplify_sign5 : -(b_•-(a_)) → b_•a_

  mult_minus_1a : -1•-1 → 1
  mult_minus_1b : -1•-1•a_ → a_
% ==========================================================================
%       "minus_to_minus_1" : change -a into -1•a
%       "minus_1_to_minus" :
% ==========================================================================
  minus_to_minus_1 : -a_ → -1•a_
  minus_1_to_minus : -1•a_ → -a_
% ==========================================================================
%       "expansion" :
%       "inverse_expansion" :
% ==========================================================================
  expansion1 : a_•(b_+c_) → a_•b_ + a_•c_
  expansion2 : a_•-(b_+c_) → -a_•b_-a_•c_
  inverse_expansion1 : a_•b_ + a_•c_ → a_•(b_ + c_)
  inverse_expansion2 : -a_•b_ + a_•c_ → a_•(-b_ + c_)
  inverse_expansion3 : a_•b_ - (a_)•c_ → a_•(b_ + -c_)
  inverse_expansion4 : a_•b_ + -a_•(c_+d_) → a_•(b_ - c_ + d_)
% ==========================================================================
%       "mult_with_1" :
% ==========================================================================
  mult_with_1a : a_•1 → a_
  mult_with_1b : a_•1•b_ → a_•b_

  sim_diva : c_•a_/a_ → c_
  sim_divb : c_•-a_/a_ → -c_
  sim_divc : c_•b_/a_ → b_•c_/a_
% ==========================================================================
%       "integral_linearity" :
%       "inverse_integral_linearity" :
%       "integral_1" :
%       "integral_0" :
% ==========================================================================
  integral_linearity : ∫ a_ + b_ đx_ → ∫ a_ đx_ + ∫ b_ đx_
  inverse_integral_linearity : ∫ a_ đx_ + ∫ b_ đx_ → ∫ a_+b_ đx_
  integral_1 : ∫1 đx_ → meas_xReg
  integral_0 : ∫ 0 đx_ → 0
% ==========================================================================
%       "pretty_integral" : 1. Put intergral behind other termss
% ==========================================================================
  pretty_integral1 : a_•∫b_ đx_ → a_•∫b_ đx_ if theta(a_) = ∅
% ==========================================================================
```

```
%       "take_constant_out_of_integral" : not only constant, but also
%       variable that is considered as a constant
% ================================================================================
   take_constant_out_of_integral0 : ∫ y_•a_ đx_ → y_•∫ a_ đx_
                                    if not(x_ ∈ theta(y_))
   take_constant_out_of_integral1 : ∫ const_•a_ đx_ → const_•∫ a_ đx_
                                    if (theta(const_) = ∅) or (const_ = eps)
                                    or (const_ = 1/eps)
   take_constant_out_of_integral2 : ∫1/const_•a_ đx_ → 1/const_•∫a_ đx_
                                    if theta(const_) = ∅ or const_ = eps
   take_constant_out_of_integral3 : ∫const_•a_•b_ đx_ → const_•∫a_•b_ đx_
                                    if theta(const_) = ∅ or const_ = eps
   take_constant_out_of_integral4 : ∫∫ a_•b_•c_ đx_ đy_ → ∫(∫ a_•c_ đx_)•b_đy_
                                    if not(x_ ∈ theta(b_)) or const_ = eps
   take_constant_out_of_integral5 : ∫ ∂u0/∂y_•b_ đx_ → ∂u0/∂y_•∫ b_ đx_
   take_constant_out_of_integral6 : ∫ ∂u_/∂xs•b_ đx1 → (∫ b_ đx1)•∂u_/∂xs
% ================================================================================
%       "remove_constant_from_equation_equal_0" : simplify the equation = 0
% ================================================================================
   remove_constant_from_equation_equal_0_v1 :  0 = a_•b_  → 0 = b_ if theta(a_) = ∅
   remove_constant_from_equation_equal_0_v2 : a_•b_ = 0 → b_ = 0 if theta(a_) = ∅
% ================================================================================
%       "pretty_meas" : 1.
% ================================================================================
   pretty_meas1 : 1/meas_rw_•∫b_ đx_ → 1/meas_rw_•∫b_ đx_
   pretty_meas2 : -1/meas_rw_•∫b_ đx_ → -1/meas_rw_•∫b_ đx_
   pretty_meas3 : b_/meas_rw_ → 1/meas_rw_•b_
   pretty_meas4 : a_/meas_rw_ → 1/meas_rw_•a_
   pretty_meas5 : -(a_•1)/meas_rw_ → -(a_)•1/meas_rw_
% ================================================================================
%       "simplify_derivative" : 1. derivative of ∂f(x)/∂y = 0
%                               2. derivative of a constant
%       "derivative_product_rule" : ∂f•g/∂x = ∂f/∂x•g + ∂g/∂x•f
%       "derivative_change_order" :
%       "linearity_derivative" :
%       "take_constant_out_of_derivative" :
% ================================================================================
   simplify_derivative1 : ∂v_/∂x_ → 0 if (v_.Variable(1) ≠ x_)
                          and (v_.Variable(2) ≠ x_)
   simplify_derivative2 : ∂v_/∂x_ → 0 if (v_.Variable ≠ x_) and (v_ ≠ ut1)
   simplify_derivative3 : ∂0/∂x_ → 0
   simplify_derivative4 : ∂v_/∂x_ → 0 if (v_.Variable ≠ x_) and (v_ ≠ v1)
   derivative_product_rule : ∂(a_•b_)/∂x_ → ∂a_/∂x_•b_ + ∂b_/∂x_•a_
   derivative_change_order : ∂(∂a_/∂y_)/∂x_ → ∂(∂a_/∂x_)/∂y_
   linearity_derivative : ∂(a_ + b_)/∂x_ → ∂a_/∂x_ + ∂b_/∂x_
   take_constant_out_of_derivative1 : ∂(const_•a_)/∂x_ → const_•∂a_/∂x_
                                      if theta(const_)= ∅
```

```
    take_constant_out_of_derivative2 : ∂(a_•b_)/∂x_ → b_•∂a_/∂x_
                                       if not(x_ ∈ theta(b_))
% ==============================================================================
%       "simplify_equation" :
% ==============================================================================
   simplify_equation1 : a_•b_ = a_•c_ → b_ = c_
   simplify_equation2 : a_ + b_ = a_ + c_ → b_ = c_
   simplify_equation3 : a_ = a_ + c_ → 0 = c_
   simplify_equation4 : a_•b_ = 0 → b_ = 0 if theta(a_) = ∅
   simplify_equation5 : -a_ = -b_ → a_ = b_
       change_side_left : a_ = b_ → a_ - b_ = 0
       change_side_right : a_ + b_ = 0 → b_ = -a_
    Strategy
       sim_div :   sim_diva
       | sim_divb
       | sim_divc

       simplify_sign :   simplify_sign1
       | simplify_sign2
       | simplify_sign3
       | simplify_sign4
       | simplify_sign5
       | mult_minus_1a
       | mult_minus_1b

       simplify_minus :   simplify_minus1
       | simplify_minus2

       mult_with_inverse :   mult_with_inverse1
       | mult_with_inverse2
       | mult_with_inverse3
       | mult_with_inverse4
       | mult_with_inverse5
       | mult_with_inverse6
       | mult_with_inverse7
       | mult_with_inverse8

       mult_with_1 :   mult_with_1a
       | mult_with_1b s

       expansion :   expansion1
       | expansion2

       simplify_math :
       plus_0
       | mult_0
       | integral_0
```

```
   | mult_with_inverse
   | simplify_minus

   take_constant_out_of_integral : take_constant_out_of_integral1
   | take_constant_out_of_integral2
   | take_constant_out_of_integral3

   remove_constant_from_equation_equal_0 :
     remove_constant_from_equation_equal_0_v1
   | remove_constant_from_equation_equal_0_v2

   pretty_integral :   pretty_integral1

   pretty_meas :   pretty_meas1
   | pretty_meas2
   | pretty_meas3
   | pretty_meas4
   | pretty_meas5

   inverse_expansion :   inverse_expansion1
   | inverse_expansion2
   | inverse_expansion3
   | inverse_expansion4

   simplify_derivative :   simplify_derivative1
   | simplify_derivative2
   | simplify_derivative3

   take_constant_out_of_derivative :   take_constant_out_of_derivative1
   | take_constant_out_of_derivative2

   simplify_equation :   simplify_equation1
   | simplify_equation2
   | simplify_equation3
   | simplify_equation4
   | simplify_equation5
Rule % (O(epsilon))
% ===============================================================================
%        "mult_with_oe" : any term multiply with oe = oe
% ===============================================================================
   mult_with_oe1 : a_•oe_eps → oe_eps
   mult_with_oe2 : a_•-oe_eps → oe_eps
   mult_with_oe3 : -oe_eps → oe_eps

% ===============================================================================
%        "eps_to_oe" : replace eps by O(eps)
% ===============================================================================
```

```
   eps_to_oe1 : a_/eps → a_/eps
   eps_to_oe2 : a_ + eps → a_ + oe_eps
   eps_to_oe3 : a_•eps → a_•oe_eps

% ==============================================================================
%       "change_side_oe" :
% ==============================================================================
   change_side_oe1 : oe_eps = oe_eps + a_  → oe_eps = a_
   change_side_oe2 : oe_eps + a_ = oe_eps + b_  → a_ = b_ + oe_eps

% ==============================================================================
%       "plus_oe" :
% ==============================================================================
   plus_oe1 : oe_eps + oe_eps → oe_eps
   plus_oe2 : a_ + oe_eps + oe_eps → a_ + oe_eps

% ==============================================================================
%       "pretty_oe" : simplify oe from both side of an equation
% ==============================================================================
   pretty_oe1 : oe_eps = -a_  → oe_eps = a_
   pretty_oe2 : oe_eps = -a_•b_  → oe_eps = a_•b_
   pretty_oe3 : oe_eps = -1•a_  → oe_eps = a_

% ==============================================================================
%       "simplify_multi_scale" : 1. Multiply with oe
%                                2. Integral of oe
%                                3. Plus oe
%                                4. Change side oe
%                                5. Pretty oe
% ==============================================================================
   integral_of_oe : ∫oe_eps đx_  → oe_eps
   oe_pass_eps_to_0_rule : oe_eps → 0

% ==============================================================================
%       "remove_oe" :
% ==============================================================================
   remove_oe_from_addition : a_ + oe_eps → a_
Strategy % (multiscale)
   mult_with_oe :   mult_with_oe1
   | mult_with_oe2
   | mult_with_oe3

   eps_to_oe :   eps_to_oe1
   | eps_to_oe2
   | eps_to_oe3

   change_side_oe :   change_side_oe1
```

```
        | change_side_oe2

        pretty_oe :   pretty_oe1
        | pretty_oe2
        | pretty_oe3

        simplify_multi_scale :   mult_with_oe
        | integral_of_oe
        | plus_oe
        | change_side_oe
        | pretty_oe
```

7.2 Propositions specialized to two-scale approximation

All implementations of the propositions in Subsection 6.1.1 are grouped in this section.

```
   Operator % (GENERAL DEFINITION)
        opB_rw_ : "B" [opB_rw_Ind_] [opB_rw_Fun_] [opB_rw_InV1_,opB_rw_InV2_]
                  [opB_rw_OutV_] [opB_rw_Pa_]
        % B() : L2(Ω#xΩ1) → L2(Ωε)

        opT_rw_  : "T" [opT_rw_Ind_] [opT_rw_Fun_] [opT_rw_InV_]
                   [opT_rw_OutV1_,opT_rw_OutV2_] [opT_rw_Pa_]
        % T() : L2(Ωε) → L2(Ω#xΩ1)

    opTS_rw_ : "TS" [opTS_rw_Ind_] [opTS_rw_Fun_] [opTS_rw_InV1_, opTS_rw_InV2_]
               [opTS_rw_OutV_] [opTS_rw_Pa_]
    % TS(): L2(Ω#xΩ1) → L2(Ωε)

    trace_rw_: "Trace" [trace_rw_Ind_] [trace_rw_Fun_] [trace_rw_InV_]
               [trace_rw_OutV_] [trace_rw_Pa_]

 Function
    oe_rw_TS_Pa : "oe" [oe_rw_Ind_] [eps] [] "Given"
    %  O(epsTS)

 % ==========================================================================
 %        "interpretation_of_a_weak_equality"
 % P72 :
 % ==========================================================================
 Rule
    interpretation_of_a_weak_equality_v1  : ∫∫ u_•v_ đx_ đy_ = 0 → ∫ u_•v_ đx_ = 0
                                            if v_.Type = "Test"
    interpretation_of_a_weak_equality_v2  : ∫ u_•v_ đx_ = 0 → u_ = 0
                                            if v_.Type = "Test"
    interpretation_of_a_weak_equality_v3  : 0 = ∫∫ u_•v_ đx_ đy_  → 0 = ∫ u_•v_ đx_
                                            if v_.Type = "Test"
```

```
   interpretation_of_a_weak_equality_v4  : 0 = ∫ u_•v_ đx_ → 0 = u_
                                           if v_.Type = "Test"
   interpretation_of_a_weak_equality_v5  : ∫∫ u_•v_ đx_ đy_ = 0 → ∫ u_•v_ đx_ = 0
                                           if u_.Type = "Test"
   interpretation_of_a_weak_equality_v6  : ∫ u_•v_ đx_ = 0 → v_ = 0
                                           if u_.Type = "Test"
   interpretation_of_a_weak_equality_v7  : 0 = ∫∫ u_•v_ đx_ đy_  → 0 = ∫ u_•v_ đx_
                                           if u_.Type = "Test"
   interpretation_of_a_weak_equality_v8  : 0 = ∫ u_•v_ đx_ → 0 = v_
                                           if u_.Type = "Test"
   interpretation_of_a_weak_equality_v9  : ∫∫ u_•v_ đx_ đy_ = ∫∫ h_•v_ đx_ đy_ →
                                           ∫ u_•v_ đx_ = ∫ h_•v_ đx_ if v_.Type="Test"
   interpretation_of_a_weak_equality_v10 : ∫ u_•v_ đx_ = ∫ h_•v_ đx_ → u_ = h_
                                           if v_.Type = "Test"

% ============================================================================
%        "adjoint_or_dual_of_TS"
% D83 :
% ============================================================================
Operator
   opT_expr : "T" [opTS_rw_.Index] [expr_] [opTS_rw_.Outputvar]
            [opTS_rw_.Inputvar(1),opTS_rw_.Inputvar(2)] [opTS_rw_.Parameter]
   % T(Expr)

   opTS_expr : "TS" [opT_rw_.Index] [expr_]
             [opT_rw_.Outputvar(1),opT_rw_.Outputvar(2)] [opT_rw_.Inputvar]
             [opT_rw_.Parameter]
   % TS(Expr)

Rule
   adjoint_or_dual_of_TS_v1 :
   kappa0•∫ expr_•opTS_rw_ đx_ →
   kappa1•∫∫ opT_expr•opTS_rw_.Expr đopTS_rw_.Inputvar(1) đopTS_rw_.Inputvar(2)

   adjoint_or_dual_of_TS_v2 :
   -(kappa0)•∫ expr_•opTS_rw_ đx_ →
   -(kappa1)•∫∫ opT_expr•opTS_rw_.Expr đopTS_rw_.Inputvar(1) đopTS_rw_.Inputvar(2)

   adjoint_or_dual_of_TS_v3 :
   kappa0•a_•∫ expr_•opTS_rw_ đx_ →
   kappa1•a_•∫∫ opT_expr•opTS_rw_.Expr đopTS_rw_.Inputvar(1) đopTS_rw_.Inputvar(2)

   adjoint_or_dual_of_TS_v4 :
   -kappa0•a_•∫ expr_•opTS_rw_ đx_ →
   -kappa1•a_•∫∫ opT_expr•opTS_rw_.Expr đopTS_rw_.Inputvar(1) đopTS_rw_.Inputvar(2)

   adjoint_or_dual_of_T :
```

```
      kappa1•∫∫ opT_rw_•expr_ đx_ đy_ →
      kappa0•∫ opT_rw_.Expr•opTS_expr đopT_rw_.Inputvar

% ============================================================================
%        "boundary_condition_of_Bv" : if v ∈ Cinf[Γ#](Ω#;Cinf(Ω1)) then
%                                      B(v) ∈ Cinf[Γ](Ωε)
% P85 :
% ============================================================================
Operator
   trace_opB_rw_ : "Trace" [trace_rw_Ind_] [opB_rw_] [trace_rw_InV_]
                   [trace_rw_OutV_] [trace_rw_Pa_]

Rule
   boundary_condition_of_Bv : trace_opB_rw_ → 0

% ============================================================================
%        "derivation_rule_for_B"
% P86 : ∂B(Expr(xs,x1))/∂x → B(∂Expr(xs,x1)/∂xs) + 1/eps•B(∂Expr(xs,x1)/∂x1)
% ============================================================================
Expression
   dB_Expr_dInVar_1 : ∂opB_rw_.Expr/∂opB_rw_.Inputvar(1)
   dB_Expr_dInVar_2 : ∂opB_rw_.Expr/∂opB_rw_.Inputvar(2)

Operator
   opB_dB_Expr_dInVar_1 : "B" [opB_rw_.Index] [dB_Expr_dInVar_1]
   [opB_rw_.Inputvar(1),opB_rw_.Inputvar(2)] [opB_rw_.Outputvar]
   [opB_rw_.Parameter]
   % B(∂B.Expr(xs,x1)/∂xs)

   opB_dB_Expr_dInVar_2 : "B" [opB_rw_.Index] [dB_Expr_dInVar_2]
   [opB_rw_.Inputvar(1),opB_rw_.Inputvar(2)] [opB_rw_.Outputvar]
   [opB_rw_.Parameter]
   % B(∂B.Expr(xs,x1)/∂x1)

Rule
   derivation_rule_for_B : ∂opB_rw_/∂x_ → opB_dB_Expr_dInVar_1
      + 1/opB_rw_.Parameter•opB_dB_Expr_dInVar_2   if x_ = opB_rw_.Outputvar
   % Expr.Var1 = xs, Expr.Var2 = x1

% ============================================================================
%        "approximation_between_B_and_TS"
% P87
% ============================================================================
Function
   oe_BPa  : "oe" [oe_rw_Ind_] [opB_rw_.Parameter] [] "Given"
   % O(opB.Parameter)
```

```
Expression
   x1_dBExpr_dxs : opB_rw_.Inputvar(2)•∂opB_rw_.Expr/∂opB_rw_.Inputvar(1)
   % x1.∂BExpr/∂xs

   x1_dTSExpr_dxs : opTS_rw_.Inputvar(2)•∂opTS_rw_.Expr/∂opTS_rw_.Inputvar(1)
   % x1.∂ExprTS/∂xs

Operator
   opTS_BExpr : "TS" [opB_rw_.Index] [opB_rw_.Expr]
   [opB_rw_.Inputvar(1),opB_rw_.Inputvar(2)] [opB_rw_.Outputvar]
   [opB_rw_.Parameter]
   % TS(B.Expr)

   opTS_x1_dBExpr_dxs : "TS" [opB_rw_.Index] [x1_dBExpr_dxs]
   [opB_rw_.Inputvar(1),opB_rw_.Inputvar(2)] [opB_rw_.Outputvar]
   [opB_rw_.Parameter]
   % TS(x1.∂Expr/∂xs)

   opB_TSExpr : "B" [opTS_rw_.Index] [opTS_rw_.Expr]
   [opTS_rw_.Inputvar(1),opTS_rw_.Inputvar(2)] [opTS_rw_.Outputvar]
   [opTS_rw_.Parameter]
   % B(TSExpr)

   opB_x1_dTSExpr_dxs : "B" [opTS_rw_.Index] [x1_dTSExpr_dxs]
   [opTS_rw_.Inputvar(1),opTS_rw_.Inputvar(2)] [opTS_rw_.Outputvar]
   [opTS_rw_.Parameter]
   % B(x1.∂ExprTS/∂xs)

Rule
   approximation_between_B_and_TS_l1 : opB_rw_ → opTS_BExpr
   + opB_rw_.Parameter•oe_eps
   approximation_between_B_and_TS : opB_rw_ → opTS_BExpr
   + opB_rw_.Parameter•opTS_x1_dBExpr_dxs + opB_rw_.Parameter•oe_eps
   approximation_between_TS_and_B : opTS_rw_ → opB_TSExpr
   - opTS_rw_.Parameter•opB_x1_dTSExpr_dxs + opTS_rw_.Parameter•oe_eps

% ================================================================================
%        "green_rule"
% P88 : ∫ ∂u/∂x•v đx → ∫ trace(u)•trace(v)•nΓ đxg - ∫ u(x)•∂v(x)/∂x đx
% ================================================================================
Variable
   xg : "xg" [xg_Ind_] x_.Region.Boundary

Operator
   trace_a : "Trace" [trace_Ind_] [a_] [x_] [xg] [trace_Pa_]
   trace_b : "Trace" [trace_Ind_] [b_] [x_] [xg] [trace_Pa_]
```

```
Rule
   green_rule : ∫ ∂a_/∂x_•b_ đx_ →  -∫ a_•∂b_/∂x_ đx_
       + ∫ trace_a•trace_b•x_.Region.Boundary.Normal đtrace_a.Outputvar

% ==============================================================================
%        "the_linear_operator_associated_to_the_microscopic_problem"
% P89 :
% ==============================================================================
Rule
   the_linear_operator_associated_to_the_microscopic_problem :
       ∫ a0•a_•∂w1/∂x1 đx1 = -mu_•∫ a0•∂w1/∂x1 đx1 → a_ = mu_•∂theta1/∂x1
   % P89

% ==============================================================================
%        "introduction_of_a_kronecker_symbol"
% P91 :
% ==============================================================================
Rule
   introduction_of_a_kronecker_symbol1 : ∂u_/∂x_ + ∂u_/∂x_•∂v_/∂y_ →
       (1 + ∂v_/∂y_)•∂u_/∂x_
   introduction_of_a_kronecker_symbol2 : ∂u_/∂x_ + ∂v_/∂y_•∂u_/∂x_ →
       (1 + ∂v_/∂y_)•∂u_/∂x_
   % P91

% ==============================================================================
%        "two_scale_limit_of_a_derivative"
% P97 :
% ==============================================================================
Rule
   two_scale_limit_of_a_derivative : u1-x1•∂u0/∂xs → ut1
   % P97

% ==============================================================================
%        "product_rule_of_opT"
% ==============================================================================
Expression
   expr1expr2 : expr1_•expr2_

Operator
   opT_Expr1Expr2_ : "T" [opT_Expr1Expr2_Ind_] [expr1expr2] [opT_Expr1Expr2_InV_]
   [opT_Expr1Expr2_OutV1_,opT_Expr1Expr2_OutV2_] [opT_Expr1Expr2_Pa_]

   opT_Expr1 : "T" [opT_Expr1Expr2_.Index] [expr1_] [opT_Expr1Expr2_.Inputvar]
   [opT_Expr1Expr2_.Outputvar(1),opT_Expr1Expr2_.Outputvar(2)]
   [opT_Expr1Expr2_.Parameter]

   opT_Expr2 : "T" [opT_Expr1Expr2_.Index] [expr2_] [opT_Expr1Expr2_.Inputvar]
```

```
   [opT_Expr1Expr2_.Outputvar(1),opT_Expr1Expr2_.Outputvar(2)]
   [opT_Expr1Expr2_.Parameter]

Rule
   product_rule_of_opT : opT_Expr1Expr2_ → opT_Expr1•opT_Expr2
   % P77 : T(a•b) → T(a)•T(b)

% ================================================================================
%        "simplify_opB" : change the order of integral
%        "linearity_opB" :
%        "inverse_linearity_opB" :
%        "take_const_out_of_opB" :
% ================================================================================
Operator
   opB_rw1_ : "B" [opB_rw1_Ind_] [opB_rw1_Fun_] [opB_rw1_InV1_,opB_rw1_InV2_]
       [opB_rw1_OutV_] [opB_rw1_Pa_]
   % B() : L2(Ω#xΩ1) → L2(Ωε)

Expression
   opB_opB1_Expr : opB_rw_.Expr + opB_rw1_.Expr
   linearity_opB_Expr : expr1_ + expr2_
   constExpr : const_•expr_
   expr1expr2 : expr1_•expr2_

Operator
   opB_opB1 : "B" [opB_rw_.Index] [opB_opB1_Expr]
   [opB_rw_.Inputvar(1),opB_rw_.Inputvar(2)] [opB_rw_.Outputvar]
   [opB_rw_.Parameter]

   opB_Expr1_plus_Expr2_ : "B" [opB_rw_Ind_] [linearity_opB_Expr]
   [opB_rw_Invar1_,opB_rw_Invar2_]
   [opB_rw_Outvar_]  [opB_rw_Pa_]

   opB_Expr1 : "B" [opB_Expr1_plus_Expr2_.Index] [expr1_]
   [opB_Expr1_plus_Expr2_.Inputvar(1),opB_Expr1_plus_Expr2_.Inputvar(2)]
   [opB_Expr1_plus_Expr2_.Outputvar] [opB_Expr1_plus_Expr2_.Parameter]

   opB_Expr2 : "B" [opB_Expr1_plus_Expr2_.Index] [expr2_]
   [opB_Expr1_plus_Expr2_.Inputvar(1),opB_Expr1_plus_Expr2_.Inputvar(2)]
   [opB_Expr1_plus_Expr2_.Outputvar] [opB_Expr1_plus_Expr2_.Parameter]

   opB_constExpr_ : "B" [opB_rw_Ind_] [constExpr]
   [opB_rw_Invar1_,opB_rw_Invar2_] [opB_rw_Outvar_]  [opB_rw_Pa_]

   opB_Expr : "B" [opB_constExpr_.Index] [expr_]
   [opB_constExpr_.Inputvar(1),opB_constExpr_.Inputvar(2)]
   [opB_constExpr_.Outputvar] [opB_constExpr_.Parameter]
```

```
Rule
   simplify_opB : opB_rw_ → 0 if opB_rw_.Expr = 0

   linearity_opB : opB_Expr1_plus_Expr2_ → opB_Expr1 + opB_Expr2
   % B(a+b) → B(a) + B(b)

   inverse_linearity_opB : opB_rw_ + opB_rw1_ → opB_opB1
   % B(a) + B(b) → B(a+b)

   take_const_out_of_opB : opB_constExpr_ → const_•opB_Expr
       if theta(const_) = ∅
   % B(const•expr) → const•B(expr)

% ================================================================================
%        "simplify_trace" gives value of its functions on the boundary
% ================================================================================
Operator
   trace_Expr1_ : "Trace" [trace_Ind_] [expr1_] [trace_Expr1_Invar_]
       [trace_Expr1_Outvar_] [trace_Pa_]
   trace_Expr2_ : "Trace" [trace_Ind_] [expr2_] [trace_Expr2_Invar_]
       [trace_Expr1_Outvar_] [trace_Pa_]

Rule
   simplify_trace_v1 : ∫ trace_Expr1_•trace_Expr2_•n_ đx_ →
       ∫ trace_Expr1_.Expr.BCLhsExpr•trace_Expr2_•n_ đx_

% ================================================================================
%        "fubini_theorem" change the order of integral
% ================================================================================
Rule
   fubini_theorem : ∫∫ a_ đx_ đy_ → ∫∫ a_ đy_ đx_

% ================================================================================
%        "adjoint_of_d"
% ================================================================================
Rule
   adjoint_of_d : ∫∫ x1•u_•∂v_/∂xs_ đx_ đy_ → -∫∫ x1•v_•∂u_/∂xs_ đx_ đy_
   % D90 : ∫∫ u(xs,x1)•x1•∂v(xs,x1)/∂xs đxs đx1 →
     -∫∫v(xs,x1)•x1•∂u(xs,x1)/∂xs đxs đx1

Strategy
   interpretation_of_a_weak_equality :   interpretation_of_a_weak_equality_v1
   | interpretation_of_a_weak_equality_v2
   | interpretation_of_a_weak_equality_v3
   | interpretation_of_a_weak_equality_v4
   | interpretation_of_a_weak_equality_v5
```

```
| interpretation_of_a_weak_equality_v6
| interpretation_of_a_weak_equality_v7
| interpretation_of_a_weak_equality_v8
| interpretation_of_a_weak_equality_v9
| interpretation_of_a_weak_equality_v10

adjoint_or_dual_of_TS :   adjoint_or_dual_of_TS_v1
| adjoint_or_dual_of_TS_v2
| adjoint_or_dual_of_TS_v3
| adjoint_or_dual_of_TS_v4

green_rule_strategy :   green_rule
| fubini_theorem

introduction_of_a_kronecker_symbol :   introduction_of_a_kronecker_symbol1
| introduction_of_a_kronecker_symbol2
```

7.3 First Block

The code includes the source term and the proof itself corresponding to Lemma 59 in Section 6.1.

First block source term:

```
PDE "l98_source_term"
Constant
    psi           : "psi"
    gammae_NorVec : "gammae_NorVec"
    gamma1_NorVec : "gamma1_NorVec"
    gammas_NorVec : "gammas_NorVec"
    omegae_NorVec : "omegae_NorVec"
    omega1_NorVec : "omega1_NorVec"
    omegas_NorVec : "omegas_NorVec"
    norvec : "norvec"

Region
    gammae : "gammae" [gamma_Ind_] [gamma_Dim_] [] gammae_Bou_ gammae_NorVec
    % Γε

    gamma1 : "gamma1" [gamma1_Ind_] [gamma1_Dim_] [] gamma1_Bou_ gamma1_NorVec
    % Γ1

    gammas : "gammas" [gammas_Ind_] [gammas_Dim_] [] gammas_Bou_ gammas_NorVec
    % Γ#

    omegae : "omegae" [omegae_Ind_] [1] [] gammae omegae_NorVec
    % Ωε(Γε)
```

```
    omega1 : "omega1" [omega1_Ind_] [1] [] gamma1 omega1_NorVec
    % Ω1(Γ1)

    omegas : "omegas" [omegas_Ind_] [1] [] gammas omegas_NorVec
    % Ω#(Γ#)

    gammav : "gammav" [gammav_Ind_] [1] [gammas,gamma1] gammav_Bou_ gammav_NorVec_
    % Γv = Γ#∪Γ1

    eps_reg : "eps_reg" [eps_reg_Ind_] [1] [0,1] eps_reg_Bou_ eps_reg_NorVec_

Expression % use in source term
    o1xos  : omega1●omegas
    % Ω1xΩ# !!!

Function % use in source term
    meas_rw_ : "Measure" [meas_Ind_] [meas_Var_] [] "Given"
    % Measure function
    meas_omegae : "Measure" [meas_Ind_] [omegae] [] "Given"
    % |Ωε| !!! meas_Var =
    meas_o1xos : "Measure" [meas_Ind_] [o1xos] [] "Given"
    % |Ω1xΩ#|

Expression % use in source term
    kappa0 : 1/meas_omegae
    kappa1 : 1/meas_o1xos
    null : 0

Variable
    x  : "x" []  omegae
    x1 : "x1" [] omega1
    xs : "xs" [] omegas
    eps : "eps" [] eps_reg

Function
    ue : "ue" [] [x] [(gammae ue_ null)] "Unknown"
    % uε : Ωε, uε = 0

    v : "v" [] [xs,x1] [(gammas v_ null),(gamma1 v_ null)] "Test"
    % v ∈ CΓ#(Ω#,CΓ1(Ω1)), v = 0 on Γ#, v = 0 on Γ1, Test function

    ae : "ae" [] [x] [] "Unknown"

    f : "f" [] [x] [] "Unknown"

Expression % USE IN SOURCE TERM
    dv_dx1 : ∂v/∂x1
```

```
        due_dx : ∂ue/∂x

    Operator % USE IN SOURCE TERM
        % opB_v : "B" [opB_Ind_] [v] [xs,x1] [x] [eps]
        % B(v)

        opB_v_rw : "B" [opB_Ind_] [v_] [v_.Variable(1),v_.Variable(2)] [ue_.Variable] [eps]

        opT_ue : "T" [opT_Ind_] [ue] [x] [xs,x1] [eps]
        % T(ue)

    Expression % USE TO TEST
        x1_ddv_dx1_dxs : x1•∂(∂v/∂x1)/∂xs

    Operator % USE TO TEST
        opB_dv_dx1 : "B" [opB_Ind_] [dv_dx1] [xs,x1] [x] [eps]
        opTS_dv_dx1 : "TS" [opB_Ind_] [dv_dx1] [xs,x1] [x] [eps]
        opTS_x1_ddv_dx1_dxs : "TS" [opB_Ind_] [x1_ddv_dx1_dxs] [xs,x1] [x] [eps]

    PDE
        l98_source_term : kappa0•∫ ae•∂ue/∂x•∂v/∂x đx = kappa0•∫ f•v đx
```

First block in the reference proof:

```
Model "l98ref" % Lemma 98, First Block
Function
   oe_eps : "oe" [oe_Ind_] [eps] [] "Given"

   oe_rw_ : "oe" [oe_Ind_]  [oe_Var_] [] "Given"
   % O() tend to zero as ε->0 !

   u0 : "u0" [u0_Ind_] [xs] [] "Unknown"

   u1 : "u1" [u1_Ind_] [xs,x1] [] "Unknown"

   ut1 : "ut1" [ut1_Ind_] [xs,x1] [] "Unknown"

   a0 : "a0" [a0_Ind_] [x] [] "Unknown"

   w1 : "w1" [w1_Ind_] [x1] [] "Unknown"
   % w1 ∈ C#(Ω1)

   theta1 : "theta1" [theta1_Ind_] [x1] [] "Unknown"

   v1 : "v1" [v1_Ind_] [xs,x1] [(gammas v1_ null)] "Test"
   % v1 ∈ CΓ#(Ω#,CΓ#(Ω1)), Test function
```

```
   #Include "basic_math_rule.proof"
   #Include "math_rule.proof"

   Rule
   mult_equality_by_eps : a_ = b_ → eps•a_ = eps•b_

   approximation_of_Tu  : opT_ue → u0 + oe_eps
   % Assumption  : eq47

   create_source_term : kappa0•∫ ae_•∂ue_/∂x_•∂v_/∂x_ đx_ = k_→
   psi = kappa0•∫∂ue_/∂x_•opB_v_rw đx_

   temporary_1 : ∫ a_•b_•c_ đx_ → 0

Step
   step_cst : create_source_term ↑  % Create Source Term (cst) % Correct Source Term
   step0   : mult_equality_by_eps ↑
   step1   : green_rule ↑
      ; boundary_condition_of_Bv ↑
      ; simplify_math ↑
   step2   : derivation_rule_for_B ↑
      ; expansion ↑
      ; integral_linearity ↑
      ; expansion ↑
      ; take_constant_out_of_integral1 ↑
      ; minus_to_minus_1 ↑
      ; simplify_math ↑
      ; mult_with_1 ↑
      ; eps_to_oe ↑
      ; simplify_multi_scale ↑
   step3   : approximation_between_B_and_TS_l1 ↑
      ; simplify_multi_scale ↑
      ; remove_oe_from_addition ↑
   step4   : adjoint_or_dual_of_TS_v1 ↑
   step5   : approximation_of_Tu ↑
      ; remove_oe_from_addition ↑
   step6   : fubini_theorem ↑
      ; green_rule ↑
      ; temporary_1 ↑
      ; simplify_math ↑
      ; oe_pass_eps_to_0_rule ↑
      ; remove_constant_from_equation_equal_0 ↑
      ; mult_with_1 ↓ ; mult_with_1 ↓
      ; simplify_sign ↑
   step7   : interpretation_of_a_weak_equality ↑
      ; interpretation_of_a_weak_equality ↑
Model l98ref : step_cst; step0; step1; step2; step3; step4; sep5;step6; step7
```

7.4 Second Block

The code includes the source term and the proof itself corresponding to Lemma 60 in Section 6.1.

Second block source term:

```
PDE "l99_source_term"
Constant
    psi           : "psi"
    gammae_NorVec : "gammae_NorVec"
    gamma1_NorVec : "gamma1_NorVec"
    gammas_NorVec : "gammas_NorVec"
    omegae_NorVec : "omegae_NorVec"
    omega1_NorVec : "omega1_NorVec"
    omegas_NorVec : "omegas_NorVec"
    norvec : "norvec"

Region
    gammae : "gammae" [gamma_Ind_] [gamma_Dim_] [] gammae_Bou_ gammae_NorVec
    % Γε

    gamma1 : "gamma1" [gamma1_Ind_] [gamma1_Dim_] [] gamma1_Bou_ gamma1_NorVec
    % Γ1

    gammas : "gammas" [gammas_Ind_] [gammas_Dim_] [] gammas_Bou_ gammas_NorVec
    % Γ#

    omegae : "omegae" [omegae_Ind_] [1] [] gammae omegae_NorVec
    % Ωε(Γε)

    omega1 : "omega1" [omega1_Ind_] [1] [] gamma1 omega1_NorVec
    % Ω1(Γ1)

    omegas : "omegas" [omegas_Ind_] [1] [] gammas omegas_NorVec
    % Ω#(Γ#)

    gammav : "gammav" [gammav_Ind_] [1] [gammas,gamma1] gammav_Bou_ gammav_NorVec_
    % Γv = Γ#∪Γ1

    eps_reg : "eps_reg" [eps_reg_Ind_] [1] [0,1] eps_reg_Bou_ eps_reg_NorVec_

Expression % use in source term
    o1xos  : omega1•omegas

Function % use in source term
    meas_rw_ : "Measure" [meas_Ind_] [meas_Var_] [] "Given"
    % Measure function
    meas_omegae : "Measure" [meas_Ind_] [omegae] [] "Given"
```

```
        % |Ωε| !!! meas_Var =
        meas_o1xos : "Measure" [meas_Ind_] [o1xos] [] "Given"
        % |Ω1xΩ#|

    Expression % use in source term
        kappa0 : 1/meas_omegae
        kappa1 : 1/meas_o1xos
        null : 0

    Variable
        x  : "x" [x_Ind_]  omegae
        x1 : "x1" [x1_Ind_] omega1
        xs : "xs" [xs_Ind_] omegas
        xg : "xg" [xg_Ind_] gammae
        eps : "eps" [eps_Ind_] eps_reg

    Function
        ue : "ue" [ue_Ind_] [x] [(gammae ue_ null)] "Unknown"
        % uε : Ωε, uε = 0

        v : "v" [v_Ind_] [xs,x1] [(gammav v_ null)] "Test"
        % v ∈ CΓ#(Ω#,CΓ1(Ω1)), v = 0 on Γ#, v = 0 on Γ1, Test function

        ae : "ae" [ae_Ind_] [x] [] "Unknown"

        f : "f" [f_Ind_] [x] [] "Unknown"

    Expression % USE IN SOURCE TERM
        dv_dx1 : ∂v/∂x1

    PDE
    l99_source_term : kappa0•∫ ae•∂ue/∂x•∂v/∂x đx = kappa0•∫ f•v đx
```

Second block in the reference proof:

```
Model "l99ref"
Function
    oe_eps : "oe" [oe_Ind_] [eps] [] "Given"

    oe_rw  : "oe" [oe_Ind_]  [oe_Var_] [] "Given"

    u0 : "u0" [u0_Ind_] [xs] [] "Unknown"

    u1 : "u1" [u1_Ind_] [xs,x1] [] "Unknown"

    v1 : "v1" [v1_Ind_] [xs,x1] [(gammas v1_ null)] "Test"
    % v1 ∈ CΓ#(Ω#,CΓ#(Ω1)), Test function
```

```
    ut1 : "ut1" [ut1_Ind_] [xs,x1] [] "Unknown"
    % u~1

    a0 : "a0" [a0_Ind_] [x] [] "Unknown"

    w1 : "w1" [w1_Ind_] [x1] [] "Unknown"
    % w1 ∈ C#(Ω1)

    eta : "eta" [eta_Ind_] [xs,x1] [] "Unknown"

    theta1 : "theta1" [theta1_Ind_] [x1] [] "Unknown"

#Include "basic_math_rule.proof"
#Include "math_rule.proof"

Operator
    opT_ue : "T" [opT_Ind_] [ue] [x] [xs,x1] [eps]
    % T(ue)

    trace_ue : "Trace" [trace_Ind_] [ue] [x] [xg_rw_] [trace_Pa_]
    % trace(ue)

Rule
    ue_on_gamma : trace_ue → 0
    % ue = 0 on Γ check

    approximation_of_Tu  : opT_ue → u0 + oe_eps
    % Assumption  : eq47

    assumption_L99 : opT_ue → u0 + eps•u1
    % T(ue) → u0 + eps•u1

    two_scale_limit_of_a_derivative : u1-x1•∂u0/∂xs → ut1

    special1 : kappa1•e_ + (-kappa1)•d_ + (-kappa1)•a_•c_ + (-kappa1)•b_ + oe_eps →
    kappa1•(e_ - d_ - a_•c_ - b_) + oe_eps

    special2 : kappa1•-1•a_ + kappa1•-1•b_•c_ + kappa1•-1•∫∫ d_•∂v/∂x1 đx_ đy_
    + kappa1•∫∫ e_•∂v/∂x1 đx_ đy_ + oe_eps → -1•kappa1•a_ + -1•kappa1•b_•c_
    + -1•kappa1•∫∫(d_-e_)•∂v/∂x1 đx_ đy_ + oe_eps

    special3 : kappa1•∫∫ k_•a_ đx_ đy_ + kappa1•b_•c_ + kappa1•∫∫ k_•d_ đx_ đy_
     + oe_eps → kappa1•∫∫( a_ + d_)•k_ đx_ đy_ + kappa1•b_•c_ + oe_eps

% ==========================================================================
%            CREATE SOURCE TERM FROM WEAK FORM (PDE)
```

```
% ===============================================================================
Expression
    due_dx_rw : ∂ue_/∂x_
Operator
    opT_due_dx_rw : "T" [opT_Ind_] [due_dx_rw] [x_]
    [v_.Variable(1),v_.Variable(2)] [eps]
    % T(∂ue/∂x)
Rule
    source_term : kappa0•∫ ae_•∂ue_/∂x_•∂v_/∂x_ đx_ = k_→
    psi = kappa1•∫∫opT_due_dx_rw•v_ đv_.Variable(1) đv_.Variable(2)

Step
    step_sc : source_term ↑
    step1   : adjoint_or_dual_of_T ↑
    step2   : approximation_between_TS_and_B ↑
    ; eps_to_oe ↑
    ; expansion ↓
    ; simplify_multi_scale ↑
    ; integral_linearity ↑
    ; expansion ↓
    ; simplify_multi_scale ↑
    step3   : green_rule ↑
    ; simplify_math ↑
    step4   : derivation_rule_for_B ↑
    ; expansion ↓
    ; integral_linearity ↑
    ; take_constant_out_of_integral ↑
    ; expansion ↑
    step5   : approximation_between_B_and_TS ↑
    ; expansion ↓
    ; integral_linearity ↓
    ; expansion ↓
    ; take_constant_out_of_integral1 ↑
    ; simplify_math ↑
    ; eps_to_oe ↑
    ; expan_sign ↓
    ; simplify_multi_scale ↑
    ; minus_to_minus_1 ↑
        step6   : adjoint_or_dual_of_TS3 ↑
    step7   : assumption_L99 ↑
    ; expansion ↓
    ; integral_linearity ↑
    ; take_constant_out_of_integral ↑
    ; expansion ↓
    ; simplify_math ↑
    ; eps_to_oe ↑
    ; simplify_multi_scale ↑
```

```
    ; simplify_multi_scale ↑
     step8   : adjoint_of_d ↑
    ; minus_to_minus_1 ↑
    ; simplify_sign ↓
     step9   : special2 ↑
     step10  : green_rule ↑
    ; simplify_math ↑
    ; simplify_sign ↑
    ; simplify_sign ↑
    ; fubini_theorem ↑
     step11  : green_rule ↑
    ; simplify_math ↑
    ; simplify_sign ↑
    ; simplify_sign ↑
     step12   : special3 ↑
     step13  : two_scale_limit_of_a_derivative ↑
    ; simplify_derivative2 ↑
    ; simplify_math ↑
    ; oe_pass_eps_to_0_rule ↑
     step14  :% weak_limit_of_T ↑
    simplify_math ↑
     ; simplify_equation ↑
     ; interpretation_of_a_weak_equality ↑
     ; interpretation_of_a_weak_equality ↑
Model
     l99ref :      step_sc; step1; step2; step3; step4
  ; step5; step6; step7; step8; step9; step10; step11
  ; step12; step13; step14
```

7.5 Third Block

The code includes the source term and the proof itself corresponding to Lemma 61 in Section 6.1.

Third block source term:

```
    PDE "l100_source_term"
    Constant
        eps  : "eps"
        psi           : "psi"
        gammae_NorVec : "gammae_NorVec"
        gamma1_NorVec : "gamma1_NorVec"
        gammas_NorVec : "gammas_NorVec"
        omegae_NorVec : "omegae_NorVec"
        omega1_NorVec : "omega1_NorVec"
        omegas_NorVec : "omegas_NorVec"
        norvec : "norvec"
```

```
Region
    gammae : "gammae" [] [] [] gammae_Bou_ gammae_NorVec
    % Γε

    gamma1 : "gamma1" [] [] [] gamma1_Bou_ gamma1_NorVec
    % Γ1

    gammas : "gammas" [] [] [] gammas_Bou_ gammas_NorVec
    % Γ#

    omegae : "omegae" [omegae_Ind_] [1] [] gammae omegae_NorVec
    % Ωε(Γε)

    omega1 : "omega1" [omega1_Ind_] [1] [] gamma1 omega1_NorVec
    % Ω1(Γ1)

    omegas : "omegas" [omegas_Ind_] [1] [] gammas omegas_NorVec
    % Ω#(Γ#)

    gammav : "gammav" [gammav_Ind_] [1] [gammas,gamma1] gammav_Bou_ gammav_NorVec_
    % Γv = Γ#∪Γ1

    eps_reg : "eps_reg" [eps_reg_Ind_] [1] [0,1] eps_reg_Bou_ eps_reg_NorVec_

Expression % use in source term
    o1xos  : omega1•omegas

Function % use in source term
    meas_rw_ : "Measure" [meas_Ind_] [meas_Var_] [] "Given"
    % Measure function
    meas_omegae : "Measure" [meas_Ind_] [omegae] [] "Given"
    % |Ωε|
    meas_o1xos : "Measure" [meas_Ind_] [o1xos] [] "Given"
    % |Ω1xΩ#|

Expression % use in source term
    kappa0 : 1/meas_omegae
    kappa1 : 1/meas_o1xos
    null : 0

Variable
    x  : "x" [x_Ind_]  omegae
    x1 : "x1" [x1_Ind_] omega1
    xs : "xs" [xs_Ind_] omegas
    xg : "xg"[xg_Ind_] gammae
    eps : "eps" [eps_Ind_] eps_reg
```

```
Function
    ue : "ue" [ue_Ind_] [x] [(gammae ue_ null)] "Unknown"
    % uε : Ωε, uε = 0

    v : "v" [v_Ind_] [xs,x1] [(gammas v_ null)] "Test"
    % v ∈ CΓ#(Ω#,C#(Ω1)), v = 0 on Γ#, Test function

    ae : "ae" [ae_Ind_] [x] [] "Unknown"

    f : "f" [f_Ind_] [x] [] "Unknown"

    u0 : "u0" [u0_Ind_] [xs] [] "Unknown"

    Expression % USE IN SOURCE TERM
    dv_dx1 : ∂v/∂x1

PDE
l100_source_term : kappa0•∫ ae•∂ue/∂x•∂v/∂x đx = kappa0•∫ f•v đx
```

Third block in the reference proof:

```
Model "l100ref"
Function
    oe_eps : "oe" [oe_Ind_] [eps] [] "Bigo"

    oe_rw_ : "oe" [oe_Ind_]  [oe_Var_] [] "Bigo"
    % O() tend to zero as ε->0 !

    v1 : "v1" [v1_Ind_] [xs,x1] [(gammas v1_ null)] "Test"
    % v1 ∈ CΓ#(Ω#,CΓ#(Ω1)), Test function

    ut1 : "ut1" [ut1_Ind_] [xs,x1] [] "Unknown"
    % u~1

    a0 : "a0" [a0_Ind_] [x] [] "Unknown"

    w1 : "w1" [w1_Ind_] [x1] [] "Unknown"
    % w1 ∈ C#(Ω1)

    eta : "eta" [eta_Ind_] [xs,x1] [] "Unknown"

    theta1 : "theta1" [theta1_Ind_] [x1] [] "Unknown"

    u1 : "u1" [u1_Ind_] [xs,x1] [] "Unknown"

#Include "basic_math_rule.proof"
#Include "math_rule.proof"
```

```
% ================================================================================
%           RULES USED IN STEP
% ================================================================================
Operator
    opB_v : "B" [opB_Ind_] [v] [xs,x1] [x] [eps]
    % B(v)

    opT_ue : "T" [opT_Ind_] [ue] [x] [xs,x1] [eps]
    % T(ue)

    trace_ue : "Trace" [trace_Ind_] [ue] [x] [xg_rw_] [trace_Pa_]
    % trace(ue)

    trace_v : "Trace" [trace_Ind_] [v] [x1] [gamma1] [trace_Pa_]
    % trace(v)

    opB_dv_dx1 : "B" [opB_Ind_] [dv_dx1] [xs,x1] [x] [eps]
    % B(∂v/∂x1)

    opTS_dv_dx1 : "TS" [opTS_Ind_] [dv_dx1] [xs,x1] [x] [eps]
    % TS(∂v/∂x1)

    trace_u0 : "Trace" [trace_Ind_] [u0] [x1] [gamma1] [trace_Pa_]
    % trace(ue)

Rule
    ue_on_gamma : trace_ue → 0
    % ue = 0 on Γ check

    approximation_of_Tu  : opT_ue → u0 + oe_eps
    % Assumption  : eq47

    v_on_gamma1 : trace_v → 0
    % v = 0 on Γ1

    assumption_L99 : opT_ue → u0 + eps•u1
    % T(ue) → u0 + eps•u1

    two_scale_limit_of_a_derivative : u1-x1•∂u0/∂xs → ut1

    test : a_ - a_ + b_ → b_

    special1 : kappa1•e_ + (-kappa1)•d_ + (-kappa1)•a_•c_
    + (-kappa1)•b_ + oe_eps → kappa1•(e_ - d_ - a_•c_ - b_) + oe_eps

    special2 : (-kappa1)•a_ + (-kappa1)•b_•c_ + (-kappa1)•∫∫ d_•∂v/∂x1 đx_ đy_
```

```
 + kappa1•∫∫ ∂v/∂x1•e_ đx_ đy_ + oe_eps → (-kappa1)•a_ + (-kappa1)•b_•c_
 + (-kappa1)•∫∫(d_ - e_)•∂v/∂x1 đx_ đy_ + oe_eps

special3 : kappa1•∫∫ k_•a_ đx_ đy_ + kappa1•b_•c_ + kappa1•∫∫ k_•d_ đx_ đy_
+ oe_eps → kappa1•∫∫( a_ + d_)•k_ đx_ đy_ + kappa1•b_•c_ + oe_eps

v_periodic_on_gamma1 : ∫trace_v•trace_u0•gamma1_NorVec đgamma1 → 0

result_of_l99 : eta → ∂u0/∂xs + ∂ut1/∂x1

% ================================================================================
%           CREATE SOURCE TERM FROM WEAK FORM (PDE)
% ================================================================================
    Rule
        source_term : kappa0•∫ ae_•∂ue_/∂x_•∂v_/∂x_ đx_ = k_→ psi = 1

    Step
        step_sc : source_term ↑
        step1   : adjoint_or_dual_of_T ↑
        step2   : approximation_between_TS_and_B ↑
        ; eps_to_oe ↑
        ; expansion ↓
        ; simplify_multi_scale ↑
        ; integral_linearity ↑
        ; expansion ↓
        ; simplify_multi_scale ↑
        step3   : green_rule ↑
        ; simplify_math ↑
        step4   : derivation_rule_for_B ↑
        ; expansion ↓
        ; integral_linearity ↑
        ; take_constant_out_of_integral ↑
        ; expansion ↑
        step5   : approximation_between_B_and_TS ↑
        ; expansion ↓
        ; integral_linearity ↓
        ; expansion ↓
        ; take_constant_out_of_integral ↑
        ; simplify_math ↑
        ; eps_to_oe ↑
        ; simplify_sign ↓
        ; simplify_multi_scale ↑
        ; mult_with_1 ↑
        ; pretty_integral ↑
        step6   : adjoint_or_dual_of_TS ↑
        step7   : assumption_L99 ↑
        ; expansion ↓
```

```
    ; integral_linearity ↑
    ; take_constant_out_of_integral ↑
    ; expansion ↓
    ; simplify_math ↑
    ; eps_to_oe ↑
    ; simplify_sign ↓
    ; simplify_multi_scale ↑
    ; simplify_multi_scale ↑
    ; pretty_integral ↑
    step8   : adjoint_of_d ↑
    ; simplify_sign ↓
    step9   : special2 ↑
    step10  : green_rule ↑
    ; simplify_math ↑
    ; simplify_sign ↑
    ; simplify_sign ↑
    ; fubini_theorem ↑
    step11  :  green_rule ↑
    ; simplify_sign ↑
    ; simplify_sign ↑
    ; v_periodic_on_gamma1 ↑
    ; simplify_math ↑
    ; integral_linearity ↑
    ; expansion ↑
    ; simplify_sign ↑
    ; simplify_sign ↑
    step12  :  two_scale_limit_of_a_derivative ↑
    ; simplify_derivative2 ↑
    ; simplify_math ↑
    ; oe_pass_eps_to_0_rule ↑
    ; simplify_math ↑
    step13  :  result_of_l99 ↑
    ; expansion ↑
    ; integral_linearity ↑
    ; expansion ↑
    ; change_side_left ↑
    ; expan_sign ↑
    ; expan_sign ↑
    ; simplify_math ↑
    ; simplify_math ↑
    ; simplify_sign ↑
    ; simplify_sign ↑

Model l100ref : step_sc; step1; step2; step3; step4; step5
    ; step6; step7; step8; step9; step10; step11; step12; step13
```

7.6 Fourth Block

The code includes the source term and the proof itself corresponding to Lemma 62 in Section 6.1.

Fourth block source term:

```
PDE "l101_source_term"
Constant
   psi           : "psi"
   gammae_NorVec : "gammae_NorVec"
   gamma1_NorVec : "gamma1_NorVec"
   gammas_NorVec : "gammas_NorVec"
   omegae_NorVec : "omegae_NorVec"
   omega1_NorVec : "omega1_NorVec"
   omegas_NorVec : "omegas_NorVec"
   norvec : "norvec"

Region
   gammae : "gammae" [gamma_Ind_] [gamma_Dim_] [] gammae_Bou_ gammae_NorVec
   % Γε

   gamma1 : "gamma1" [gamma1_Ind_] [gamma1_Dim_] [] gamma1_Bou_ gamma1_NorVec
   % Γ1

   gammas : "gammas" [gammas_Ind_] [gammas_Dim_] [] gammas_Bou_ gammas_NorVec
   % Γ#

   omegae : "omegae" [omegae_Ind_] [1] [] gammae omegae_NorVec
   % Ωε(Γε)

   omega1 : "omega1" [omega1_Ind_] [1] [] gamma1 omega1_NorVec
   % Ω1(Γ1)

   omegas : "omegas" [omegas_Ind_] [1] [] gammas omegas_NorVec
   % Ω#(Γ#)

   gammav : "gammav" [gammav_Ind_] [1] [gammas,gamma1] gammav_Bou_ gammav_NorVec_
   % Γv = Γ#∪Γ1

   eps_reg : "eps_reg" [eps_reg_Ind_] [1] [0,1] eps_reg_Bou_ eps_reg_NorVec_

Expression % use in source term
   o1xos  : omega1•omegas

Function % use in source term
   meas_rw_ : "Measure" [meas_Ind_] [meas_Var_] [] "Given"
   % Measure function
   meas_omegae : "Measure" [meas_Ind_] [omegae] [] "Given"
```

```
   % |Ωε| !!! meas_Var =
   meas_o1xos : "Measure" [meas_Ind_] [o1xos] [] "Given"
   % |Ω1xΩ#|

Expression % use in source term
   kappa0 : 1/meas_omegae
   kappa1 : 1/meas_o1xos
   null : 0

Variable
   x  : "x" [x_Ind_]  omegae
   x1 : "x1" [x1_Ind_] omega1
   xs : "xs" [xs_Ind_] omegas
   xg : "xg"[xg_Ind_] gammae
   eps : "eps" [eps_Ind_] eps_reg

Function
   ue : "ue" [ue_Ind_] [x] [(gammae ue_ null)] "Unknown"
   % uε : Ωε, uε = 0

   v : "v" [v_Ind_] [xs] [] "Test"
   % v ∈ C(Ω#), Test function

Expression % USE IN SOURCE TERM
   dv_dx1 : ∂v/∂x1
   due_dx : ∂ue/∂x

Operator % USE IN SOURCE TERM
   opB_v : "B" [opB_Ind_] [v] [xs,x1] [x] [eps]
   % B(v)

   opT_ue : "T" [opT_Ind_] [ue] [x] [xs,x1] [eps]
   % T(ue)

   trace_ue : "Trace" [trace_Ind_] [ue] [x] [xg_rw_] [trace_Pa_]
   % trace(ue)

   trace_v : "Trace" [trace_Ind_] [v] [trace_invar_] [xg_rw_] [trace_Pa_]
   % trace(v)

   opB_dv_dx1 : "B" [opB_Ind_] [dv_dx1] [xs,x1] [x] [eps]
   % B(∂v/∂x1)

   opT_due_dx : "T" [opT_Ind_] [due_dx] [x] [xs,x1] [eps]
   % T(∂ue/∂x)

   opTS_dv_dx1 : "TS" [opTS_Ind_] [dv_dx1] [xs,x1] [x] [eps]
```

```
   % TS(∂v/∂x1)

Operator % USE TO TEST
   opB_dv_dx1 : "B" [opB_Ind_] [dv_dx1] [xs,x1] [x] [eps]
   opTS_dv_dx1 : "TS" [opB_Ind_] [dv_dx1] [xs,x1] [x] [eps]

PDE
   l101_source_term : psi = kappa1•∫∫opT_due_dx•v đx1 đxs  % WORKING SOURCE TERM
```

The Fourth

Fourth block in the reference proof:

```
Model "l101ref"
Function
    oe_eps : "oe" [oe_Ind_] [eps] [] "Given"

    oe_rw_ : "oe" [oe_Ind_]  [oe_Var_] [] "Given"

    u0 : "u0" [u0_Ind_] [xs] [] "Unknown"

    u1 : "u1" [u1_Ind_] [xs,x1] [] "Unknown"

    v1 : "v1" [v1_Ind_] [xs,x1] [(gammas v1_ null)] "Test"
    % v1 ∈ CΓ#(Ω#,CΓ#(Ω1)), Test function

    ut1 : "ut1" [ut1_Ind_] [xs,x1] [] "Unknown"
    % u~1

    eta : "eta" [eta_Ind_] [xs,x1] [] "Unknown"

    a0 : "a0" [a0_Ind_] [x] [] "Unknown"

    w1 : "w1" [w1_Ind_] [x1] [] "Unknown"
    % w1 ∈ C#(Ω1)

    phi : "phi" [phi_Ind_] [xs] [] "Test"
    % w1 ∈ C(Ω#)

    theta1 : "theta1" [theta1_Ind_] [x1] [] "Unknown"

#Include "basic_math_rule.proof"
#Include "math_rule.proof"

Rule
    ue_on_gamma : trace_ue → 0
    % ue = 0 on Γ check
```

```
    approximation_of_Tu  : opT_ue → u0 + oe_eps
    % Assumption  : eq47

    chose_v_on_gammas : trace_v → 0
    % v = 0 on Γs

    assumption_L99 : opT_ue → u0 + eps•u1
    % T(ue) → u0 + eps•u1

    two_scale_limit_of_a_derivative : u1-x1•∂u0/∂xs → ut1

    substitue_psi : psi → kappa1•∫∫opT_due_dx•v đx1 đxs

    psi_pass_eps_to_0 : kappa1•∫∫opT_due_dx•v đx1 đxs →
     kappa1•∫∫eta•v đx1 đxs

    test : a_ + (b_ + c_) → a_ + b_ + c_

    pretty_oe_eps : a_ + oe_eps → a_ - oe_eps

    make_clear_intergral1 : ∫∫ a_•b_ đx_ đy_ → ∫(∫ a_ đx_)•b_ đy_
     if a_ = eta

    make_clear_intergral2 : ∫∫ a_•b_ đx_ đy_ → ∫1 đy_•∫ a_•b_ đx_
     if a_ = v

    special1 : kappa1•∫a_•v đxs_ - kappa1•c_•∫b_•v đxs_ = 0
     → kappa1•∫(a_ - b_)•v đxs_ = 0

Step
    step1   : adjoint_or_dual_of_T ↑
    step2   : approximation_between_TS_and_B ↑
    ; eps_to_oe ↑
    ; expansion ↓
    ; simplify_multi_scale ↑
    ; integral_linearity ↑
    ; expansion ↓
    ; simplify_multi_scale ↑
    step3   : green_rule ↑
    ; simplify_trace ↑
    ; simplify_math ↑
    step4   : derivation_rule_for_B ↑
    ; simplify_derivative2 ↑
    ; simplify_opB ↑
    ; simplify_math ↑
    step5   : approximation_between_B_and_TS ↑
    ; expansion ↓
```

```
    ; integral_linearity ↓
    ; expansion ↓
    ; take_constant_out_of_integral ↑
    ; simplify_math ↑
    ; eps_to_oe ↑
    ; simplify_sign ↓
    ; simplify_multi_scale ↑
    step6   : adjoint_or_dual_of_TS ↑
    step7   : assumption_L99 ↑
    ; expansion ↓
    ; integral_linearity ↑
    ; take_constant_out_of_integral ↑
    ; expansion ↓
    ; simplify_math ↑
    ; eps_to_oe ↑
    ; simplify_multi_scale ↑
    step8   : green_rule ↑
    ; integral_linearity ↓
    ; expansion ↑
    ; simplify_sign ↑
    ; simplify_sign ↑
    step9   : substitue_psi ↑
    ; psi_pass_eps_to_0 ↑
    ; oe_pass_eps_to_0_rule ↑
    ; simplify_math ↑
    step10  : chose_v_on_gammas ↑
    ; simplify_math ↑
    ; change_side_left ↑
    ; make_clear_intergral1 ↑
    ; make_clear_intergral2 ↑
    ; integral_1 ↑
    step11  : special1 ↑
    ; simplify_equation ↑
    step12  : interpretation_of_a_weak_equality ↑
    ; change_side_right ↑
    ; simplify_equation ↑

Model l101ref : step1; step2; step3; step4; step5; step6; step7
    ; step8; step9; step10; step11; step12
```

7.7 Fifth Block

The code includes the source term and the proof itself corresponding to Lemma 65 in Section 6.1.

Fifth block source term:

```
PDE "l104_source_term"
```

```
Constant
    psi          : "psi"
    gammae_NorVec : "gammae_NorVec"
    gamma1_NorVec : "gamma1_NorVec"
    gammas_NorVec : "gammas_NorVec"
    omegae_NorVec : "omegae_NorVec"
    omega1_NorVec : "omega1_NorVec"
    omegas_NorVec : "omegas_NorVec"
    norvec : "norvec"
    block1 : "block1"

Region
    gammae : "gammae" [gamma_Ind_] [gamma_Dim_] [] gammae_Bou_ gammae_NorVec
    % Γε

    gamma1 : "gamma1" [gamma1_Ind_] [gamma1_Dim_] [] gamma1_Bou_ gamma1_NorVec
    % Γ1

    gammas : "gammas" [gammas_Ind_] [gammas_Dim_] [] gammas_Bou_ gammas_NorVec
    % Γ#

    omegae : "omegae" [omegae_Ind_] [1] [] gammae omegae_NorVec
    % Ωε(Γε)

    omega1 : "omega1" [omega1_Ind_] [1] [] gamma1 omega1_NorVec
    % Ω1(Γ1)

    omegas : "omegas" [omegas_Ind_] [1] [] gammas omegas_NorVec
    % Ω#(Γ#)

    gammav : "gammav" [gammav_Ind_] [1] [gammas,gamma1] gammav_Bou_ gammav_NorVec_
    % Γv = Γ#∪Γ1

    eps_reg : "eps_reg" [eps_reg_Ind_] [1] [0,1000] eps_reg_Bou_ eps_reg_NorVec_

Expression % use in source term
    o1xos  : omega1•omegas

Function % use in source term
    meas_rw_ : "Measure" [meas_Ind_] [meas_Var_] [] "Given"
    % Measure function
    meas_omegae : "Measure" [meas_Ind_] [omegae] [] "Given"
    % |Ωε| !!! meas_Var =
    meas_o1xos : "Measure" [meas_Ind_] [o1xos] [] "Given"
    % |Ω1xΩ#|

Expression % use in source term
```

```
    kappa0 : 1/meas_omegae
    kappa1 : 1/meas_o1xos
    null : 0

Variable
    x  : "x" [x_Ind_]  omegae
    x1 : "x1" [x1_Ind_] omega1
    xs : "xs" [xs_Ind_] omegas
    xg : "xg"[xg_Ind_] gammae
    eps : "eps" [eps_Ind_] eps_reg

Function
    ue : "ue" [ue_Ind_] [x] [(gammae ue_ null)] "Unknown"
    % uε : Ωε, uε = 0

    v  : "v" [v_Ind_] [x] [(gammas v_ null)] "Test"

    ae : "ae" [ae_Ind_] [x] [] "Unknown"

    f : "f" [f_Ind_] [x] [] "Unknown"

Expression % USE IN SOURCE TERM
    dv_dx  : ∂v/∂x
    due_dx : ∂ue/∂x

Operator % USE IN SOURCE TERM
    opB_v : "B" [opB_Ind_] [v] [xs,x1] [x] [eps]
    % B(v)

    opT_ue : "T" [opT_Ind_] [ue] [x] [xs,x1] [eps]
    % T(ue)

    trace_ue : "Trace" [trace_Ind_] [ue] [x] [xg_rw_] [trace_Pa_]
    % trace(ue)

    trace_v : "Trace" [trace_Ind_] [v] [xs,x1] [xg_rw_] [trace_Pa_]
    % trace(v)

    opT_due_dx : "T" [opT_Ind_] [due_dx] [x] [xs,x1] [eps]
    % T(∂ue/∂x)
PDE
    l104_source_term : kappa0•∫ ae•due_dx•dv_dx đx = kappa0•∫ f•v đx  % WORKING SOURCE
```

Fifth block in the reference proof:

```
Model "l104ref"
Function
```

```
    f0 : "f0" [f0_Ind_] [xs] [] "Unknown"

    a0 : "a0" [a0_Ind_] [x] [] "Unknown"

    w1 : "w1" [w1_Ind_] [x1] [] "Unknown"
    % w1 ∈ C#(Ω1)

    phi : "phi" [phi_Ind_] [xs] [] "Test"
    % w1 ∈ C(Ω#)

    theta1 : "theta1" [theta1_Ind_] [x1] [] "Unknown"

    oe_eps : "oe" [oe_Ind_] [eps] [] "Given"

    oe_rw_ : "oe" [oe_Ind_]  [oe_Var_] [] "Given"
    % O() tend to zero as ε->0 !

    u0 : "u0" [u0_Ind_] [xs] [] "Unknown"

    u1 : "u1" [u1_Ind_] [xs,x1] [] "Unknown"

    v0 : "v0" [v0_Ind_] [xs] [(gammas v0_ null)] "Test"
    % v0 ∈ CΓ#(Ω#), Test function

    v1 : "v1" [v1_Ind_] [xs,x1] [(gammas v1_ null)] "Test"
    % v1 ∈ CΓ#(Ω#,CΓ#(Ω1)), Test function

    ut1 : "ut1" [ut1_Ind_] [xs,x1] [] "Unknown"
    % u~1

    eta : "eta" [eta_Ind_] [xs,x1] [] "Unknown"

Expression
    v0_epsv1 : v0 + eps•v1

Operator
    opB_v0_epsv1 : "B" [opB_Ind_] [v0_epsv1] [xs,x1] [x] [eps]

#Include "basic_math_rule.proof"
#Include "math_rule.proof"

Rule
    ue_on_gamma : trace_ue → 0
    % ue = 0 on Γ check

    approximation_of_Tu  : opT_ue → u0 + oe_eps
    % Assumption  : eq47
```

```
    v_on_gamma1 : trace_v → 0
    % v = 0 on Γ1

    assumption_L99 : opT_ue → u0 + eps•u1
    % T(ue) → u0 + eps•u1

    two_scale_limit_of_a_derivative : u1-x1•∂u0/∂xs → ut1

    substitute_psi : psi → kappa1•∫∫opT_due_dx•v đx1 đxs

    psi_pass_eps_to_0 : opT_due_dx → eta

    result_of_P97 : eta → ∂u0/∂xs + ∂ut1/∂x1

    pretty_oe_eps : a_ + oe_eps → a_ - oe_eps

    test : a_ → a_

    repalace_v : v → opB_v0_epsv1

    eps_expansion : 1/eps•(a_+b_) → 1/eps•a_+1/eps•b_

    special1 : kappa0•∫ k_•a2_ đx_ + kappa0•∫ k_•b2_ đx_
     = d_ → kappa0•∫ k_•(a2_ + b2_) đx_ = d_

Operator
    opT_ae : "T" [opT_ae_Ind_] [ae] [x] [xs,x1] [eps]
    % T(ae)

    opT_f : "T" [opT_f_Ind_] [f] [x] [xs,x1] [eps]
    % T(f)
Rule
    special2 : opT_ae → a0
    special3 : opT_f  → f0 + oe_eps

Step
    step1   : repalace_v ↑
    step2   : derivation_rule_for_B ↑
    ; linearity_derivative ↑
    ; take_constant_out_of_derivative ↑
    step3   : linearity_opB ↑
    step4   : take_const_out_of_opB ↑
    ; eps_expansion ↑
    ; simplify_math ↑
    ; simplify_derivative4 ↑
    step5   : simplify_opB ↑
```

```
    ; simplify_math ↑
    ; eps_to_oe ↑
    ; simplify_multi_scale ↑
    step6   : linearity_opB ↑
    ; expansion ↓
    ; integral_linearity ↓
    ; expansion ↓
    ; simplify_multi_scale ↑
    ; pretty_oe ↑
    ; mult_with_1 ↑
    ; pretty_meas ↑
    ; special1 ↑
    step7   : inverse_linearity_opB ↑
    step8   : approximation_between_B_and_TS ↑
    ; expansion ↓
    ; integral_linearity ↓
    ; expansion ↓
    ; take_constant_out_of_integral ↑
    ; eps_to_oe ↑
    ; simplify_multi_scale ↑
    step9   : adjoint_or_dual_of_TS ↑
    step10  : product_rule_of_opT ↑
    ; special2 ↑
    ; special3 ↑
    ; psi_pass_eps_to_0 ↑
    ; oe_pass_eps_to_0_rule ↑
    ; simplify_multi_scale ↑
    ; simplify_math ↑
    step11  : result_of_P97 ↑

Model l104ref : step1; step2; step3; step4; step5; step6
; step7; step8; step9; step10; step11
```

7.8 Sixth Block

The code includes the source term and the proof itself corresponding to Lemma 66 in Section 6.1.

Sixth block source term:

```
PDE "l105_source_term"
Constant
    psi           : "psi"
    gammae_NorVec : "gammae_NorVec"
    gamma1_NorVec : "gamma1_NorVec"
    gammas_NorVec : "gammas_NorVec"
    omegae_NorVec : "omegae_NorVec"
    omega1_NorVec : "omega1_NorVec"
```

```
    omegas_NorVec : "omegas_NorVec"
    norvec : "norvec"
    block1 : "block1"

Region
    gammae : "gammae" [gamma_Ind_] [gamma_Dim_] [] gammae_Bou_ gammae_NorVec
    % Γε

    gamma1 : "gamma1" [gamma1_Ind_] [gamma1_Dim_] [] gamma1_Bou_ gamma1_NorVec
    % Γ1

    gammas : "gammas" [gammas_Ind_] [gammas_Dim_] [] gammas_Bou_ gammas_NorVec
    % Γ#

    omegae : "omegae" [omegae_Ind_] [1] [] gammae omegae_NorVec
    % Ωε(Γε)

    omega1 : "omega1" [omega1_Ind_] [1] [] gamma1 omega1_NorVec
    % Ω1(Γ1)

    omegas : "omegas" [omegas_Ind_] [1] [] gammas omegas_NorVec
    % Ω#(Γ#)

    gammav : "gammav" [gammav_Ind_] [1] [gammas,gamma1] gammav_Bou_ gammav_NorVec_
    % Γv = Γ#∪Γ1

    eps_reg : "eps_reg" [eps_reg_Ind_] [1] [0,1] eps_reg_Bou_ eps_reg_NorVec_

Expression % use in source term
    o1xos  : omega1•omegas

Function % use in source term
    meas_rw_ : "Measure" [meas_Ind_] [meas_Var_] [] "Given"
    % Measure function
    meas_omegae : "Measure" [meas_Ind_] [omegae] [] "Given"
    % |Ωε| !!! meas_Var =
    meas_o1xos : "Measure" [meas_Ind_] [o1xos] [] "Given"
    % |Ω1xΩ#|

Expression % use in source term
    kappa0 : 1/meas_omegae
    kappa1 : 1/meas_o1xos
    null : 0

Variable
    x  : "x" [x_Ind_]  omegae
    x1 : "x1" [x1_Ind_] omega1
```

```
    xs : "xs" [xs_Ind_] omegas
    xg : "xg"[xg_Ind_] gammae
    eps : "eps" [eps_Ind_] eps_reg

Function
    ue : "ue" [ue_Ind_] [x] [(gammae ue_ null)] "Unknown"
    % uε : Ωε, uε = 0

    v  : "v" [v_Ind_] [x] [(gammas v_ null)] "Test"

    v0 : "v0" [v0_Ind_] [xs] [(gammas v0_ null)] "Test"
    % v0 ∈ CΓ#(Ω#), Test function

    u0 : "u0" [u0_Ind_] [xs] [] "Unknown"

    a0 : "a0" [a0_Ind_] [x] [] "Unknown"

    ut1 : "ut1" [ut1_Ind_] [xs,x1] [] "Unknown"
    % u~1

    v1 : "v1" [v1_Ind_] [xs,x1] [(gammas v1_ null)] "Test"
    % v1 ∈ CΓ#(Ω#,CΓ#(Ω1)), Test function

    f0 : "f0" [f0_Ind_] [x] [] "Unknown"

Expression % USE IN SOURCE TERM
    dv_dx  : ∂v/∂x
    due_dx : ∂ue/∂x
    v0_epsv1 : v0 + eps•v1

Operator % USE IN SOURCE TERM
    opB_v : "B" [opB_Ind_] [v] [xs,x1] [x] [eps]
    % B(v)

    opT_ue : "T" [opT_Ind_] [ue] [x] [xs,x1] [eps]
    % T(ue)

    trace_ue : "Trace" [trace_Ind_] [ue] [x] [xg_rw_] [trace_Pa_]
    % trace(ue)

    trace_v : "Trace" [trace_Ind_] [v] [xs,x1] [xg_rw_] [trace_Pa_]
    % trace(v)

    opT_due_dx : "T" [opT_Ind_] [due_dx] [x] [xs,x1] [eps]
    % T(∂ue/∂x)

    opB_v0_epsv1 : "B" [opB_Ind_] [v0_epsv1] [xs,x1] [x] [eps]
```

```
PDE
    l105_source_term : ∫∫ a0•(∂u0/∂xs+∂ut1/∂x1)•(∂v0/∂xs+∂v1/∂x1) đxs đx1 = ∫ f0•v0 đ
```

Sixth block in the reference proof:

```
Model "l105ref"
Function
    oe_eps : "oe" [oe_Ind_] [eps] [] "Bigo"

    oe_rw_ : "oe" [oe_Ind_]  [oe_Var_] [] "Bigo"
    % O() tend to zero as ε->0 !

    u1 : "u1" [u1_Ind_] [xs,x1] [] "Unknown"

    eta : "eta" [eta_Ind_] [xs,x1] [] "Unknown"

    ae : "ae" [ae_Ind_] [x] [] "Unknown"

    w1 : "w1" [w1_Ind_] [x1] [] "Unknown"
    % w1 ∈ C#(Ω1)

    phi : "phi" [phi_Ind_] [xs] [] "Test"
    % w1 ∈ C(Ω#)

    f : "f" [f_Ind_] [x] [] "Unknown"

    theta1 : "theta1" [theta1_Ind_] [x1] [] "Unknown"
#Include "basic_math_rule.proof"
#Include "math_rule.proof"

Rule
    ue_on_gamma : trace_ue → 0
    % ue = 0 on Γ check

    approximation_of_Tu  : opT_ue → u0 + oe_eps
    % Assumption  : eq47

    v_on_gamma1 : trace_v → 0
    % v = 0 on Γ1

    assumption_L99 : opT_ue → u0 + eps•u1
    % T(ue) → u0 + eps•u1

    two_scale_limit_of_a_derivative : u1-x1•∂u0/∂xs → ut1

    substitue_psi : psi → kappa1•∫∫opT_due_dx•v đx1 đxs
```

```
    psi_pass_eps_to_0 : kappa1•∫∫opT_due_dx•v đx1 đxs → kappa1•∫∫eta•v đx1 đxs

    pretty_oe_eps : a_ + oe_eps → a_ - oe_eps

    repalace_v : v → opB_v0_epsv1

    repalace_v0 : v0 → 0

    repalace_v1 : v1 → w1•phi

    eps_expansion : 1/eps•(a_+b_) → 1/eps•a_+1/eps•b_

Step
    step1   : repalace_v0 ↑
    step2   : repalace_v1 ↑
    ; simplify_derivative ↑
    ; simplify_math ↑
    ; take_constant_out_of_derivative ↑
    ; fubini_theorem ↑
    ; take_constant_out_of_integral4 ↑
    ; expansion ↑
    step3  : interpretation_of_a_weak_equality ↑
    ; integral_linearity ↑
    ; take_constant_out_of_integral5 ↑
    ; change_side_right ↑
    ; simplify_sign ↑
    step4   : the_linear_operator_associated_to_the_microscopic_problem ↑

Model l105ref : step1; step2; step3; step4
```

7.9 Seventh Block

The code includes the source term and the proof itself corresponding to Lemma 67 in Section 6.1.

Seventh block source term:

```
PDE "l106_source_term"
Constant
    psi           : "psi"
    gammae_NorVec : "gammae_NorVec"
    gamma1_NorVec : "gamma1_NorVec"
    gammas_NorVec : "gammas_NorVec"
    omegae_NorVec : "omegae_NorVec"
    omega1_NorVec : "omega1_NorVec"
    omegas_NorVec : "omegas_NorVec"
    norvec : "norvec"
```

```
    block1 : "block1"

Region
    gammae : "gammae" [gamma_Ind_] [gamma_Dim_] [] gammae_Bou_ gammae_NorVec
    % Γε

    gamma1 : "gamma1" [gamma1_Ind_] [gamma1_Dim_] [] gamma1_Bou_ gamma1_NorVec
    % Γ1

    gammas : "gammas" [gammas_Ind_] [gammas_Dim_] [] gammas_Bou_ gammas_NorVec
    % Γ#

    omegae : "omegae" [omegae_Ind_] [1] [] gammae omegae_NorVec
    % Ωε(Γε)

    omega1 : "omega1" [omega1_Ind_] [1] [] gamma1 omega1_NorVec
    % Ω1(Γ1)

    omegas : "omegas" [omegas_Ind_] [1] [] gammas omegas_NorVec
    % Ω#(Γ#)

    gammav : "gammav" [gammav_Ind_] [1] [gammas,gamma1] gammav_Bou_ gammav_NorVec_
    % Γv = Γ#∪Γ1

    eps_reg : "eps_reg" [eps_reg_Ind_] [1] [0,1] eps_reg_Bou_ eps_reg_NorVec_

Expression % use in source term
    o1xos  : omega1•omegas                                      % Ω1xΩ# !!!

Function % use in source term
    meas_rw_ : "Measure" [meas_Ind_] [meas_Var_] [] "Given"
    % Measure function
    meas_omegae : "Measure" [meas_Ind_] [omegae] [] "Given"
    % |Ωε| !!! meas_Var =
    meas_o1xos : "Measure" [meas_Ind_] [o1xos] [] "Given"
    % |Ω1xΩ#|

Expression % use in source term
    kappa0 : 1/meas_omegae
    kappa1 : 1/meas_o1xos
    null : 0

Variable
    x  : "x" [x_Ind_]  omegae
    x1 : "x1" [x1_Ind_] omega1
    xs : "xs" [xs_Ind_] omegas
    xg : "xg"[xg_Ind_] gammae
```

```
    eps : "eps" [eps_Ind_] eps_reg

Function
    a0 : "a0" [a0_Ind_] [x] [] "Unknown"

    u0 : "u0" [u0_Ind_] [xs] [] "Unknown"

    ut1 : "ut1" [ut1_Ind_] [xs,x1] [] "Unknown"
    % u~1

    v0 : "v0" [v0_Ind_] [xs] [(gammas v0_ null)] "Test"
    % v0 ∈ CΓ#(Ω#), Test function

    v1 : "v1" [v1_Ind_] [xs,x1] [(gammas v1_ null)] "Test"
    % v1 ∈ CΓ#(Ω#,CΓ#(Ω1)), Test function

    f0 : "f0" [f0_Ind_] [x] [] "Unknown"

    v  : "v" [v_Ind_] [x] [(gammas v_ null)] "Test"

    ue : "ue" [ue_Ind_] [x] [(gammae ue_ null)] "Unknown"
    % uε : Ωε, uε = 0

Expression % USE IN SOURCE TERM
    dv_dx  : ∂v/∂x
    due_dx : ∂ue/∂x
    v0_epsv1 : v0 + eps•v1

Operator % USE IN SOURCE TERM
    opB_v : "B" [opB_Ind_] [v] [xs,x1] [x] [eps]
    % B(v)

    opT_ue : "T" [opT_Ind_] [ue] [x] [xs,x1] [eps]
    % T(ue)

    trace_ue : "Trace" [trace_Ind_] [ue] [x] [xg_rw_] [trace_Pa_]
    % trace(ue)

    trace_v : "Trace" [trace_Ind_] [v] [xs,x1] [xg_rw_] [trace_Pa_]
    % trace(v)

    opT_due_dx : "T" [opT_Ind_] [due_dx] [x] [xs,x1] [eps]
    % T(∂ue/∂x)

    opB_v0_epsv1 : "B" [opB_Ind_] [v0_epsv1] [xs,x1] [x] [eps]

PDE
```

```
l106_source_term : ∫∫ a0•(∂u0/∂xs+∂ut1/∂x1)•(∂v0/∂xs+∂v1/∂x1) đxs đx1 = ∫ f0•v0 đ
```

Seventh block in the reference proof:

```
Model "l106ref"
Function
    oe_eps : "oe" [oe_Ind_] [eps] [] "Given"

    oe_rw_ : "oe" [oe_Ind_]  [oe_Var_] [] "Given"
    % O() tend to zero as ε->0 !

    u1 : "u1" [u1_Ind_] [xs,x1] [] "Unknown"

    eta : "eta" [eta_Ind_] [xs,x1] [] "Unknown"

    ae : "ae" [ae_Ind_] [x] [] "Unknown"

    f : "f" [f_Ind_] [x] [] "Unknown"

    w1 : "w1" [w1_Ind_] [x1] [] "Unknown"
    % w1 ∈ C#(Ω1)

    phi : "phi" [phi_Ind_] [xs] [] "Test"
    % w1 ∈ C(Ω#)

    theta1 : "theta1" [theta1_Ind_] [x1] [] "Unknown"

#Include "basic_math_rule.proof"
#Include "math_rule.proof"

Rule
    ue_on_gamma : trace_ue → 0
    % ue = 0 on Γ check

    approximation_of_Tu  : opT_ue → u0 + oe_eps
    % Assumption  : eq47

    v_on_gamma1 : trace_v → 0
    % v = 0 on Γ1

    assumption_L99 : opT_ue → u0 + eps•u1
    % T(ue) → u0 + eps•u1

    two_scale_limit_of_a_derivative : u1-x1•∂u0/∂xs → ut1

    substitue_psi : psi → kappa1•∫∫opT_due_dx•v đx1 đxs
```

```
psi_pass_eps_to_0 : kappa1•∫∫opT_due_dx•v đx1 đxs → kappa1•∫∫eta•v đx1 đxs

pretty_oe_eps : a_ + oe_eps → a_ - oe_eps

repalace_v : v → opB_v0_epsv1

repalace_v0 : v0 → 0

repalace_v1 : v1 → ∂v0/∂xs•theta1

eps_expansion : 1/eps•(a_+b_) → 1/eps•a_+1/eps•b_

result_of_l105 : ∂ut1/∂x1 → ∂u0/∂xs•∂theta1/∂x1

Step
    step1   : repalace_v1 ↑
    ; derivative_product_rule ↑
    ; derivative_change_order ↑
    ; simplify_derivative2 ↑
    ; simplify_derivative3 ↑
    ; simplify_math ↑
    step2   : result_of_l105 ↑
    step3   : introduction_of_a_kronecker_symbol ↑
    ; fubini_theorem ↑
    ; take_constant_out_of_integral6 ↑
    ; take_constant_out_of_integral6 ↑

Model l106ref : step1; step2; step3
```

8 Implementation of extensions

The Green rule, i.e. Proposition 49 in the reference proof, has been extended to the n-dimensional case in Proposition 71. Its extension to vector valued functions is stated as follows.

Proposition 84 ***[Green Rule]*** *If two vector valued functions* $\mathbf{u} = (u_i)_{i=1,..,n}$, $\mathbf{v} = (v_j)_{j=1,..,n} \in (H^1(\Omega))^n$ *then the traces of* $\mathbf{u}$ *and* $\mathbf{v}$ *on* Γ *are well defined and*

$$\int_\Omega u_i \frac{\partial v_j}{\partial x}\ dx = \int_\Gamma tr(u_i)\ tr(v_j)\ n_\Gamma\ ds(x) - \int_\Omega v_i \frac{\partial u_j}{\partial x}\ dx \tag{116}$$

for all i *and* $j \in \{1, .., n\}$.

The implementations of these two extensions are detailed in the two following subsections and the result of their combination appears in the last section devoted to the program outputs.

8.1 Implementation of extension to n-dimensional regions

The implementation of this extension includes declarations of variables, functions, operators and rules related to the reference proof, and then declarations of a variable and a rule for the extension.

```
Extension "green_rule_extension_ndim" of Model "green_rule"

% ================================================================================
%        " green_rule_extension_ndim"
% ================================================================================
Variable
    var_ : "x" [] varRegion_

   Rule
     green_rule : X__ → Y__
% ================================================================================
%        Extension ndim
% ================================================================================
Variable
  var_'ndim : "x" [varIndex_] varRegion_

  Rule
    ext1'ndim : var_ ⇉ var_'ndim

  Extension
     green_rule_extension_ndim : green_rule;; p_; ext1'ndim
```

The extension itself is the command

```
green_rule_extension_ndim : green_rule;; p_; ext1'ndim
```

where `green_rule;;` is a pattern for localization of the operation refering to the Green rule of the reference proof and `p_;ext1'ndim` is translated into the strategy `s=`InnerMost`(p→p;ext1'ndim)`. Since `ext1'ndim` is a rewriting rule at the top, the strategy `s` applies this rule using the InnerMost strategy.

8.2 Extension to vector-valued solution

The implementation of the extension of the Green rule to the case of vector valued functions as in Proposition 116 follows the same principle and is not further discussed.

```
 Extension "green_rule_extension_vvf" of Model "green_rule"

% ============================================================================
%        "reen_rule_extension_vvf"
% ============================================================================

Function
  fun1_ : "u" [] [fun1InputVar_] [(fun1Bou_ fun1OnBou_ fun1Value_)] "Unknown"
  fun2_ : "v" [] [fun2InputVar_] [(fun2Bou_ fun2OnBou_ fun2Value_)] "Test"

Rule
  green_rule : X__  →  Y__
```

```
% ===========================================================================
%        Extension vvf
% ===========================================================================
Function
   fun1_'vvf : "u" [fun1Index_] [fun1InputVar_] [(fun1Bou_ fun1OnBou_ fun1Value_)]
                    "Unknown"
   fun2_'vvf : "v" [fun2Index_] [fun2InputVar_] [(fun2Bou_ fun2OnBou_ fun2Value_)]
               "Test"

Rule
    ext2'vvf : fun1_ ⇉ fun1_'vvf
    ext3'vvf : fun2_ ⇉ fun2_'vvf

Extension
    green_rule_extension_vvf : green_rule;; p_; (ext2'vvf | ext3'vvf)
```

9 Latex outputs

Any expression, proof or extension in the Processing Language can be transformed into Latex format for the purpose of checking its correctness. We provide Latex outputs of the reference Green rule, of the two extensions and their results when applied to the reference Green rule, of their combination of the two extensions and finally of the result of the application of the combination to the reference Green formula. Different options of display can be used, but in all cases the keyword as `Oper`, `Fun`, `Var` etc are hidden. Here, only the most important arguments of the operators, the functions and the variables are visible. This can be changed on demand. The notations ↑, `IM`, `LC` represent the strategy BottomUp, InnerMost and LeftChoice.

9.1 Green rule extensions

9.1.1 Reference Green rule

The global structure of this little reference proof is kept on the format of the expressions in the Processing Language with the keywords `Proof`, `Model`, `Step` and the names `step1` of step and `Green_rule` of strategy. The functions `NormalOf`, `BoundaryOf` and `RegionOf` are to recover the region field, the boundary field and the normal direction in a variable, a region and boundary of a region respectively.

Proof :
Model (
Step(`step1`,
(`green_rule`:
$\int \mathtt{u} \cdot \frac{\partial \mathtt{v}}{\partial \mathtt{x}} \, d\mathtt{x} \rightarrow -\int \frac{\partial \mathtt{u}}{\partial \mathtt{x}} \cdot \mathtt{v} \, d\mathtt{x}$
$+\int \mathtt{Trace(u)} \cdot \mathtt{Trace(v)} \cdot \mathtt{NormalOf(BoundaryOf(RegionOf(x)))} \, d\mathtt{s}) \uparrow$
)
)

9.1.2 Green rule extension to n-dimensional regions

An extension starts with the function name `Extension` instead of `Proof` for a proof, then it follows the grammar defined for extensions with the possible use of the two strategies `IM`, `LC` and localization at relative positions (here the position 2 is relative to the position of the root of `x`). The patterns of the search are `green_rule`: `X___` $\to$ `Y___`, `x` and `p_` The added context is [`i__`, $\perp$] where the brackets refer to the function List. The gain in using an extension over defining a complete proof is visible in the size of the added term which is the complement brought to the proof.

Extension:

```
(IM(green_rule: X__→Y__),
    (IM(p__),
    (x,
      (2, [i__, ⊥])
    )
  )
```

The structure of the reference Green rule is kept after application of the above extension, only the indices have been added.

Proof :

Model(
Step(`step1`,
(`green_rule`:

$$\int \mathtt{u} \cdot \frac{\partial \mathtt{v}}{\partial \mathtt{x}_{\mathtt{i\_}}}\, d\mathtt{x}_{\mathtt{i\_}} \to -\int \frac{\partial \mathtt{u}}{\partial \mathtt{x}_{\mathtt{i\_}}} \cdot \mathtt{v}\, d\mathtt{x}_{\mathtt{i\_}} + \int \mathtt{Trace(u)} \cdot \mathtt{Trace(v)} \cdot \mathtt{NormalOf(BoundaryOf(RegionOf(}\mathtt{x}_{\mathtt{i\_}}\mathtt{)))}\, d\mathtt{s}) \uparrow$$

)
)

9.1.3 Green rule extension to vector valued functions

The structure of the extension is the same except that the strategy LeftChoice is used to add different indices on the function `u` and on the function `v`.

Extension:

```
(IM(green_rule: X__→Y__),
    IM(p__),
    LC(
      (u,
        (2, [j__, ⊥])
      ),
      (v,
        (2, [k__, ⊥])
      )
    )
  )
```

Proof :

Model(
Step(step1,
(green_rule:
$\int \mathtt{u}_{\mathtt{j}\_} \cdot \frac{\partial \mathtt{v}_{\mathtt{k}\_}}{\partial \mathtt{x}} \, d\mathtt{x} \rightarrow - \int \frac{\partial \mathtt{u}_{\mathtt{j}\_}}{\partial \mathtt{x}} \cdot \mathtt{v}_{\mathtt{k}\_} \, d\mathtt{x}$
$+ \int \mathtt{Trace}(\mathtt{u}_{\mathtt{j}\_}) \cdot \mathtt{Trace}(\mathtt{v}_{\mathtt{k}\_}) \cdot \mathtt{NormalOf}(\mathtt{BoundaryOf}(\mathtt{RegionOf}(\mathtt{x}))) \, d\mathtt{s}) \uparrow$
)
)

9.1.4 Combination of the two extensions

The following combination of the two extensions has been built automatically. Evidently, it combines the features of the two extensions and save the time to design another extension.

Extension:

```
(IM(green_rule: X__→Y__),
   IM(p__),
   LC(
     (v,
       (2, [k__, ⊥])
     ),
     (x,
       (2, [i__, ⊥])
     )
     (u,
       (2, [j__, ⊥])
     )
   )
 )
```

Proof :
Model(
Step(step1,
(green_rule:
$\int \mathtt{u}_{\mathtt{j}\_} \cdot \frac{\partial \mathtt{v}_{\mathtt{k}\_}}{\partial \mathtt{x}_{\mathtt{i}\_}} \, d\mathtt{x}_{\mathtt{i}\_} \rightarrow - \int \frac{\partial \mathtt{u}_{\mathtt{j}\_}}{\partial \mathtt{x}} \cdot \mathtt{v}_{\mathtt{k}\_} \, d\mathtt{x}_{\mathtt{i}\_}$
$+ \int \mathtt{Trace}(\mathtt{u}_{\mathtt{j}\_}) \cdot \mathtt{Trace}(\mathtt{v}_{\mathtt{k}\_}) \cdot \mathtt{NormalOf}(\mathtt{BoundaryOf}(\mathtt{RegionOf}(\mathtt{x}_{\mathtt{i}\_}))) \, d\mathtt{s}) \uparrow$
)
)

References

[BCHPM04] Y. Bertot, P. Castéran, G. Huet, and C. Paulin-Mohring. *Interactive theorem proving and program development : Coq'Art : the calculus of inductive constructions*. Springer, Berlin, New York, 2004.

[BGL14] W. Belkhir, A. Giorgetti, and M. Lenczner. A symbolic transformation language and its application to a multiscale method. *Journal of Symbolic Computation*, 65:49–78, 2014.

[BL04] Einar B.J. and Christoph L. Theorem reuse by proof term transformation. In *Theorem Proving in Higher Order Logics*, pages 152–167, 2004.

[BN99] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, 1999.

[LS07] 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*, 46(5-6):776–805, 2007.

[YBL14a] B. Yang, W. Belkhir, and M. Lenczner. Computer-aided derivation of multiscale models: A rewriting framework. *International Journal for Multiscale Computational Engineering*, 12(2), 2014.

[YBL14b] Bin Yang, Walid Belkhir, and Michel Lenczner. Computer-aided derivation of multiscale models: A rewriting framework. *International Journal for Multiscale Computational Engineering.*, 12(2):91–114, 2014.