Preuves formelles en mathématiques appliquées : formalisation en Coq des éléments finis de Lagrange simpliciaux

Houda Mouhcine

université PARIS-SACLAY

Formal Proofs in Applied Mathematics : A Coq Formalization of Simplicial Lagrange Finite Elements

Preuves Formelles en Mathématiques Appliquées : Formalisation en Coq des Éléments Finis de Lagrange Simpliciaux

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°580, sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique mathématique
Graduate School : Informatique et sciences du numérique. Référent : Faculté des sciences d'Orsay

Thèse préparée dans l'unité de recherche : **Laboratoire Méthodes Formelles (Université Paris Saclay, CNRS, ENS Paris-Saclay)**, sous la direction de **Sylvie BOLDO**, directrice de recherche, la co-direction de **Micaela MAYERO**, maîtresse de conférences, le co-encadrement de **François CLÉMENT**, chargé de recherche

Thèse soutenue à Paris-Saclay, le 09 décembre 2024, par

Houda MOUHCINE

Composition du Jury

Membres du jury avec voix délibérative

Sylvain CONCHON Professeur des universités, université Paris Saclay, France	Président
Damien POUS Directeur de recherche, CNRS et ENS Lyon, France	Rapporteur & Examinateur
Alan SCHMITT Directeur de recherche, Inria Rennes, France	Rapporteur & Examinateur
Laura GRIGORI Professeure, EPFL, Suisse	Examinatrice
Sébastien IMPÉRIALE Chargé de recherche, Inria Saclay, France	Examinateur

THESE DE DOCTORAT

NNT : 2024UPASG112

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Preuves Formelles en Mathématiques Appliquées: Formalisation en Coq des Éléments Finis de Lagrange Simpliciaux.

Mots clés : Preuves formelles, Coq, Méthode des éléments finis, Lagrange FE.

Résumé : Cette thèse est consacrée au développement de preuves formelles de théorèmes et de propositions mathématiques dans le domaine de l'analyse réelle, en utilisant l'assistant de preuve Coq pour assurer leur exactitude. Le cœur de ce travail est divisé en deux parties principales. La première partie se concentre sur l'utilisation de Coq pour formaliser le principe d'induction de Lebesgue et le théorème de Tonelli, permettant le calcul d'intégrales doubles sur des espaces produits en intégrant itérativement par rapport à chaque variable. Ce travail s'appuie sur des recherches antérieures en théorie de la mesure et sur l'intégrale de Lebesgue. La deuxième partie s'inscrit dans le cadre de la méthode des éléments finis (MEF), une technique numérique largement utilisée pour résoudre numériquement les équations aux dérivées partielles. La MEF joue un rôle important dans de nombreux programmes de simulation industrielle, en particulier pour l'approximation de solutions à des problèmes complexes. Plus précisément, nous visons à construire les éléments finis, en nous concentrant sur les éléments finis de Lagrange simpliciaux. Ce travail nécessite l'utilisation d'un large éventail de concepts algébriques tels que les familles finies, les monoïdes, les espaces vectoriels, les espaces affines et les espaces de dimension finie. Pour mener cette étude, nous commençons par définir un élément fini général dans Coq. Ensuite, nous démontrons l'efficacité de cette définition en construisant les éléments finis simpliciaux de Lagrange. Cela implique la formalisation en logique classique de plusieurs composants fondamentaux, y compris la construction de l'espace d'approximation, l'expression de ses polynômes de base de Lagrange, et la formalisation des transformations géométriques affines et de la propriété d'unisolvance des éléments finis de Lagrange.

Title : Formal Proofs in Applied Mathematics: A Coq Formalization of Simplicial Lagrange Finite Elements

Keywords : Formal proofs, Coq, Finite Element Method, Lagrange FE.

Abstract : This thesis is dedicated to developing formal proofs of mathematical theorems and propositions within the field of real analysis, using the Coq proof assistant to ensure their correctness. The core of this work is divided into two main parts. The first part focuses on using Coq to formalize the Lebesgue induction principle and the Tonelli theorem, allowing the computation of double integrals on product spaces by iteratively integrating with respect to each variable. This work builds upon previous research in measure theory and the Lebesgue integral. The second part is within the framework of the Finite Element Method (FEM), a widely used numerical technique for numerically solving partial differential equations. FEM plays an important role in numerous industrial simulation programs, particularly in approximating solutions to complex problems. Specifically, we aim to construct finite elements, focusing on simplicial Lagrange finite elements. This work requires the use of a broad range of algebraic concepts such as finite families, monoids, module spaces, affine spaces, and finite-dimensional spaces. To conduct this study, we begin by defining a general finite element in Coq. Then, we show the effectiveness of this definition by building the widely used simplicial Lagrange finite elements. This involves the formalization in classical logic of several foundational components, including the construction of the approximation space, expressing its Lagrange polynomial basis, and formalizing affine geometric transformations and the unisolvence property of Lagrange finite elements.

Synthèse en Français

Cette thèse est consacrée au développement de preuves formelles de théorèmes et de propositions mathématiques dans le domaine de l'analyse réelle, en utilisant l'assistant de preuve Coq pour assurer leur exactitude. Le cœur de ce travail est divisé en deux parties principales.

La première partie se concentre sur l'utilisation de Coq pour formaliser des concepts mathématiques fondamentaux en analyse fonctionnelle, notamment le *principe d'induction de Lebesgue* et le *théorème de Tonelli*. Ce travail sert de base pour prouver des propriétés liées aux fonctions mesurables. Le principe d'induction de Lebesgue se présente comme une technique de preuve pour établir des résultats concernant les fonctions mesurables non négatives, notamment celles qui impliquent des intégrales. Il reflète les trois étapes de construction suivies par *Henri Lebesgue* pour définir son intégrale. Ces étapes comprennent d'abord l'établissement de la propriété pour des fonctions indicatrices, puis son extension aux fonctions simples non négatives en vérifiant sa compatibilité avec les opérations linéaires positives. Enfin, la propriété s'applique à toutes les fonctions mesurables non négatives en s'assurant de sa compatibilité avec l'opération de supremum. Ce principe est dérivé d'un type inductif et apparaît comme un outil nécessaire pour prouver le théorème de Tonelli.

Le théorème de Tonelli permet de simplifier le calcul d'intégrales multiples en établissant leur égalité avec des intégrales itérées, facilitant ainsi l'interchangeabilité des ordres d'intégration. Ce théorème est applicable spécifiquement aux fonctions mesurables non négatives et s'aligne avec le théorème de Fubini, qui étend son domaine d'application aux fonctions intégrables avec des signes arbitraires. Dans le cadre de cette thèse, l'accent est mis sur les fonctions non négatives à deux variables, ce qui prépare le terrain pour de futures explorations concernant les équations aux dérivées partielles (EDP) impliquant plusieurs variables.

La deuxième partie de ce travail de recherche, qui constitue le principal axe de cette thèse, s'intéresse à la formalisation dans Coq et à la construction des éléments finis, qui sont essentiels à la méthode des éléments finis (MEF). Cette méthode est largement utilisée pour trouver des solutions approximatives à des problèmes de valeur aux limites pour des EDP dans divers domaines scientifiques tels que la physique, la mécanique et la biologie. L'utilisation étendue et le succès de la MEF par les numériciens peuvent être attribués à sa flexibilité et à sa précision dans la gestion de problèmes présentant des géométries complexes, des discontinuités et des contraintes. Cependant, malgré les larges applications de la MEF, plusieurs défis critiques peuvent nuire à son efficacité et à sa fiabilité. Le processus de mise en place des simulations, de leur exécution et de l'analyse des résultats est souvent extrêmement chronophage, surtout pour les simulations complexes. De plus, les programmes de MEF sont généralement complexes, nécessitant une attention particulière aux détails. Même des erreurs mineures dans la mise en œuvre peuvent entraîner des contradictions significatives dans les résultats, compromettant la validité des résultats de simulation et pouvant conduire à des conclusions

erronées. Pour faire face à ces problèmes, des preuves formelles sont employées pour garantir la correction et la fiabilité des algorithmes et des méthodes utilisés dans les simulations de MEF.

Plus précisément, nous visons à formaliser un *élément fini (EF)*, qui est défini mathématiquement comme un triplet et formalisé sous forme d'un record. La MEF subdivise un grand domaine en parties plus petites et plus simples appelées éléments finis, typiquement reliés par des points appelés sommets. Chaque élément est caractérisé par un triplet constitué de trois composants complémentaires, notés $(K, \mathcal{P}, \Sigma)$. Le composant K représente l'élément géométrique pour l'élément fini, qui varie en forme, des intervalles dans des espaces unidimensionnels aux formes plus complexes comme des triangles ou des quadrilètres en deux dimensions, et des tétraèdres ou des cuboïdes en trois dimensions, en fonction de la complexité du domaine et des exigences de la simulation. Dans le cadre de cette thèse, nous nous concentrons spécifiquement sur les simplexes, qui sont une généralisation du concept de triangle ou de tétraèdre à des dimensions arbitraires. Le second composant $\mathcal{P}$ du triplet d'éléments finis représente un espace vectoriel de dimension finie de fonctions polynomiales définies sur l'élément géométrique K. Ces polynômes sont utilisés pour approximer la solution au sein de l'élément fini, où le degré du polynôme (par exemple, linéaire, quadratique) détermine la précision de l'approximation. Le troisième composant Σ du triplet correspond à une famille de formes linéaires, également appelées degrés de liberté locaux, qui agissent sur les fonctions au sein de l'espace $\mathcal{P}$. Ces formes sont choisies pour satisfaire la propriété d'unisolvance, garantissant que l'élément fini est bien défini. Cependant, traiter le problème pour des éléments individuels n'est pas suffisant. La méthode des éléments finis (MEF) assemble les équations locales de chaque élément fini en un système global d'équations qui représente l'ensemble du problème. En résolvant ce système global, nous obtenons une approximation de la solution qui couvre l'ensemble du domaine du problème.

L'accent de cette dernière partie de la recherche est la construction des éléments finis de la famille *Lagrange simpliciaux* en utilisant Coq. L'objectif principal de ce développement est d'établir un élément fini utile et de garantir que nos définitions pour les éléments finis sont suffisantes et valides pour être utilisées. Le choix de formaliser les éléments finis de Lagrange pour cette étude découle de leur utilisation répandue dans les applications de la MEF et de leur simplicité inhérente. Ce travail explore également la formalisation des définitions et des preuves concernant des propriétés essentielles telles que les polynômes de Lagrange, les transformations géométriques affines, et les éléments finis courrants et de références, entre autres.

Pour mettre cela en œuvre, nous devrons utiliser l'analyse réelle et des structures algébriques allant des monoïdes abéliens aux espaces de modules de la bibliothèque Coquelicot. Chaque composant contribue à l'objectif principal de cette thèse, qui est d'établir formellement la propriété d'unisolvance des éléments finis de Lagrange. Les orientations futures incluent la formalisation des éléments finis quadrangulaires, certaines formules de quadrature et la construction d'espaces de Sobolev. Ainsi, les efforts collectifs décrits dans ce travail visent un objectif à long terme plus large : la vérification formelle des programmes de calcul scientifique et des parties de bibliothèques C++ comme FreeFEM++ et XLiFE++, qui implémentent la méthode des éléments finis.

الإهداء

أهدي ثمرة جهدي المتواضع إلى من وهبوني الحياة والأمل، والنشأة على شغف الاطلاع والمعرفة، ومن علموني أن أرتقي سلم الحياة بحكمة وصبر؛ برا، وإحسانا، ووفاء لهما والدي العزيز، ووالدتي العزيزة.

Acknowledgments

Je tiens tout d'abord à exprimer ma profonde gratitude à mes encadrants de thèse pour leur encadrement exceptionnel tout au long de ces trois années.

Un immense merci à ma directrice de thèse, Sylvie Boldo, pour son soutien, ses précieux conseils, ses nombreuses relectures, sa gentillesse et sa patience, qui ont été des piliers dans l'accomplissement de ce travail. Je vous remercie du fond du cœur.

Je remercie également mon co-encadrant, François Clément, pour son guidage éclairé et ses retours constructifs pertinents qui ont grandement enrichi mes recherches. Sa capacité à ré-expliquer avec patience des concepts qui lui paraissent évidents a été d'une aide inestimable. Merci pour son engagement et sa persévérance.

Un grand merci aussi à ma co-encadrante, Micaela Mayero, pour son soutien constant et sa pédagogie remarquable. Sa capacité à comprendre mes interrogations et à y répondre de manière constructive, ainsi que son accompagnement tout au long de ces trois années, ont été cruciaux pour moi.

Je souhaite également remercier Vincent Martin pour son aide précieuse. Son expertise mathématique, sa créativité dans la résolution de preuves complexes, et son talent pour expliquer clairement, souvent à l'aide de dessins, ont été des atouts inestimables dans l'avancée de ce travail.

Je suis également reconnaissante envers Alan Schmitt et Damien Pous d'avoir accepté de rapporter ma thèse. Vos retours précieux m'ont permis d'améliorer significativement mon manuscrit. Un grand merci également à Sylvain Conchon d'avoir présidé mon jury de soutenance, ainsi qu'à Sébastien Impériale et Laura Grigori pour votre participation active au jury. Vos remarques constructives ont été très bénéfiques.

Je remercie chaleureusement les membres des laboratoires LMF de l'Université Paris-Saclay, l'équipe SERENA d'Inria Paris, et le LIPN de l'Université Sorbonne Paris Nord pour leur accueil. Un remerciement spécial à Annabelle Tissier, Stéphanie Raynaud et Joyce Soares pour leur soutien administratif essentiel à mon intégration et à la préparation de ma soutenance.

Je dédie un immense merci à ma famille, en particulier à mes parents adorés, Ahmed Mouhcine et Zohra El Wahabi. Votre amour inconditionnel, votre soutien et vos encouragements constants ont été ma force dans les moments bons comme difficiles. Je vous aime profondément.

Je remercie chaleureusement mes sœurs Hajar, Kaoutar, Soukaina, Chaimaa, et mon frère Oussama pour la joie et le soutien sans faille que vous m'apportez. Chacun de vous a contribué à cette aventure, par vos sourires, vos mots encourageants et votre capacité à me faire rire, même quand les défis semblaient insurmontables.

Un remerciement spécial à mon époux, Saâd, pour sa compréhension et son soutien durant les moments difficiles, et pour avoir partagé avec moi chaque étape de ce parcours.

Pour clore ce préambule, je souhaite dédier cette thèse à mon cher père, réalisant ainsi son rêve de voir tous ses enfants réussir et obtenir un doctorat. Ce travail est le fruit des sacrifices que tu as faits pour mon éducation et ma réussite. Merci pour tout.

Contents

Chapter 1

Introduction

1.1 Research Framework

In the digital age, where software and hardware systems form the backbone of global industries, ensuring the reliability and safety of these systems is paramount. This is where *formal methods* come into play, offering a solid framework for developing, verifying, and analyzing complex computing systems. Based on mathematical logic, formal methods consist of a comprehensive set of techniques designed to verify the correctness of systems relative to defined specifications. These techniques include methods like model checking [49], abstract interpretation [25], as well as various formal proof techniques [45]. These proof techniques encompass deductive verification, automated theorem proving, and interactive theorem proving, among others. As technology becomes increasingly sophisticated and integrated into multiple aspects of daily life, the potential impact of software and hardware failures grows exponentially. From automotive safety and aerospace engineering to financial services and healthcare, failures in these systems can lead to catastrophic outcomes, including loss of life and significant economic damage. Formal methods address this risk by providing a framework that can rigorously verify that systems perform as intended under all specified conditions. For instance, consider an autonomous vehicle designed to detect obstacles and make real-time driving decisions. If the software responsible for obstacle detection fails due to a coding error or a hardware malfunction, the vehicle might not recognize a pedestrian crossing the street or another vehicle stopping suddenly. This could result in a failure to apply brakes or take evasive action, leading to a collision. To mitigate these risks, formal methods can be employed during the development phase of the automotive software. Formal methods involve using mathematical models to verify the correctness of algorithms governing the vehicle's operations under various conditions. By applying these methods, developers can ensure that the software adheres strictly to all operational specifications and can handle both expected and unexpected scenarios safely. This verification helps in reducing the risk of software failures that could lead to catastrophic outcomes.

This thesis specifically focuses on the domain of formal proofs through *interactive theorem proving*, a process wherein a human user develops proofs using proof assistant software. To verify these formal proofs, a variety of tools are employed, including `ACL2` [50], Coq [23], `HOL Light` [44], Lean [29], Mizar [61] and PVS [63]. Such a proof assistant primarily involves formalizing statements and their corresponding proof into a form that computers can understand. This process requires defining all the variables, functions, and statements in mathematical and logical terms. Subsequently, the focus is on constructing these proofs thoroughly using a variety of *tactics*, assuming the hypotheses of the statements are correct. Tactics represent instructions

or commands that guide the proof assistant through logical deductions (i.e., through the process of building a formal proof step by step). When a user writes a proof, they apply tactics to break down the problem, simplify expressions, and logically derive the desired conclusion from the given hypotheses and assumptions. Moreover, the process of constructing a proof is more than just verifying that a formula or statement is correct; it is a mental exercise that forces the user to engage deeply with the underlying concepts and principles, thereby enhancing their grasp of the material. Instead of merely inputting a formula into an automated proof verifier and verifying its correctness, the act of deliberately thinking through why a formula holds provides valuable insights. In this work, we particularly focus on validating pen-and-paper/informal proofs, built by human beings, using Coq.

We have opted to work with the Coq proof assistant for multiple reasons. First and foremost, there have been several libraries developed that are highly relevant to our research. Notable among these are the *standard Reals library* for real numbers [57], the Coquelicot extension [14] for advanced real analysis, and the `Flocq` library for floating-point arithmetic [15]. In addition, there have been several libraries implemented addressing mathematical concepts, such as the *Lax-Milgram theorem* [10] and the *Lebesgue integral* [11]. As a student who joined during the later stages of these developments and came from a mathematical background without prior exposure to Coq, I had to learn Coq from scratch. Nonetheless, given the breadth of existing resources and the direct support from my supervisors, who have extensive experience working with Coq and are well-versed in its usage, it made practical sense for me to continue working in Coq. Additionally, Coq includes the Flocq library, a comprehensive resource for floating-point arithmetic that is essential for bounding *rounding errors*. This library will prove useful in the context of finite element methods (FEM), particularly for estimating the error between the computed and exact solutions of a given partial differential equation (PDE).

When working with traditional tools like pen and paper for proving mathematical statements, we might use shorthand or make intuitive leaps that rely on assumed knowledge or assumptions that are not explicitly stated. This includes using abbreviations or assuming certain mathematical properties without explicitly proving them at every step. These could create gaps in the proof where not every logical step is fully detailed. However, when we use a proof assistant (like Coq), every element of the proof must be explicitly defined and every step must be logically justified. This means that the user cannot simply assume or omit details; they must specify which definitions are being used and how the parts of the argument fit together. Let us illustrate with a simple example, in the case of the intermediate value theorem. It is essential that $a \leqslant b$, where a and b are the endpoints of the interval $[a, b]$ for the function. This theorem states that if we have a continuous function on an interval from a to b, then for any value d between $f(a)$ and $f(b)$, there exists some point c within that interval where $f(c) = d$. However, if we overlook specifying that $a \leqslant b$ in a traditional, informal proof, we might incorrectly apply the theorem to the case where $a > b$, which does not make sense. In a formal proof system like Coq, one must clearly define each condition, including $a \leqslant b$, before proving the theorem. This requirement ensures that all necessary conditions are met and prevents logical errors from missing assumptions. This method ensures there are no gaps because the computer requires complete clarity and will point out any missing parts. To this end, it is easier to start with a pen-and-paper proof that acts as a draft that guides the formal verification process, which allows for more freedom to sketch out ideas and explore how different parts of the proof interact without the constraints of formal syntax. Then, once the rough ideas are clearly understood and laid out, the next step is to organize and formalize each part of the proof in a proof assistant.

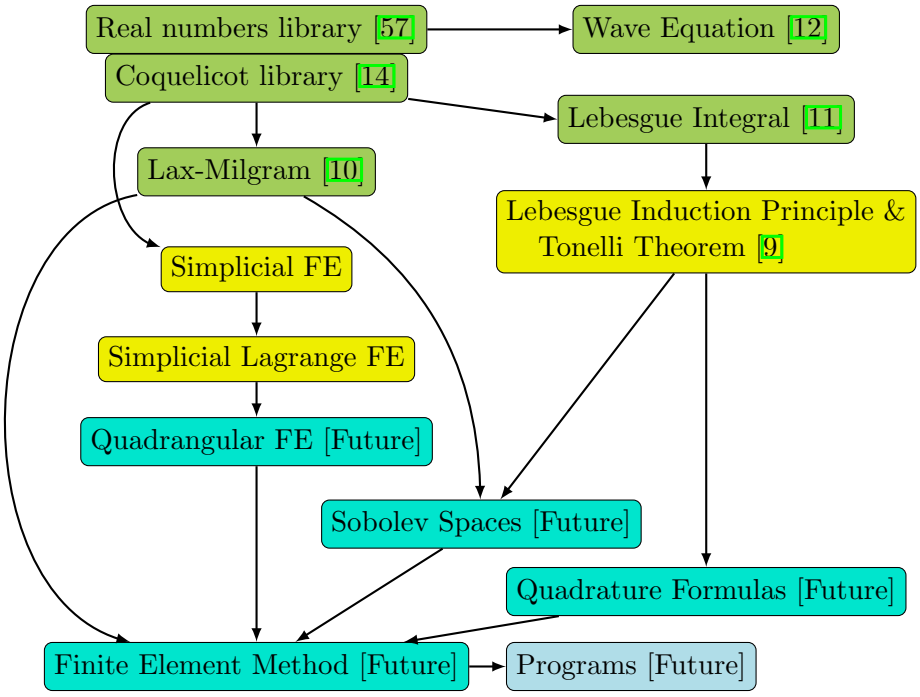

Figure 1.1: A diagram illustrating a chronological and thematic roadmap of the Coq formalization of mathematical concepts, distinguished by color-coded stages of completion. The green boxes denote prior work. The yellow boxes, which are completed, represent the focus of the thesis. The turquoise blue boxes outline anticipated future work. The light blue box sets a long-term goal.

1.2 Motivations of the Thesis

My thesis builds upon a structured series of prior developments, as illustrated in the work diagram in Figure 1.1. The foundation of these developments began with the creation of a real numbers library in 2001 [57], and the Coquelicot library [14] as its conservative extension in 2015. This groundwork enabled the first full formalization and proof of a numerical C program designed for the simulation of the wave equation, published in 2013 [12, 13]. Specifically, the finite difference scheme was addressed for the simulation of the wave equation, a method suitable for a restricted class of problems, particularly those constrained by the geometry of the domain. The next phase of this work aims to adopt a more powerful tool, the finite element method, which is based on a stronger theoretical foundation. To implement this method, an existence and uniqueness result for solutions to a class of partial differential equations (PDEs) is needed, here the Lax-Milgram theorem. This theorem, which operates within a general Hilbert space, a complete vector space with an inner product, was formally proven in 2017 [10]. In addition, establishing the appropriate context for applying the Lax-Milgram theorem requires the development and formalization of Sobolev spaces, which involves first constructing the Lebesgue integrals. The formalization of Lebesgue integration for non-negative measurable functions was published in 2021 [11], with future work focusing on the development of Lebesgue and Sobolev spaces. Building on this substantial groundwork, my thesis is underpinned by two primary motivations that collectively ensure the correctness of essential concepts in functional and numerical analysis using formal proofs within the Coq proof assistant.

The first part of the thesis focuses on the Coq formalization of two main mathematical

concepts in functional analysis, specifically, the Lebesgue induction principle and the Tonelli theorem. This work has been published in 2023 [9]. The Lebesgue induction principle serves as a proof technique for properties associated with nonnegative measurable functions, particularly those involving integrals. It reflects the three construction steps followed by Henri Lebesgue to build his integral [53]. The principle unfolds in three steps: initially, the property is established for indicator functions. Subsequently, it extends to nonnegative simple functions by verifying its compatibility with positive linear operations. Finally, the property applies to all nonnegative measurable functions by verifying its compatibility with the supremum operation. This principle is derived from an inductive type and emerges as a tool for proving the Tonelli theorem, as elaborated in more detail in Section 4.1.

The Tonelli theorem offers a convenient method for simplifying the computation of multiple integrals by establishing their equality with iterated integrals, each occurring in a single dimension, thereby facilitating the interchangeability of integration orders. This theorem is applicable specifically to nonnegative measurable functions. It aligns with the Fubini theorem, a related result that extends its scope to integrable functions with arbitrary signs. In the context of this thesis, the focus is on nonnegative functions with two variables, as detailed in Section 4.3. This part of the thesis lays the groundwork for future explorations when we will deal with PDEs that involve multiple variables.

Furthermore, the second part of this research, which represents the main focus of this thesis, delves into the Coq formalization and the construction of finite elements that represent the cornerstone of the Finite Element Method (FEM). This latter is a widely used numerical technique for finding approximate solutions to boundary value problems for partial differential equations (PDEs) arising across various scientific domains, like physics, mechanics, and biology. It can be applied to a wide range of problems, including for instance structural analysis (stress, deformation), thermal analysis (heat transfer), fluid dynamics, and electromagnetic potentials, among others. This extensive use and success of FEM by numericians is largely attributed to its flexibility and precision in handling problems with complex geometries, discontinuities, and constraints common in engineering and physical sciences, which are explored and cited throughout the literature, see for instance [19, 35, 66, 73]. Despite the wide applications of the FEM, several critical challenges can adversely affect its efficiency and reliability [4]. The process of setting up simulations, executing them, and analyzing the results is often extremely time-consuming, especially for complex simulations. Furthermore, FEM programs are typically lengthy and complex, requiring high attention to detail. Even minor errors in implementation can result in significant contradictions in the results, compromising the validity of the simulation outcomes and potentially leading to incorrect conclusions. To address these issues, formal proofs are employed to ensure the correctness and reliability of the algorithms and methods used in FEM simulations.

More specifically, we aim to thoroughly formalize a generic finite element (FE), which is mathematically defined as a triplet and formalized as a record as detailed in Section 7.1. The FEM subdivides a large problem into smaller, simpler parts that are called *finite elements*, typically connected by points called *vertices*, as depicted in Figure 1.2. Each element is characterized by a triplet consisting of three complementary components, denoted as $(K, \mathcal{P}, \Sigma)$.

The component K represents the geometric element for the finite element, which varies in form from intervals in one-dimensional spaces to more complex shapes like triangles or quadrilaterals

[1] https://omni.wikiwand.com/fr/articles/Maillage

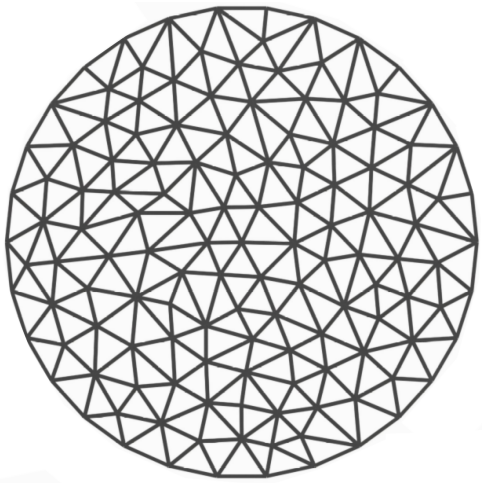

Figure 1.2: Visual representation of a 2D disk approximated with a mesh made of triangles[1]. We note that the boundary of the mesh is not perfectly circular. A triangular mesh is made up of triangular elements that are connected at their vertices and faces (see Section 6.2.2).

in two-dimensional spaces, and tetrahedrons or cuboids in three-dimensions, depending on the complexity of the domain and the requirements of the simulation. For the purposes of this thesis, the focus will be exclusively on working with *simplices*, which are a generalization of the concept of a triangle or tetrahedron to arbitrary dimensions (see Equation (6.6) in Section 6.2.2). The second component $\mathcal{P}$ of the finite element triplet, represents a finite-dimensional vector space of polynomial functions defined on the geometric element K. These polynomials are used to approximate the solution within the finite element, where the degree of the polynomial (e.g. linear, quadratic) determines the accuracy of the approximation. The third component Σ of the triplet, corresponds to a family of linear forms $(\sigma_i)_{i\in[0..n_{\text{dof}}]}$, also known as local degrees of freedom, which act on the functions within the space $\mathcal{P}$. These forms are chosen to satisfy the *unisolvence property*, ensuring the mapping $p \in \mathcal{P} \longmapsto (\sigma_i(p))_{i\in[0..n_{\text{dof}}]}$ is bijective. However, addressing the problem for individual elements alone is not sufficient. The Finite Element Method (FEM) assembles the local equations from each finite element into a global system of equations that represents the entire problem. By solving this global system, we obtain an approximation of the solution that spans the entire problem domain.

The focus of this last part of the research is the construction of the finite elements of the *simplicial Lagrange* family [36, Section 7.4 p.78-81] using the Coq proof assistant. The main objective of this development is to establish a useful finite element, and also ensure that our definitions for finite elements are sufficient and valid to be used. The choice to formalize Lagrange FE for this study stems from its widespread use in FEM applications and its inherent simplicity. This work further delves into the formalization of definitions and proofs concerning essential properties such as the Lagrange polynomials, affine geometric transformations, and both current and reference finite elements, among others. To implement this, we will need to use real analysis and algebraic structures ranging from abelian monoids to module spaces of the Coquelicot library [14], as detailed in Section 2.2. Additionally, we will use various algebraic properties described in Chapter 5, including definitions of `fintype` and `bigop`, which are part of the Mathematical Component library [56]. Each of these components contributes towards the primary aim of this thesis, which is to formally establish the *unisolvence property* of the Lagrange FE, as detailed in Section 10.3.7. Additionally, future directions include the formalization of quadrangular finite elements, some quadrature formulas, and the construction

of Sobolev spaces. Thus, the collective efforts described in this work are directed towards a broader long-term goal, which is the formal verification of scientific computing programs and parts of C++ libraries like FreeFEM++[2] and XLiFE++[3], which implement the FEM.

The mathematical definitions and proofs in this research are derived from a variety of textbooks, specifically [34, 35, 36, 41], and primarily based on [20, 21]. The Coq code is available in the repositories Lebesgue and FEM at:

https://lipn.univ-paris13.fr/coq-num-analysis

It is also available in the Opam package (version ≥ 2.0) at (search with category "Mathematics/Reals Calculus and Topology"):

https://coq.inria.fr/coq-package-index

1.3 Related Work

This section reviews various formalizations relevant to the subject of the thesis. Beyond the Reals library, there are alternative approaches to developing a library of real numbers. One such alternative is the CoRN (Constructive Reals and Numbers) library [26], which is available within the Coq system and is designed for the constructive formalization of real numbers. In addition, the Lebesgue integral and in particular the Tonelli theorem, a fundamental result in measure theory, have been extensively formalized across different proof assistants. The formalizations that align closely with our work appear in Isabelle/HOL [62] and Lean [29]. In 2007, David Lester formalized the Tonelli theorem within the PVS-NASA library[4], building on his earlier work [54]. In 2011, Johannes Hölzl and Armin Heller developed foundational concepts in measure theory in Isabelle/HOL [46], including the formalization of the binary and iterated product measures, along with the Lebesgue integral and proving the Fubini theorem. Moreover, a significant work occurred in 2019, Noboru Endou covered the formalization in Mizar [33] of various aspects necessary to establish the Fubini theorem, including product measures, measurable functions, and integration over product spaces. Another related work, Floris van Doorn's contributions to Lean Mathlib in 2021[5] [71] include the integration of functions with respect to product measures, the existence and uniqueness of product measures, along with the measurability and integrability of functions in product spaces, particularly the formal proof of the Tonelli and the Fubini theorems in a way similar to our, but for the Bochner integral, an extension of the Lebesgue integral of functions taking values in a Banach space. He also introduced a Lebesgue induction principle, which bears some resemblance to our method; however, to our knowledge our approach of deriving it from an inductive type represents a novel contribution. The construction and the Coq formalization of Fubini's theorem were developed by Affeldt Reynald and Cyril Cohen in 2023 for functions of arbitrary sign [2] within the math-comp Analysis library[6]. In contrast, our work is limited to nonnegative functions.

Now, in terms of contributions to the finite element method framework, specifics on finite elements have not yet been studied elsewhere up to our knowledge. However, there have

[2] https://freefem.org/
[3] https://uma.ensta-paris.fr/soft/XLiFE++/
[4] https://github.com/nasa/pvslib/blob/master/measure_integration/fubini_tonelli.pvs
[5] https://leanprover-community.github.io/mathlib4_docs/Mathlib/MeasureTheory/Constructions/Prod/Integral.html
[6] https://github.com/math-comp/analysis/blob/master/theories/lebesgue_integral.v

been some developments in the areas of polynomial interpolation and mesh construction. Our research primarily focuses on working with simplices. We focus on constructing finite elements on individual elements within the mesh. A mesh is a discretization of the domain into finite elements of varying geometric shapes, such as simplices or quadrilaterals. Some advancements have been made in the construction of entire meshes, and the geometry of individual elements.

The `SciLean` library involves computational geometry concepts within the Lean proof assistant. The development in this library seems still under construction, as there are admitted proofs throughout and numerous unfinished results, but it remains useful for calculating approximate solutions to partial differential equations. It includes definitions and properties for handling the geometric construct of *prism*[7], allowing the representation of various geometric shapes such as simplices, cubes, and pyramids through induction. Furthermore, the `SciLean` library proceeded in constructing a triangular mesh[8], as the one depicted in Figure 1.2, from a list of triangles by identifying unique edges and organizing them along with their corresponding triangles. Following this, the calculation of the barycenter and the barycentric coordinates for points within the prism have been presented in the `SciLean` library as well. In our Coq work, the implementation of barycenters is applicable to any family of points in an affine space, regardless of the specific geometric shape these points might form. Whether the points form a simplex, a quadrilateral, or any other polygonal/polyhedral shape, the barycenter can be computed as long as we have a representation of these points in the affine space and a set of weights (i.e., the barycentric coordinates when scaled to sum one).

Additionally, the `SciLean` library defines and implements Lagrange polynomial bases, such that at each node of an element, the function has a value of 1 at one node and 0 at all others, providing a simple way to interpolate values across prisms. In our work, however, we extend the formalization and definition of these polynomial bases to encompass all geometric shapes in the form of simplices within a real affine space. Another formalization similar to our work of Lagrange polynomial bases has also been undertaken in `mathlib` Lean[9], that offers a thorough study in addition to the one in `SciLean`, by emphasizing the definition of the Lagrange polynomial bases, along with the Lagrange nodes for the interpolation process, supported by various theorems to validate that the Lagrange polynomial will exactly match the provided value at each specified point.

Moreover, in 2010, Laura I. Meikle and Jacques D. Fleuriot focused on enhancing two-dimensional planar geometry using `Isabelle/HOL`, such as convex hulls and Delaunay triangulations [58]. This includes basic geometric definitions such as points, and orientation of points (collinearity, betweenness). Additionally, Yves Bertot and Jean-François Dufourd, focused on the formal verification of an algorithm for constructing Delaunay triangulations of a mesh using the Coq proof assistant [31]. This algorithm was implemented to construct triangles such that none of the input points lie inside the circumcircle of any triangle. In a more recent work, published in 2018, Bertot extended his earlier work in [6], by addressing a broader class of triangulation algorithms, that guarantees that the resulting mesh satisfies important geometric properties, such as covering the convex hull and maintaining boundary edges. In our development, we specifically focus on viewing a given mesh cell as the convex hull of its affinely independent vertices, and subsequently constructing a finite element on this cell.

[7] https://github.com/lecopivo/SciLean/blob/fea6cf7/SciLean/Data/Mesh/PrismRepr.lean
[8] https://github.com/lecopivo/SciLean/blob/fea6cf7/SciLean/Data/Mesh/TriangularMesh.lean
[9] https://leanprover-community.github.io/mathlib4_docs/Mathlib/LinearAlgebra/Lagrange.html

There have been various formal developments in the domain of ordinary differential equation (ODE) and partial differential equation (PDE) resolutions, including [47, 69, 30], but these are beyond the scope of our work. In this work, we do not aim to formalize the Finite Element Method itself; that will be addressed in future efforts. Instead, our current focus is on formalizing finite elements, which will be useful in future formalizations of ODE and PDE resolutions.

1.4 Organization of the Thesis

The thesis is organized into two main parts, each comprising several distinct chapters. Chapters 2, 3 and 6 present the formal and mathematical needed contents to offer valuable context and insight into the work pertinent to this study. I have contributed to the remaining chapters, with minor contributions in Chapter 5.

Chapter 2 introduces the Coq proof assistant and some of its supporting libraries, including the `Reals` standard library, Coquelicot which enhances Coq's handling of real numbers, and the math-comp library. It explains the differences between constructive and classical logic within Coq, and highlights its type inference capabilities. Additionally, the chapter also addresses the limitations of the Reals standard library, presents an approach to structuring algebraic hierarchies by canonical structures.

Part I focuses on the formalization of the Lebesgue induction principle and the Tonelli theorem.

Chapter 3 outlines essential concepts from [11] developed in 2021 that are necessary for understanding this part, focusing on the Coq formalization of the Lebesgue integral. This chapter introduces fundamental measure theory concepts, including the measurability of subsets and functions in Section 3.1, a formal definition of measure in Section 3.2 along with its properties, the canonical representation of simple functions in Section 3.3, and their integration process. Additionally, it covers the integration of nonnegative measurable functions in Section 3.4. As I did not contribute to this work, I have included only pertinent details to provide clarity and context for this thesis.

Chapter 4 delves into the formalization of the Lebesgue induction principle (LIP) and the Tonelli theorem. This chapter represents my initial contributions during my PhD, starting with the formal proof of LIP in Section 4.1, which is a proof technique for dealing with properties related to nonnegative measurable functions. Subsequently, Section 4.2 is dedicated to the construction of product measures within a product space, including the development of the corresponding product σ-algebra and the properties of the section of subsets. Furthermore, Section 4.3 is devoted to the construction of iterated integrals and the formalization of the proof of the Tonelli theorem. This theorem focuses on the study of double integrals and the interchangeability of integration orders, highlighting the ability of the Lebesgue integral in handling multi-variable functions.

Part II addresses the formalization of simplicial finite elements.

Chapter 5 outlines the formalization of algebraic concepts and properties that underpin the subsequent chapters on finite elements. It leverages several libraries, such as the math-comp library for handling finite types, iterated operators, and binomial coefficients, and the Coquelicot

library to incorporate new algebraic structures, such as affine spaces, into its hierarchy. The development described in this chapter draws inspiration from the works of Gostiaux [41]. However, my contributions to the formalizations described in this chapter were very minor. Section 5.1 explores complementary aspects of functions, with a particular focus on functions on subsets. Section 5.2 addresses support for finite families of any type. Discussions of algebraic structures are presented in Section 5.3. The study of finite dimensional subspaces is specifically addressed in Section 5.4. Finally, Section 5.5 offers a concise discussion on binomial coefficients.

Chapter 6 provides a purely mathematical overview of the finite element method, which is a widely-used computational technique in engineering and mathematical modeling for approximating solutions to boundary value problems associated with partial differential equations (PDEs). It studies a second-order linear elliptic PDE within a d-dimensional domain, referred to as the *Poisson problem* (6.1) in Section 6.1, and discusses its transition from the strong formulation to the weak formulation. Next, the chapter details the development of the related linear system through discretization, emphasizing the construction of the mesh in Section 6.2, as illustrated in Figure 1.2. Subsequently, the chapter introduces a comprehensive definition of finite elements as a triplet in Section 6.3, along with a detailed explanation of the principle of unisolvence in Section 6.4, which is necessary to ensure that the studied problem has a unique solution. While not all aspects discussed in this chapter are formalized using Coq, it lays the foundational steps necessary for the numerical implementation of these principles. This chapter is essential for understanding the concepts explored in subsequent chapters.

Starting from Chapter 7 onwards is my contribution to the subject of the thesis. Chapter 7 builds upon the mathematical foundations delineated in the previous chapter. It starts with a detailed discussion on the Coq formalization of a general finite element in Section 7.1, represented as a triplet within individual cells of the mesh using a record-based approach. This is followed by Section 7.2 which defines the local shape functions as a basis for the polynomial approximation space $\mathcal{P}$. The chapter concludes with Section 7.3, focusing on the construction of a local interpolation operator associated with this finite element.

Chapter 8 delves into the construction of the polynomial space $\mathcal{P}_k^d$, which comprises polynomials of degrees less than k on $\mathbb{R}^d$. This chapter begins with an in-depth discussion of multi-indices in Section 8.1, essential for the construction of multivariate monomials. This section introduces the definition of the $\mathcal{A}_k^d$, $\mathcal{C}_k^d$, and $\check{\mathcal{S}}_{k,k-i}^d$ families, explores the ordering of multi-indices, and studies the bijectivity of $\mathcal{A}_k^d$. Section 8.2 then moves on to define the polynomial space $\mathcal{P}_k^d$ as the linear combinations of monomials, detailing its canonical basis $\mathcal{B}^{d,k}$, establishing the linear independence of this family, and reviewing some key properties of $\mathcal{P}_k^d$ spaces. The discourse narrows down further in Sections 8.3 and 8.4, where the focus shifts to specific polynomial spaces: $\mathcal{P}_1^d$ and $\mathcal{P}_k^1$, studying cases where the spatial dimension d or the degree of approximation k is equal to 1.

Chapter 9 starts with a study of simplicial geometric aspects in Section 9.1, where it defines reference vertices, reference and current Lagrange nodes, and establishes connections between vertices and nodes, as well as introduces sub-vertices and sub-nodes. Section 9.2 discusses the $\mathcal{P}_1^d$ Lagrange polynomial bases on the reference geometric element when the degree k is equal to 1, along with several key properties of these polynomials. Section 9.3 delves into the affine geometrical transformation of finite elements that map the reference element to current elements and its inverse function, setting the stage for Section 9.4, which outlines the process of constructing current finite elements (FEs) from the reference FE, including the formulation

of current shape functions and local interpolation operators. Section 9.5 elaborates on the $\mathcal{P}_1^d$ Lagrange polynomials basis on a current geometric element when $k = 1$, while Section 9.6 focuses on $\mathcal{P}_k^1$ Lagrange polynomial bases on a segment when the spatial dimension d is equal to 1. We thoroughly examine how these polynomials are formalized in Coq and used across both reference and current geometrical elements.

Chapter 10 focuses exclusively on constructing simplicial Lagrange finite elements, beginning with a formal definition of face hyperplanes in Section 10.1. Section 10.2 explores geometric mappings, detailing the process of transforming nodes from the reference geometric element in dimension $d - 1$ to an hyperplane of the current geometric element. The focus then shifts to the construction of current simplicial Lagrange finite elements within Coq in Section 10.3, which covers nodal linear forms, specifics of the triplet of the Lagrange finite elements, and a comprehensive formal proof of the unisolvence property addressing specific cases where $k = 1$ or $k = 0$ and $d = 1$. This section also discusses some factorization properties of polynomials related to these elements. Finally, Section 10.4 is devoted to the construction of the reference simplicial Lagrange finite elements.

Chapter 11 concludes the thesis by summarizing the discussed topics and proposing directions for future research.

In this thesis, I try to keep code snippets straightforward and simple. As a consequence, these snippets may differ slightly from the actual implementation: details such as 'scopes' are omitted, and some arguments are replaced with underscores to enhance clarity and relevance.

Chapter 2

Coq and Support Libraries

This chapter provides an overview of Coq, a proof assistant used for verifying mathematical theorems and certifying programs. We explore the differences between constructive and classical logic within Coq, and highlight essential tools such as its type inference capabilities. The chapter presents the Coquelicot library, which enhances Coq's handling of real numbers. Finally, we examine an approach to structuring algebraic hierarchies by canonical structures, and we give a brief introduction to the math-comp library.

2.1 The Coq Proof Assistant

Coq [23] is an interactive proof assistant, designed for formalizing mathematical theorems and verifying proofs. It uses a special command language called the *Vernacular* to manage the logical content in a Coq file and organize proof development. *Gallina* is the core language of Coq used to express programs, types, and proofs. It is a *dependently typed functional language*, meaning that it is based on type theory, where types can depend on values. Additionally, Coq provides a *tactic language* to simplify the proof process by allowing users to construct proofs interactively, step by step, without having to write the proof terms manually. These tactics guide the proof towards completion, and Coq automatically generates the corresponding proof term when the proof is complete. The Coq's logic is based on an extension of the Calculus of Constructions (CoC) [24], originally designed in 1986, known as the Calculus of Inductive Constructions (CIC) [65]. CIC combines *type theory* (which includes functions, types, and type constructors) with *inductive types* to express both programs and proofs. Unlike *classical logic*, which is commonly used by most mathematicians and allows the use of principles like the law of excluded middle (`EM`), stating that every proposition is either true or false, Coq is based on *constructive logic*. This type of logic, also called *intuitionistic logic*, does not accept the law of excluded middle as universally valid. Instead, constructive logic emphasizes that the truth of a proposition depends on our ability to build a constructive proof for it.

One of the valuable characteristics of Coq is *type checking* that helps guarantee that proofs are correct and that functions behave as intended. Moreover, Coq provides a system of *type inference* that helps simplify the process of writing proofs and programs by automatically deducing the types of expressions without the user having to explicitly annotate everything. This makes the system much more concise and easier to use. For instance, if a user writes a function that takes a natural number, in an environment where the notations also involve integers, and returns its square, Coq can infer that the input type is `nat` without requiring the user to explicitly declare it. Without type inference, users would need to manually specify the type of every object or variable, which would quickly become tedious, resulting in difficult-to-read code. For this reason,

we primarily rely on *implicit arguments*, as making them explicit all the time would clutter the code and reduce clarity. In Coq, explicit arguments are the default, meaning that we must specify them every time we invoke the function or use the type. For example:

```
Definition add (x : nat) : nat := x + 2.
```

Here, + is a notation of `Nat.add`, and `x` is an explicit argument of the function `add`. An implicit argument, on the other hand, is an argument that Coq can automatically infer from the context. Suppose we want to define the identity function `id` that works for any type, Coq will automatically infer what type `A` should be, based on the argument provided for `x`. For instance:

```
Definition id {A : Type} (x : A) : A := x.
```

Here, `A : Type` is an implicit argument, denoted by the {} brackets around it.

In Coq, inductive types are established through one or more constructors that enable the creation of new instances of the type. These constructors allow the construction of every possible value of the type, starting from a base set of initial values. Inductive types can be used to define various structured data, such as natural numbers, lists, and trees. They often include *recursive definitions*, meaning the type being defined can refer back to itself. For instance, a simple inductive definition for a list type might feature a constructor for an empty list and another constructor that combines a head element with a tail, where the tail is itself a list. Moreover, working with inductive types usually involves *pattern matching*, where different cases corresponding to the constructors of the type are considered. There is another fundamental mechanism in Coq named `Records`, which are a specialized form of inductive types, thereby inheriting all the benefits associated with inductive types, including the ability to be used in pattern matching and recursive definitions. Essentially, records are single-constructor inductive types designed to group several related values together into a single value. Additionally, each field within a record can have a type dependent on the values of the previous fields. Moreover, records can employ coercion [68]; this is achieved by declaring one of the fields as a coercion with the syntax `:>`, enabling implicit conversion of this record type to another specified type when necessary.

A significant aspect of constructive logic is its approach to existential statements. In classical logic, to verify a statement of the form $\exists$ `x, P(x)`, known as *weak existential quantification*, it suffices to give the existence of a witness `x` for which `P(x)` is true, without specifying a program for constructing it. This form of existential statement is used when the goal is to establish the existence of an element as part of proving certain proposition, rather than to use this element directly to define functions. In contrast, constructive logic provides a more specific result: it requires not only demontrating the existence of an object, but also providing a concrete program for its construction. Within this framework, the existential statement is formalized in Coq as {`x | P(x)`}, known as *strong existential quantification*. In more details, to prove an existential statement in constructive logic, one must construct an explicit witness for `x` that satisfies the property `P(x)`. This approach ensures that the witness can be constructed and used in further computations or definitions. This process of constructing witnesses and programs reflects the *Curry-Howard correspondence*, a key feature of Coq's logic. The Curry-Howard correspondence is an equivalence between proofs in logic and programs in computer science. It is important to note that while a proof of a strong existential allows us to infer the proof of the corresponding weak existential statement, the reverse is not true except if we require an axiom (such as (v)).

Although Coq is primarily based on constructive logic, it remains flexible enough to handle classical reasoning when necessary. This flexibility means that any theorem provable in classical

logic can also be established in Coq, by incorporating classical axioms such as the law of excluded middle (EM) (here {P} + {¬ P}) or the double negation elimination (¬ ¬ P → P, where ¬ denotes logical negation operation). Despite the foundational role of constructive logic in systems like Coq, which offers significant advantages in specific areas of mathematics and computer science, we opted to use classical logic in our work. This decision was driven by two main factors: firstly, we aim to adhere to the traditional reasoning methods commonly used by mathematicians; secondly, formalizing real analysis within an exclusively constructive framework does not align with our objectives, as the formulation and proof of statements using constructive logic do not fit the approach we want to pursue in our work.

Below, we outline the axioms used in our development:

(i) **The Law of Excluded Middle:** The principle of EM is necessary in classical analysis, and is implemented in Coq as `classic_dec` and formally expressed as:

∀ P : Prop, {P} + {¬ P}.

This axiom asserts that for any proposition P, there exists either a proof of P or a proof of its negation ¬ P. Additionally, the + symbol signifies a sum type, also known as a disjoint union, which, in the context of logic represents a strong "or". This means that we can decide whether the proposition P is either true or false. For instance, this axiom can be applied to propositions involving real functions.

(ii) **The Axiom of Choice:** The *axiom of choice* is used in real analysis to select elements from sets without explicitly constructing a method for choosing. The provided Coq code snippet formalizes this axiom as follows:

∀ {A B : Type} (R : A → B → Prop),
(∀ x : A, ∃ y : B, R x y) → ∃ f : A → B, ∀x : A, R x (f x).

This axiom asserts the existence of a choice function that assigns to each set a chosen element from that set.

(iii) **Propositional Extensionality:** The propositional extensionality axiom states that if two propositions P and Q are equivalent, then they are also equal as propositions. Formally, the axiom is expressed as:

∀ P Q : Prop, P ↔ Q → P = Q.

Without propositional extensionality, proving equality between propositions is generally impossible. This axiom implies the proof irrelevance axiom which is stated as follows:

∀ (H : Prop) (p1 p2 : H), p1 = p2.

(iv) **Functional Extensionality:** The functional extensionality axiom states that two functions are equal if they produce the same outputs for all possible inputs.

∀ {A B : Type}, (f g : A → B), (∀ x : A, f x = g x) → f = g.

Without functional extensionality, proving equality between functions (a common task in real analysis) is generally impossible.

(v) **Constructive Indefinite Description:**

∀ (A : Type) (P : A → Prop), (∃ x : A, P x) → {x : A | P x}.

This axiom enables the transition from a *weak existence* (i.e., without computational content) to a *strong constructive existence*. It implies the previous axiom of choice through the following lemma `ChoiceFacts.constructive_indefinite_descr_fun_choice` from the Coq standard library.

In essence, we have chosen to adopt classical logic for real analysis, aligning with the traditional approach followed by most mathematicians. This approach includes the integration of the axioms previously discussed, which will be assumed throughout this thesis.

2.2 The Coquelicot Library

Upon installing Coq, users gain access to the standard library, which provides a set of theories that can be imported directly using the `Require Import` command. In this work, we employ the *standard library for real numbers* Reals [57], which is based on a classical axiomatization of real numbers developed in 2001. This library provides basic tools for dealing with real number operations and properties, including limits for sequences and functions, derivatives of functions, Riemann integrals, and a collection of theorems for these mathematical constructs. Additionally, we make use of the Coquelicot[1] library [14], an extension to the standard real numbers library, which enhances and expands the functionality related to real analysis. It offers more user-friendly handling of derivatives, integrals, continuity, and limits using filters.

2.2.1 Extended Real Numbers

While these features (i.e., derivatives, integrals, and limits) operate within the Coquelicot library in the context of real numbers $\mathbb{R}$, they are adeptly designed to address boundary behaviors involving the extended real numbers $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$. This extension is essential for handling scenarios where operations might exceed the finite limits of real numbers, reaching into the infinite boundaries ($\pm\infty$), such as in the evaluation of limits or integrals, for example, considering the limit of $1/x$ as x approaches zero from the positive side approaches $+\infty$. The formal representation of extended real numbers is provided in the Coquelicot library as follows:

```
Inductive Rbar : Set := Finite : R → Rbar | p_infty : Rbar | m_infty : Rbar.
```

In this context, $\overline{\mathbb{R}}$ is implemented as an *inductive type* in Coq, which can be understood as a data type allowing for multiple constructors. It is defined through three constructors: `Finite`, representing real numbers and serving as a canonical injection mapping real numbers to the extended real numbers. This constructor says that if we have a real number `R`, we can create an instance of `Rbar` by applying `Finite` to it. For example, `Finite` 3.14 creates an `Rbar` value representing the real number 3.14; the second constructor `p_infty` takes no arguments and creates a new instance of `Rbar` representing positive infinity ($+\infty$); and similarly, the last constructor `m_infty` creates an instance of `Rbar` representing negative infinity ($-\infty$). These constructors enable the expression of operations and properties that involve infinite quantities, which are otherwise not possible with standard real numbers. For example, considering the operation sup (supremum), which finds the least upper bound of a set, this operation can lead to infinite values depending on the set considered. In Coq code, this functionality is encapsulated in operations like `Rbar_lub` for the least upper bound of subsets and `Sup_seq` for the supremum of sequences.

Within the framework of extended real numbers, multiple properties for operations and order relations have been established. For instance, addition is defined as a total function for all pairs

[1] https://gitlab.inria.fr/coquelicot/coquelicot/

of extended real numbers. The indeterminate forms $\infty - \infty$ are arbitrarily mapped toward 0. Typically, lemmas concerning addition exclude these cases. Similarly, multiplication is a total function, and the indeterminate forms $0 \times +\infty$ are also assigned the value 0, aligning with common conventions applied to nonnegative real numbers in Lebesgue integral theory.

2.2.2 Total Functions

One of the challenges in the standard library of reals arises from the use of *dependent types*, i.e., types that depend on values, which can complicate the formalization of mathematical concepts like limits, integrals, and derivatives. For instance, consider the concept of *limits* in real analysis. A limit might only be defined when certain conditions (such as convergence) hold, and thus, if the limit does not exist (or if we cannot provide a proof), the limit function is said to be *partial*, i.e., it requires a proof to return a value. In Coquelicot, the aim is to reduce this burden by using *total functions*. A total function, as opposed to *partial functions*, is defined for all possible inputs in its domain and always returns a result, regardless of whether the proof exists, and handles the proofs separately when needed. Even if the limit does not exist for a particular function, the total function still provides a value. To help understand, let us consider an example of the inverse function (see eg, [57, p.18]). In the Reals library, the inverse function is defined as a total function, where it is not necessary to specify that the denominator must be nonzero. This condition is only necessary when simplifying expressions like $x \times \frac{1}{x}$. Coquelicot has extended this concept of total functions to other notions such as derivatives and more.

The fact that Coquelicot builds on top of the Reals standard library means that users do not need to choose between one or the other. Instead, they can use both libraries together, taking advantage of the best of both worlds. Coquelicot enhances real analysis without conflicting with or overriding the standard Coq definitions. In other terms, the definitions in Coquelicot are compatible with those in the Reals standard library, and users can easily switch between both libraries without needing to redefine basic concepts. This is particularly useful when importing or referencing modules from different parts of Coq's environment.

2.2.3 Algebraic Hierarchy

Furthermore, the Coquelicot library uses *canonical structures* to implement a hierarchy of *algebraic structures*, including commutative monoids and groups, rings, and module spaces (see Figure 2.1), using type inference and a series of records equipped with coercions. These structures are similar to *type classes* in other programming languages, such as `Haskell` [43], but are specifically designed for Coq's logical and functional programming environment. The resulting hierarchy allows Coquelicot to systematically organize and relate various algebraic concepts, making it easier to apply proofs and properties from one structure to another. Each algebraic structure builds upon the simpler ones as depicted in Figure 2.1; for instance, properties proven for abelian groups can be directly applied to other structures like rings, which are more complex but still share some foundational properties with groups without redundancy. This extension is possible because the ring structure includes the group structure as a part of its definition.

This is a high level method that uses the type system of Coq and its capabilities for structuring and abstracting algebraic concepts. *Abstraction* in this context means creating general definitions that can apply to many specific cases without modification. For example, once we define what it means to be a group abstractly, we can apply this definition to any

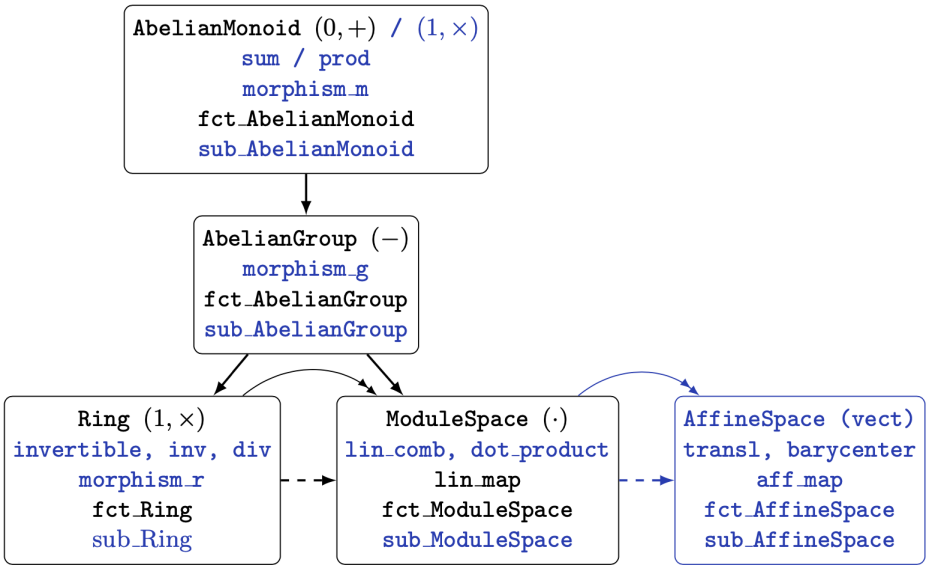

Figure 2.1: Hierarchy of algebraic structures used in this thesis. Additions to the Coquelicot library (version 3.4.0) and previous work [10] are in blue. For instance, finite iterations of operators using 'MathComps `bigop`, most morphisms, algebraic substructures, and the affine space structure have been added. Downward solid arrows specify inheritance, horizontal dashed arrows specify parameters, and the bent double-headed arrows mention that the input structure is shown to be an instance of the output one. For instance, any `ModuleSpace` (see Section 5.3) has the `AffineSpace` (see Section 5.3.6) structure over itself.

specific instance set and operation that meets the criteria. In simpler terms, suppose we define a canonical structure for a group in Coq, defined as a set G, along with a binary operation $+$ that satisfies four fundamental properties: closure, associativity, identity, and invertibility (refer to Section 5.3.3). If we prove a property or theorem for a generic group $(G, +)$ in Coq, the system can apply this theorem to any specific *instance* of a group, such as the real numbers equipped with addition $(\mathbb{R}, +)$ without requiring reproof.

To broaden the understanding of algebraic structures and illustrate the role of canonical structures within Coq, the diagram referenced in the Figure in 2.1 illustrates a sophisticated hierarchy of algebraic structures used in the present developments, delineating how foundational concepts like *abelian monoids* evolve into more complex entities such as *rings*, *module spaces*, and *affine spaces*, each building upon the properties and operations of the ones above it. At the base, we have *abelian monoids*, characterized by a single associative binary operation (either addition or multiplication) that is commutative and includes an identity element. These monoids extend into *abelian groups* by incorporating inverses for each element. The next level up is *rings*, which enhance *additive abelian groups* by adding a second associative operation, multiplication, that distributes over the addition operation. Moving further, *module spaces* generalize the concept of vector spaces, allowing operations involving scalars from a *ring*, thus extending the functionalities of *groups* with operations like scalar multiplication, linear combinations, and dot products. At the pinnacle of the hierarchy in Figure 2.1 are *affine spaces*, which abstract points and translations from *vector spaces*, focusing on geometric transformations rather than merely algebraic operations. Each level in this hierarchy introduces morphisms that are compatible with all operations of the algebraic structure called linear mappings for module spaces, and affine mappings for affine spaces. This emphasizes the flexible nature of algebraic structures, allowing properties and proofs developed for simpler structures to be extended to more complex ones. Detailed discussions on these algebraic structures will be presented in Section 5.3.

2.3 The math-comp Library

The math-comp[2] library [56], short for Mathematical Components, represents a collaborative contribution within the Coq community. This comprehensive library encompasses the formalization of a wide range of mathematical properties, with special emphasis on algebraic structures such as groups, rings, fields, and beyond. It also develops finite types such as ordinals, which are essential for managing families with a finite number of elements. Furthermore, the math-comp library features a module `bigop` for the formalization of finitely iterated operations in monoids like sums and products. Additionally, math-comp supports combinatorial structures, including binomial coefficients. All the properties mentioned will be necessary for our developments that will be discussed in detail in Chapter 5.

Bounded Natural Numbers

Bounded naturals are represented by a finite type, denoted as `'I_n`, which includes all natural numbers from the set $[0..n-1]$. This is formally defined within math-comp as an inductive type, referring to ordinals:

```
Inductive ordinal (n : nat) := Ordinal : ∀m : nat, m < n → 'I_n.
```

The construction of a value of type `ordinal` requires providing a proof that $m < n$. In this context, n represents the upper bound of the set `'I_n`.

The finite type `'I_n` coerces to `nat`, which means that elements of this finite type can be automatically treated as natural numbers within Coq. This feature allows all functions and operations defined for natural numbers to be seamlessly applied to elements of `'I_n` without needing to manually convert these elements into natural numbers. The cardinality of `'I_n` is exactly `n`. This means that the set contains `n` distinct elements, and that `'I_0` is empty.

We begin by introducing the basic concept of the ordinal `ord0` corresponding to the value 0 in the set `'I_n.+1`. It is defined as the smallest element in the finite type `'I_n.+1`, which represents the set `[0.. n]`, where `n.+1` is a notation for `S n` (the successor of `n`). We require the upper bound to be `n.+1` as `'I_0` is empty and does not contain 0. The formal definition of `ord0` is substantiated by the lemma `ltn0Sn`, which proves that 0 is strictly less than `n.+1` for any natural number `n`.

```
Definition ord0 : 'I_n.+1 := Ordinal (ltn0Sn n).
```

In addition to identifying the smallest ordinal, we also present the largest ordinal in a finite set. The value `ord_max` serves this purpose, representing the maximum ordinal in the set `'I_n.+1`, which includes every natural number up to `n`. This is achieved by setting the value of `ord_max` to `n` in `'I_n.+1`, and using the lemma `ltnSn`, which formally proves that `n` is strictly less than `n.+1`.

```
Definition ord_max : 'I_n.+1 := Ordinal (ltnSn n).
```

Let us now explore some specific utility functions and properties designed to manipulate and transform ordinal types.

The operation of converting an ordinal from the set of size n to the set of size m, provided that $n = m$, is defined as follows:

```
Definition cast_ord : ∀n m : nat, n = m → 'I_n → 'I_m.
```

[2] https://github.com/math-comp

Moreover, "widening" an ordinal involves adjusting the type of an index when transitioning from a set with a smaller number of elements `'I_n` to a larger one `'I_m`, given that `n` $\leqslant$ `m`. This is given as:

```
Definition widen_ord : ∀n m : nat, n ⩽ m → 'I_n → 'I_m.
```

This function guarantees that the ordinal maintains its original value despite the increase in set size. For example, consider `n` $= 3$ and `m` $= 5$, with a proof that `H: n` $\leqslant$ `m`. If we take an ordinal `i` in `'I_3`, corresponding to the value 2 (noting that $2 < 3$), then the `widen_ord H i` is an ordinal in `'I_5`, still representing the number 2, satisfying $2 < 5$.

On the other hand, another important function in math-comp, referred to as `lift`, not only transfers an ordinal from a smaller set to a larger one but also alters its value. This function is mathematically expressed as follows:

$$\forall i \in [0..n-1], \forall j \in [0..n-2], \quad \texttt{lift}\ i\ j = \begin{cases} j & \text{if } j < i \\ j+1 & \text{else.} \end{cases}$$

This is given in Coq as:

```
Definition lift : ∀n : nat, 'I_n → 'I_n.−1 → 'I_n.
```

For example, consider `i`, the ordinal in `'I_6`, having a value of 2 (since $2 < 6$), and the ordinals `j1` and `j2` in `'I_5` with values 1 and 3 respectively (note $1 < 5$ and $3 < 5$). Then, `lift i j1` yields an ordinal in `'I_6` with value 1 (since $1 < 2$ and $1 < 6$), and `lift i j2` results in an ordinal in `'I_6` of value $3 + 1 = 4$ (since $3 \geq 2$ and $4 < 6$).

Part I

Formalization of The Tonelli Theorem

Chapter 3

Lebesgue Integration Theory

This chapter summarizes the necessary concepts from [11] needed for the continuation of this work and delves into a formalization of the Lebesgue integral using the Coq proof assistant (see Section 2.1). The Lebesgue integral was chosen over the Riemann and gauge (Henstock–Kurzweil) integrals [5] for several reasons. For instance, the Riemann integral does not support that $\int \lim f_n = \lim \int f_n$, a property needed for proving that L^p spaces are complete. Additionally, the gauge integral does not easily extend to functions valued in $\mathbb{R}^n$, $\mathbb{C}^n$, or Banach spaces, which is a limitation for our purposes.

In this thesis, we use the Lebesgue integral [70] for the full formal proof of the Tonelli Theorem, as detailed in Chapter 4. This theorem focuses on the study of double integrals and their exchanges, highlighting the ability of the Lebesgue integral in handling multi-variable functions. The use of the Lebesgue integral is further anticipated to extend into the domain of numerical analysis, especially in solving partial differential equations (PDEs) through the finite element method (FEM). Moreover, its application is expected to contribute to the development of Sobolev spaces [1]. These spaces are relevant in diverse areas, including functional analysis (e.g. see [17]) as well as statistical and probabilistic mathematics (e.g. see [37]).

The Lebesgue integral is constructed by initially defining the integral for indicator functions corresponding to measurable subsets. This definition is then extended to simple functions, which consist of finite linear combinations of indicator functions, thereby expanding the concept of integration to all nonnegative measurable functions by taking the supremum. We refer to this method as the *Lebesgue scheme*. In chapter 4, we introduce the *Lebesgue induction principle*, a tool, established further in Section 4.1, to prove properties about nonnegative measurable functions that follow the Lebesgue scheme.

This chapter introduces the concepts of measure theory (e.g. see [22]). Initially, Section 3.1 discusses the measurability of subsets and functions. Section 3.2 presents a formalization of measure in Coq and outlines its properties. Following this, Section 3.3 elaborates on the canonical representation of simple functions and their integration. The chapter concludes with Section 3.4, focusing on the integration of nonnegative measurable functions.

3.1 Measurable Space

A measurable space is a tuple (X, Σ), where X is a set, often referred to as the *universe* or *space* that can vary from the real number line $\mathbb{R}$ to more complex spaces, and the component Σ represents a σ-algebra on X. Specifically, a σ-algebra is a collection of subsets from the power

set, denoted as $\mathcal{P}(X)$, that includes the empty set and is closed under the operations of taking complements and countable unions. This power set $\mathcal{P}(X)$ is defined as the set of all possible subsets of X, including the empty set and X itself. Additionally, the subsets within Σ are called measurable subsets, as detailed in Section 3.1.1. Moving further, Section 3.1.2 is devoted to the cartesian product spaces, formed by the product of two measurable spaces. Lastly, Section 3.1.3 is focused on the measurability of functions.

3.1.1 σ-algebra and Measurability of Subsets

In Coq, we pose a generic set `X` as `X : Type`. The subsets of `X` can be represented by the type `X → Prop`, which leads to defining the power set of `X`, of type `(X → Prop) → Prop`. In general, listing every subset within a σ-algebra can be complicated due to its potentially vast size. Instead, we specify a smaller collection of subsets known as generators, denoted by G, and represented in Coq by `genX : (X → Prop) → Prop`. From these generators, the σ-algebra is derived through operations including taking complements, countable unions, and intersections (refer to [20, Section 8.5]). The smallest σ-algebra containing these generators G, is referred to as a *generated* σ-algebra and symbolized by $\sigma(G)$. It is often sufficient to confirm a particular property for these generators to deduce that the same property holds for the entire σ-algebra they generate.

The formalization of σ-algebras in Coq is defined as an inductive type, characterizing *measurable* subsets. Considering a set of generators `genX`, a subset is called *measurable* if it meets one of the following properties: it is a generator, it is empty, it is the complement of a measurable subset, or it is the countable union of measurable subsets.

```
Inductive measurable : (X → Prop) → Prop :=
  | measurable_gen : ∀A : X → Prop, genX A → measurable A
  | measurable_empty : measurable (fun _ ⇒ False)
  | measurable_compl : ∀A : X → Prop, measurable (fun x ⇒ ¬ A x) → measurable A
  | measurable_union_countable : ∀(A : nat → X → Prop),
      (∀ n, measurable (A n)) → measurable (fun x ⇒ ∃ n, A n x).
```

In this context, `(fun x : X ⇒ expr)` defines a lambda term. It means a function that takes an input `x` of type `X` and returns the value of the expression `expr`.

Building upon this definition, several lemmas have been established, such as the measurability of the entire set X (i.e., $X \in \Sigma$), and that of the intersection of countable subsets (i.e., $\bigcap_{n\in\mathbb{N}} A_n \in \Sigma$), within a σ-algebra (see [11, Section 4.2 p.11]). Formally, a σ-algebra is defined as a subset of the power set that coincides with the σ-algebra induced by itself as a generator,

```
Definition is_sigma_algebra : ((X → Prop) → Prop) → Prop :=
  fun calS ⇒ calS = measurable calS.
```

This definition aligns with one of the commonly used mathematical definitions: a collection $\mathcal{S}$ is a σ-algebra when it contains the empty set, and is closed under complement and countable unions. The Coq implementation confirms this equivalence through the following lemma,

```
Lemma is_sigma_algebra_correct : ∀calS : (X → Prop) → Prop,
   is_sigma_algebra calS ↔
    (calS (fun _ ⇒ False) ∧
    (∀ (A : X → Prop), calS (fun x ⇒ ¬ A x) → calS A) ∧
    (∀ (A : nat → X → Prop), (∀ n, calS (A n)) → calS (fun x ⇒ ∃ n, A n x))).
```

For instance, when the measurable space has a topological space structure, the σ- algebra typically chosen is the Borel σ-algebra. It is the smallest σ-algebra generated by all open subsets.

In the context of $\mathbb{R}$, examples of such open subsets include open intervals like (a, b), where a and b are real numbers with $a < b$.

3.1.2 Cartesian Product Space and Measurability

When considering two measurable spaces, namely (X_1, Σ_1) and (X_2, Σ_2), along with their corresponding generators G_1 and G_2, determining the measurability of the cartesian product $X_1 \times X_2$ must be carefully done in terms of choosing the appropriate σ-algebra. A key concept in this context is the product σ-algebras (see Section 4.2.2), which is especially significant as it ensures the measurability of the canonical projection, these are the mappings $((x_1, x_2) \mapsto x_1)$ and $((x_1, x_2) \mapsto x_2)$ (for further details on measurable functions, refer to Section 3.1.3). The product σ-algebra is generated by the Cartesian products of measurable subsets from X_1 and X_2.

However, when it comes to identifying a suitable generator for this σ-algebra, simply considering the Cartesian products of elements from G_1 and G_2 proves to be wrong. For example, this approach fails to establish the measurability of sets like $A_1 \times X_2$, where A_1 belongs to G_1. To address this issue, it is necessary to include the full sets in the initial generator sets. This is achieved through the following definition in Coq,

```
Variable genX1 : (X1 → Prop) → Prop.
Variable genX2 : (X2 → Prop) → Prop.
Definition Gen_Product : (X1 * X2 → Prop) → Prop :=
  fun A ⇒ ∃A1 A2, (genX1 A1 ∨ A1 = fun _ ⇒ True) ∧
  (genX2 A2 ∨ A2 = fun _ ⇒ True) ∧ (∀ B : X1*X2, A B ↔ A1 (fst B) ∧ A2 (snd B)).
```

here, the function `fst` extracts the first element of the pair `B` = (`B1`, `B2`), and `snd` extracts the second element.

3.1.3 Measurablility of Functions

Building on the concept of measurability for subsets, the measurability of a function is defined here. Considering two measurable spaces (X_1, Σ_1) and (X_2, Σ_2), along with their generators G_1 and G_2, a function $f : X_1 \to X_2$ is said measurable when for every set A_2 in Σ_2, the pre-image $f^{-1}(A_2)$ is in Σ_1. In other words, for a measurable subset A_2 in X_2, the set $\{x_1 \mid f(x_1) \in A_2\}$ in X_1 is measurable. The definition of a measurable function in Coq is then provided as follows:

```
Definition measurable_fun (f : X1 → X2) : Prop :=
  ∀ A2, measurable genX2 A2 → measurable genX1 (fun x1 : X1 ⇒ A2 (f x1)).
```

Let us focus on functions mapping from a measurable space X to $\overline{\mathbb{R}}$. We note that a σ-algebra on $\overline{\mathbb{R}}$, typically the Borel σ-algebra, consists of subsets of $\overline{\mathbb{R}}$ that include the empty set and are closed under the operations of taking complements and forming countable unions of these subsets. For $\overline{\mathbb{R}}$, a possible set of generators for this σ-algebra, among others, is the intervals such as (a, ∞) for each a in $\overline{\mathbb{R}}$. In Coq, this generator is defined as follows:

```
Definition gen_Rbar : (ℝ → Prop) → Prop := fun A ⇒ ∃ a : ℝ , ∀x : ℝ , A x ↔ a ⩽_ℝ x.
```

To classify a function $f : X \to \overline{\mathbb{R}}$ as measurable, it must satisfy the condition that the pre-image of any set from a σ-algebra on $\overline{\mathbb{R}}$ under f is a measurable set in X. The definition of measurability for this function is specified as follows:

```
Variable genX : (X → Prop) → Prop.
Definition measurable_fun_Rbar := measurable_fun genX gen_Rbar.
```

The set $\mathcal{M}(X, \Sigma)$ represents the set of measurable functions over a given measurable space X with values in $\overline{\mathbb{R}}$, and within this collection, the subset comprising solely nonnegative functions is indicated by $\mathcal{M}_+(X, \Sigma)$, and referred to in Coq as:

```
Definition Mplus : (X → R̄ ) → Prop := fun f ⇒ nonneg f ∧ measurable_fun_Rbar f.
```

One of the aspects of measurable functions is their compatibility with basic algebraic operations such as addition, scalar multiplication, and multiplication. Considering two nonnegative measurable functions $f, g : X \to \overline{\mathbb{R}}$ within a measurable space X, it is established that their summation $f+g$ and any scalar multiplication af (where $a \in \overline{\mathbb{R}}_+$) maintain measurability. These concepts are formalized within Coq as follows,

```
Lemma Mplus_plus : ∀(f1 f2 : X → R̄ ), Mplus f1 → Mplus f2 →
  Mplus (fun x ⇒ f1 x +R̄ f2 x).
Lemma Mplus_scal : ∀(a : R̄ ) (f : X → R̄ ), 0 ⩽R̄ a → Mplus f →
  Mplus (fun x ⇒ a *R̄ (f x)).
```

For additional details about $\overline{\mathbb{R}}$, refer to Section 2.2. Moreover, the pointwise multiplication of two nonnegative measurable functions f and g is also measurable:

```
Lemma Mplus_mult : ∀(f1 f2 : X → R̄ ), Mplus f1 → Mplus f2 →
  Mplus (fun x ⇒ f1 x *R̄ f2 x).
```

Furthermore, the supremum of a sequence of nonnegative measurable functions $(f_n)_{n\in\mathbb{N}}$ maintains both non-negativity and measurability:

```
Lemma Mplus_Sup_seq : ∀(f : nat → X → R̄ ), (∀ n, Mplus (f n)) →
    Mplus (fun x ⇒ Sup_seq (fun n ⇒ f n x)).
```

3.2 Measure Space

A measure space, denoted as (X, Σ, μ), is additionally equipped with a *measure* μ, a function from Σ to the extended real number line. By allowing the measure μ to take the value $+\infty$, it becomes possible to handle sets of infinite size.

Well-known examples of measures include the *Lebesgue measure*, which generalizes the concept of length to bounded intervals; the *counting measure*, assigning the number of elements to each set; and the *Dirac measure* at a point a in a space X, which assigns a measure of 1 to any set containing a and a measure of 0 to any set excluding a (see [11, Section 8 p.31]). These measures are foundational in fields such as probability theory and functional analysis, and are essential to understanding various probability measures detailed in [8].

Section 3.2.1 is dedicated to the formalization of measures within the Coq proof assistant. In this section, we detail essential axiomatic properties, such as σ-additivity, which are fundamental for a valid measure definition. Following this, Section 3.2.2 builds on this foundation to explore the main properties of measures, including monotonicity, and the continuity of measures from both below and above.

3.2.1 Formalization of Measures

In the context of a measurable space characterized by a set `X : Type` and a generator `genX: (X → Prop) → Prop` (detailed in Section 3.1.1), measures are formalized as a `Record` type in Coq. This record includes a mapping `meas : (X → Prop) → Rbar` that assigns to each subset a value in $\overline{\mathbb{R}}$, along with essential properties that qualify this mapping as a measure. The definition is structured as follows,

```
Record measure := mk_measure {
  meas :> (X → Prop) → Rbar;
```

```
meas_False : meas (fun _ ⇒ False) = 0;
meas_ge_0 : ∀A, 0 ⩽_ℝ (meas A);
meas_sigma_additivity : ∀A : nat → X → Prop,
  (∀ n, measurable genX (A n)) → (∀p n m x, A n x → A m x → n = m) →
  meas (fun x ⇒ ∃n, A n x) = Sup_seq (fun n ⇒ sum_Rbar n (fun m ⇒ meas (A m)))}.
```

To simplify the use of the `measure` record, coercion is employed (denoted by the symbol `:>`), transforming the type `measure` into a function type `(X→ Prop)→ ℝ̄` when needed. The principle `meas_sigma_additivity` of σ-additivity, states that for a sequence $(A_n)_{n\in\mathbb{N}}$ of pairwise disjoint measurable subsets of X, symbolized mathematically by $\uplus$ for disjoint unions, the measure is additive across the sequence,

$$\mu\left(\biguplus_{i\in\mathbb{N}} A_i\right) = \sum_{i\in\mathbb{N}} \mu(A_i) = \sup_{n\in\mathbb{N}} \sum_{i\in[0..n]} \mu(A_i) \tag{3.1}$$

3.2.2 Main Properties of Measures

From the assumptions defining measures, further properties such as monotonicity are derived,

$$A \subset B \Rightarrow \mu(A) \leqslant \mu(B)$$

for measurable subsets A and B. This allows the replacement of the limit of a nondecreasing sequence with its supremum. Other interesting concepts known as the *continuity from below* and *from above* of the measure are introduced, respectively (see [11, Section 5.2 p.18] for more details). In Section 4.2.4, we rely on these properties to establish the measurability of the measure of the section of subsets.

For any nondecreasing sequence of measurable subsets $A_1 \subseteq A_2 \subseteq A_3 \subseteq \cdots$ in Σ, continuity from below of μ states that the measure of the union of these subsets equals the limit (or the supremum) of the measures of these subsets. In mathematical terms,

$$\mu\left(\bigcup_{n\in\mathbb{N}} A_n\right) = \lim_{n\to\infty} \mu(A_n) = \sup_{n\in\mathbb{N}} \mu(A_n).$$

For any nonincreasing sequence of measurable subsets $A_1 \supseteq A_2 \supseteq A_3 \supseteq \cdots$ with at least one subset of finite measure $\mu(A_1) < \infty$, continuity from above of μ states that the measure of the intersection of these subsets equals the infimum of the measures of these subsets.

$$\mu\left(\bigcap_{n\in\mathbb{N}} A_n\right) = \lim_{n\to\infty} \mu(A_n) = \inf_{n\in\mathbb{N}} \mu(A_n).$$

The Coq formalization of continuity from below is straightforward as a property of the measure function.

```
Definition continuous_from_below : ((X → Prop) → ℝ̄ ) → Prop :=
  fun mu ⇒ ∀A : nat → X → Prop,
    (∀ n, measurable genX (A n)) → (∀ n x, A n x → A (S n) x) →
    mu (fun x ⇒ ∃n, A n x) = Sup_seq (fun n ⇒ mu (A n)).
```

here, `Sup_seq` calculates the supremum of a sequence of real numbers. We can prove that this continuity property holds for all measures. This is formalized as follows:

```
Lemma measure_continuous_from_below : ∀(mu : measure), continuous_from_below mu.
```

3.3 Simple Functions

This section is divided into three parts, each focusing on different aspects of simple functions and their application in measure theory. Section 3.3.1 is dedicated to the definition of simple functions. Moving forward, Section 3.3.2 addresses the canonical representation of simple functions in Coq, ensuring that the function values remain finite and are efficiently handled through a strictly sorted list. Lastly, Section 3.3.3 concentrates on the integration of nonnegative simple functions, highlighting the mathematical expression of the Lebesgue integral for these functions and its implementation in Coq.

3.3.1 Definition of Simple Functions

Simple functions are essential in constructing the Lebesgue integral. Such a function, mapping from a set X to the real numbers $\mathbb{R}$, can be expressed as a finite sum of distinct constants, each multiplied by a characteristic function, that indicates the measurable subsets of X where f attains the corresponding constant value. Specifically, they are functions whose image has finite cardinality.

Mathematically, a simple function f is defined as:

$$f = \sum_{y \in f(X)} y \times \mathbb{1}_{f^{-1}(\{y\})} \tag{3.2}$$

where $\mathbb{1}_A$ denotes the characteristic function (or indicator function) of subset A. The characteristic function is defined in Coq via pattern matching, assigns 1 to elements within the specified subset and 0 to all others,

```
Definition charac : (X → Prop) → X → R :=
  fun A x ⇒ match (excluded_middle_informative (A x)) with
    | left _ ⇒ 1
    | right _ ⇒ 0
    end.
```

Here `excluded_middle_informative` is the excluded middle axiom defined in Section 2.1. A characteristic function $\mathbb{1}_A$ is measurable if and only if its corresponding subset A belongs to the σ-algebra Σ, which is also convenient for representing the restriction of numerical functions to subsets of X (see for instance Section 4.2.3). The set of measurable characteristic functions is denoted as $\mathcal{IF}(X, \Sigma)$. Measurable simple functions are collected into the set $\mathcal{SF}(X, \Sigma)$, with a subset $\mathcal{SF}_+(X, \Sigma)$ including only the nonnegative simple functions. This subset, $\mathcal{SF}_+(X, \Sigma)$, is compatible with various mathematical operations, such as addition, multiplication, and nonnegative scalar multiplication.

3.3.2 Canonical Representation of Simple Functions

In [11], simple functions are *canonically* characterized by the strictly sorted list of their values, using the predicate `SFplus genX` : $(X \to \mathbb{R}) \to$ `Prop`. The traditional mathematical formulation of simple functions, as seen in Equation (3.2), is not directly implementable in Coq for cases where $f(X)$ might be infinite. Since f is a simple function, we guarantee that this range remains finite. To address this, a data structure in the form of a list is adopted, allowing access to the possible values, thereby enabling the computation of integrals for simple functions.

This Coq definition outlines a property called `finite_vals_canonic` that links the function f and the unique sorted list ℓ of real numbers, ensuring that each element in ℓ corresponds to at

least one element in X under the function f, and representing all possible useful values of the function f when applied to the elements of the set X.

```
Definition finite_vals_canonic : (X → R) → list R → Prop :=
  fun f l ⇒ (LocallySorted Rlt l) ∧ (∀ y, In y l → ∃x, f x = y) ∧ (∀ x, In (f x) l).
```

Here, `LocallySorted` : (ℝ→ ℝ→ `Prop`) → `list` ℝ→ `Prop` denotes the inductive definition of a sorted list, where the strict ordering `Rlt` is required to eliminate duplicates.

To establish that these simple functions indeed represent a finite linear combination of indicator functions, it has been demonstrated that `finite_vals_canonic` f ℓ leads to an equality similar to the Equation (3.2), but with y drawn from the list ℓ,

$$f = \sum_{y \in \ell} y \times \mathbb{1}_{f^{-1}(\{y\})}.$$

In Coq, this is formulated as,

```
Lemma finite_vals_sum_eq : ∀(f : X → ℝ) (l : list ℝ), finite_vals_canonic f l →
    ∀ x, f x = sum_Rbar_map l (fun y ⇒ y * (charac (fun z ⇒ f z = y) x)).
```

We now present the formalization in Coq of simple functions that have measurable preimages. The definitions are listed as follows:

```
Definition SF_aux := fun (genX : (X → Prop) → Prop) (f : X → ℝ) (l : list ℝ) ⇒
  finite_vals_canonic f l ∧ (∀ y, measurable genX (fun x ⇒ f x = y)).

Definition SF := fun (genX : (X → Prop) → Prop) (f : X → ℝ) ⇒ {l : list ℝ | SF_aux genX f l}.
```

In this context, the result is within the `Set` type because direct access to it is essential for integral computation. A weak existential declaration is not allowed for this purpose. In Coq, the expression {x | P x} signifies that there is a constructive way to prove the existence of an x such that P x holds, with the construct itself being of type `Set`. This constitutes a strong existential, meaning it is possible to extract and use the witness x in a statement (see Section 2.1).

Accordingly, the set $\mathcal{SF}_+$ is represented in Coq as,

```
Definition SFplus := fun (genX : (X → Prop) → Prop) (f : X → ℝ) ⇒
   nonneg (fun x : X ⇒ f x) ∧ (∃  l :  list ℝ, SF_aux genX f l).
```

3.3.3 Integration of Simple Functions

The Lebesgue integral of a simple function f, that belongs to the set $\mathcal{SF}$, is defined as the sum of the product between the function values and the measures of the sets on which these values are taken. This is mathematically represented as follows:

$$\int_{\mathcal{SF}} f \, d\mu \overset{\text{def.}}{=} \sum_{y \in f(X)} y \times \mu\left(f^{-1}(\{y\})\right). \tag{3.3}$$

Here, the subsets $f^{-1}(\{y\})$, corresponding to each y in the range $f(X)$, partition the domain X of the function.

The definition of the integral within $\mathcal{SF}$ is straightforward and derived from a proof of type `SF`.

```
Context {X : Type}.
Context {genX : (X → Prop) → Prop}.
Variable mu : measure genX.
Definition af1 : (X → ℝ) → ℝ̄ → ℝ̄ := fun (f : X → ℝ) (y : ℝ̄ ) ⇒
  y *_ℝ̄ (mu (fun x : X ⇒ Finite (f x) = y)).

Definition LInt_SF : ∀(f : X → ℝ), SF genX f → ℝ̄ :=
  fun f Hf ⇒ let lf := proj1_sig Hf in sum_Rbar_map lf (af1 f).
```

This definition requires the hypothesis `Hf : SF genX f`, which validates that f is indeed a simple function and provides the list witness `lf` that the definition relies upon. The function `proj1_sig` returns the first part of `Hf`, namely the list `lf`, which is then used as the basis for summation.

A remarkable property of the integral of simple functions is its linearity, which states that the integral operator is linear with respect to both the addition of functions and scalar multiplication in $\mathcal{SF}$ (refer to [11, Section 6.3 p.22]).

3.4 Integration of Nonnegative Measurable Functions

We now focus on nonnegative measurable functions of the form $\mathtt{X} \to \overline{\mathbb{R}}$, which may adopt an extensive range of values, including infinite ones. The Lebesgue integral for these nonnegative measurable functions is defined by approximating them from below using an increasing sequence of simple functions. The formulation of the Lebesgue integral, along with some of its initial properties, is introduced in Section 3.4.1. Following this, Section 3.4.2 focuses on the concept of adapted sequences. The crucial theorem of Beppo Levi (monotone convergence) is discussed in Section 3.4.3.

3.4.1 Definition and First Properties

The construction of the Lebesgue integral within the set $\mathcal{M}_+$ (see Section 3.1.3) operates in three steps. Initially, the integration of indicator functions in $\mathcal{IF}$ is achieved by taking the measure of their respective supports. Subsequently, this integration framework is extended to simple functions in $\mathcal{SF}_+$, using the principle of positive linearity (refer to Section 3.3.3 for further details). Finally, this process involves a further extension to all measurable functions in $\mathcal{M}_+$, by taking the supremum of the integrals of these simple functions.

The integral for a nonnegative measurable function f within $\mathcal{M}_+$ is represented as follows:

$$\forall f \in \mathcal{M}_+, \quad \int_{\mathcal{M}_+} f \, d\mu \overset{\text{def.}}{=} \sup_{\substack{\psi \in \mathcal{SF}_+ \\ \psi \leqslant f}} \int_{\mathcal{SF}_+} \psi \, d\mu \tag{3.4}$$

In this expression, the supremum is computed over all nonnegative measurable simple functions ψ that are pointwise less than or equal to f. The integral within $\mathcal{SF}_+$ is detailed in Section 3.3.3.

Defining this integral as a total function, we assign a value to any input function f. This Coq definition is similar to the mathematical expression (3.4).

```
Definition LInt_p (f : X → ℝ̄ ) : ℝ̄ :=
  Rbar_lub (fun y : ℝ̄ ⇒ ∃(g : X → ℝ) (Hg : SF genX g),
    non_neg g ∧ (∀ x, g x ⩽_ℝ̄ f x) ∧ LInt_SF mu g Hg = y).
```

In this definition, `Rbar_lub` has the type $(\overline{\mathbb{R}} \to \texttt{Prop}) \to \overline{\mathbb{R}}$ (see Section 2.2), and it computes the least upper bound (`LUB`) or supremum.

An elementary example of integration is the relationship between this integral and the characteristic function since the latter is a measurable function. When the subset A is measurable, $\mathbb{1}_A$ belongs to $\mathcal{M}_+$, and its integral with respect to the measure μ corresponds to the measure of A. This is written as:

$$\int_{\mathcal{M}_+} \mathbb{1}_A \, d\mu = \mu(A). \tag{3.5}$$

```
Lemma LInt_p_charac :
  ∀ A, measurable genX A → LInt_p (charac A) = mu A.
```

A fundamental aspect of the Lebesgue integral is its monotonicity property, which can be stated as follows:

$$\forall f, g \in \mathcal{M}_+, \quad f \leqslant_{\overline{\mathbb{R}}} g \Longrightarrow \int_{\mathcal{M}_+} f \, d\mu \leqslant_{\overline{\mathbb{R}}} \int_{\mathcal{M}_+} g \, d\mu.$$

This is translated into Coq as:

```
Lemma LInt_p_monotone :
  ∀ (f g : X → \bar R), (∀ x, f x ⩽ g x) → LInt_p f ⩽ LInt_p g.
```

The concept of the least upper bound used in defining the total function suffices to ensure monotonicity for any functions f and g, not only for the nonnegative and measurable functions as indicated in the mathematical statement.

Another result involves the σ-additivity of the integral stating that for any sequence of nonnegative measurable functions $(f_n)_{n\in[1..N]}$, the integral of the sum of these functions is equal to the sum of their individual integrals within $\mathcal{M}_+$. Mathematically, this can be expressed as follows:

$$\int_{\mathcal{M}_+} \left(\sup_{N\in\mathbb{N}} \sum_{n=1}^{N} f_n \right) d\mu = \sup_{N\in\mathbb{N}} \sum_{n=1}^{N} \int_{\mathcal{M}_+} f_n \, d\mu \tag{3.6}$$

3.4.2 Adapted Sequences

An *adapted sequence* for a given function f, is a nondecreasing sequence of nonnegative functions $(\varphi_n)_{n\in\mathbb{N}}$, that converges pointwise from below toward f: as n tends to infinity, the pointwise limit of the sequence becomes f, and equivalently, the supremum (or least upper bound) of the sequence at each point equates to f. Mathematically, this means:

$$f = \lim_{n\to\infty} \varphi_n = \sup_{n\in\mathbb{N}} \varphi_n.$$

In Coq, this concept is expressed as:

```
Definition is_adapted_seq : (X → \bar R) → (nat → X → ℝ) → Prop :=
  fun f g ⇒ (∀ n, non_neg (g n)) ∧ (∀ x n, g n x ⩽ g (S n) x) ∧
    (∀ x, is_sup_seq (fun n ⇒ g n x) (f x)).
```

Since the Lebesgue integral extends the notion of integration to a wider class of functions beyond simple functions, adapted sequences were constructed to provide a means to *approximate* any nonnegative measurable function by a sequence of simple functions. In [11], the value of adapted sequences corresponding to all nonnegative measurable functions was given.

3.4.3 The Theorem of Beppo Levi (Monotone Convergence)

The theorem of Beppo Levi, also referred to as the monotone convergence theorem, stands as one of the most fundamental results within the domains of measure and integration theories. Given a measure space defined by a set X, a σ-algebra Σ, and a measure μ, the theorem of Beppo Levi introduces an integral-limit interchange formula, mathematically represented as,

Theorem 1 (Beppo Levi, monotone convergence)**.**
Let $(f_n)_{n\in\mathbb{N}}$ be a sequence of pointwise nondecreasing and nonnegative measurable functions. Then, the $\sup_{n\in\mathbb{N}} f_n$ is nonnegative and measurable, and we have in $\mathcal{M}_+$

$$\int_{\mathcal{M}_+} \sup_{n\in\mathbb{N}} f_n \, d\mu = \sup_{n\in\mathbb{N}} \int_{\mathcal{M}_+} f_n \, d\mu \tag{3.7}$$

The proof of this lemma involves a comprehensive set of established results of measurable functions, including monotony of the integral, and fundamental properties of the supremum as referenced in [11]. This theorem is formalized in Coq as follows:

```
Lemma Beppo_Levi : ∀(f : nat → X → Rbar),
  Mplus_seq genX f → (∀ x n, f n x ⩽_ℝ f (S n) x) →
   LInt_p mu (fun x ⇒ Sup_seq (fun n ⇒ f n x)) = Sup_seq (fun n ⇒ LInt_p mu (f n)).
```

Chapter 4

Formalization of the Tonelli Theorem

This chapter highlights one of my contributions during my PhD, detailing the formalization of the Lebesgue induction principle and the Tonelli Theorem. This work has been published as a conference article in 2022 [9]. Section 4.1 is dedicated to the Lebesgue induction principle, a proof technique for dealing with properties related to nonnegative measurable functions. Section 4.2 is focused on the construction of product measures within a product space, including the development of the corresponding product σ-algebra and the properties of the section of subsets. Furthermore, Section 4.3 is dedicated to the construction of iterated integrals and the formalization of the proof of the Tonelli theorem. These developments are supported by the insights presented in Chapter 3, which outline the properties of Lebesgue integration.

4.1 Lebesgue Induction Principle

As explained in Section 3.1.3, within the context of a measurable space (X, Σ), the function space $\mathcal{M}_+(X, \Sigma)$ demonstrates closure under nonnegative scalar multiplication, addition, and the supremum operation. The latter implies that if we consider a collection, potentially infinite, of functions from $\mathcal{M}_+$ and take the supremum (least upper bound) of that collection pointwise, the resulting function remains within the $\mathcal{M}_+$ space.

4.1.1 Inductive Representation of Nonnegative Measurable Functions

We recall the function spaces as defined in Section 3.1.3 as follows:

$$\mathcal{M}_+(X, \Sigma) = \{f : X \to \overline{\mathbb{R}}_+ \mid \forall A \subset \overline{\mathbb{R}} \text{ measurable }, f^{-1}(A) \in \Sigma\} \tag{4.1}$$

$$\mathcal{SF}_+(X, \Sigma) = \{f : X \to \mathbb{R}_+ \ \mid \ f(x) = \sum_{y \in f(X)} y \times \mathbb{1}_{f^{-1}(\{y\})}\}, \tag{4.2}$$

$$\mathcal{IF}(X, \Sigma) = \{\mathbb{1}_A : X \to \{0, 1\} \mid A \in \Sigma\}. \tag{4.3}$$

The space $\mathcal{M}_+$ of nonnegative measurable functions was initially defined in Coq and discussed in Section 3.1.3, page 23, following the standard mathematical definition denoted as `Mplus`. According to this definition, a function is measurable if the pre-image of any measurable set is also measurable. In this section, however, we introduce an alternative representation of $\mathcal{M}_+$ as an inductive type, `Mp`, built on extended real-valued functions. Specifically, we represent the set $\mathcal{M}_+$ as the closure of measurable characteristic functions under positive linear combinations and increasing supremum operations.

```
Inductive Mp : (X → R̄) → Prop :=
  | Mp_charac : ∀A, measurable genX A → Mp (charac A)
  | Mp_scal : ∀ (a : ℝ) (f : X → R̄), 0 ⩽ a → Mp f → Mp (fun x ⇒ a *_R̄ f x)
  | Mp_plus : ∀ f g : X → R̄, Mp f → Mp g → Mp (fun x ⇒ f x +_R̄ g x)
  | Mp_sup : ∀ f : nat → X → R̄, incr_fun_seq f → (∀ n, Mp (f n)) →
             Mp (fun x ⇒ Sup_seq (fun n ⇒ f n x)).
```

Here, `charac A` denotes the characteristic function $\mathbb{1}_A$ of a subset A (see Section 3.3.1), `genX` refers to the generator of the measurable space `X` (see Section 3.1.1), and the property `incr_fun_seq f` is defined by the following relation `∀ x n, f n x ⩽_R̄ f (S n) x`.

Consequently, the Coq system provides for free the following induction lemma corresponding to `Mp`:

```
Lemma Mp_ind : ∀P : (X → R̄) → Prop,
  (∀ A, measurable genX A → P (charac A)) →
  (∀ (a : ℝ) f, 0 ⩽ a → Mp f → P f → P (fun x ⇒ a *_R̄ f x)) →
  (∀ f g, Mp f → P f → Mp g → P g → P (fun x ⇒ f x +_R̄ g x)) →
  (∀ f, incr_fun_seq f → (∀ n, Mp (f n)) →
  (∀ n, P (f n)) → P (fun x ⇒ Sup_seq (fun n ⇒ f n x))) →
   ∀ f, (Mp f → P f).
```

This representation is a widely-used proof technique in Lebesgue integration theory, that has various applications, including proving the Tonelli theorem as highlighted in the article [9], which will be explained later in Section 4.3. This approach can be informally stated as,

Lemma 1 (Lebesgue induction principle). *Let P be a predicate on functions of type $X \to \overline{\mathbb{R}}$. Assume that P holds on $\mathcal{IF}$, and that it is compatible on $\mathcal{M}_+$ with positive linear operations and with the supremum of nondecreasing sequences:*

$$\forall A, \quad A \in \Sigma \Rightarrow P(\mathbb{1}_A), \tag{4.4}$$

$$\forall a \in \mathbb{R}_+, \ \forall f \in \mathcal{M}_+, \quad P(f) \Rightarrow P(af), \tag{4.5}$$

$$\forall f, g \in \mathcal{M}_+, \quad P(f) \wedge P(g) \Rightarrow P(f+g), \tag{4.6}$$

$$\forall (f_n)_{n\in\mathbb{N}} \in \mathcal{M}_+, \quad (\forall n \in \mathbb{N}, \ f_n \leqslant f_{n+1} \wedge P(f_n)) \Rightarrow P\left(\sup_{n\in\mathbb{N}} f_n\right). \tag{4.7}$$

Then, P holds on $\mathcal{M}_+$.

There exist several alternative formulations of the Lebesgue induction principle. For example, we choose to have a in $\mathbb{R}$ rather than $\overline{\mathbb{R}}$ in the Equation (4.5), resulting in an equivalent yet more straightforward lemma to use. Moreover, as indicated in ([71, p.8]), it is possible to refine the premises of the constructors. For instance, in the Equation (4.6), it might suffice to consider simple functions f and g with distinct image values, except 0, or those with disjoint supports.

Lemma 1 (Lebesgue induction principle) holds on `Mp`. However, we want to establish its validity within `Mplus`. To achieve this, we will demonstrate the equivalence between `Mp` and `Mplus`.

4.1.2 Verifying `SFp` and `SFplus` Equivalence

We further introduce an inductive type for the set of nonnegative simple functions denoted as `SFp`, with constructors that are similar to the initial three constructors of `Mp`.

```
Inductive SFp : (X → ℝ) → Prop :=
  | SFp_charac : ∀A : X → Prop, measurable genX A → SFp (charac A)
  | SFp_scal : ∀(a : ℝ) (f : X → ℝ), 0 ⩽ a → SFp f → SFp (fun x ⇒ a * f x)
  | SFp_plus : ∀(f g : X → ℝ), SFp f → SFp g → SFp (fun x ⇒ f x + g x).
```

In this section, our primary goal is to establish the correctness of the `SFp` definition in comparison to the one outlined in Section 3.3.2 as `SFplus`. This involves proving a lemma that establishes the equivalence of the inductive type `SFp` with `SFplus`, as stated below,

```
Lemma SFp_correct : ∀f, SFp f ↔ SFplus genX f.
```

We recall that `SFplus` is defined as the set of simple functions canonically represented through a strictly sorted list of their values, as elaborated in [11]. The definition of `SFplus` includes the generator `genX` as an argument, which is essential to ensure that the function `f` meets the measurability requirements of the measurable space `X`. `genX` also serves as an argument of `SFp`, not implicit here.

The proof of this lemma is divided into two implications separately. The first part, proving `SFp f` implies `SFplus genX f`, is straightforward and uses the induction property on the hypothesis for validation. The reverse implication, from `SFplus genX f` to `SFp f`, is more complex as it relies on auxiliary results, which will be detailed later in the proof. We proceed by addressing different cases based on the structure of the list ℓ.

(i) Empty List $\ell = []$: The function f belongs to `SFp` since f is expressed as a linear combination of characteristic functions over an empty set, which are trivially measurable.

(ii) Base Case $(v_1 :: [])$: We consider f with only one value, v_1, represented as,

$$f = v_1 \times \mathbb{1}_{f^{-1}(\{v_1\})}.$$

which meets the `SFp` conditions through scaling and characteristic function properties.

(iii) Inductive Step $(v_1 :: v_2 :: \ell)$: In this case, we rely on an auxiliary lemma that will be proved after we complete the proof of this implication.

```
Lemma SFplus_cons :
  ∀ (f : X → ℝ) v1 v2 l, nonneg f → SF_aux genX f (v1 :: v2 :: l) →
    let g x := f x + (v1 − v2) * charac (fun t ⇒ f t = v2) x in
    nonneg g ∧ SF_aux genX g (v1 :: l).
```

This lemma constructs a simple function g, associated with a smaller sublist of size $n+1$, from a simple function f represented by a list of values of size $n+2$. The function g is defined as the sum of f and the difference between the first two values in the list, scaled by the characteristic function of the pre-image (or inverse image) of v_2 under the function f. This allows expressing f as:

$$f = (v_2 - v_1) \times \mathbb{1}_{f^{-1}(\{v_2\})} + g,$$

such that g maintains the `SFplus` conditions. Applying the `SFplus` rule confirms that f, as a sum of two simple functions, remains simple. The scalar multiple $(v_2 - v_1) \times \mathbb{1}_{f^{-1}(\{v_2\})}$ is then validated as a simple function using the `SFp_scal` rule, contingent on $v_2 - v_1 \geq 0$, which is established due to the ordering described in section 3.3.2. Finally, the `SFp_charac` rule substantiates the simplicity of the characteristic function. By applying the induction hypothesis to $g(x)$, we verify its simplicity by using the recursive reduction in list ℓ, thereby deducing that f indeed satisfies the `SFp` criteria. □

The construction of the function g is challenging for two reasons. Firstly, assigning the value zero to g over the set $f^{-1}(\{v_2\})$—that is, for every x in the domain of f where $f(x) = v_2$, setting $g(x) = 0$)— is unsuitable because zero might introduce a new value to the list, which goes against the objective of reducing the size of the list of values of g. Consequently, the initial list must contain at least two distinct values to avoid this issue. Secondly, the alternative approach of setting g to v_2 over $f^{-1}(\{v_1\})$ does not allow us to represent f as the sum of g and a *nonnegative* value multiplied by an indicator function.

Now, let us proceed to prove the lemma `SFplus_cons`. Given $f \in \mathcal{SF}_+$ and its associated canonical list ℓ, this lemma constructs a new function g in $\mathcal{SF}_+$, canonically associated with the list ℓ minus a specific item v_2. This implies that on the nonempty subset $f^{-1}(\{v_2\})$, g is constrained to take one of the remaining values, specifically v_1, as illustrated in Figure 4.1. Furthermore, this construction guarantees the property $g \leqslant f$ due to the property of the canonical list (see Section 3.3.2).

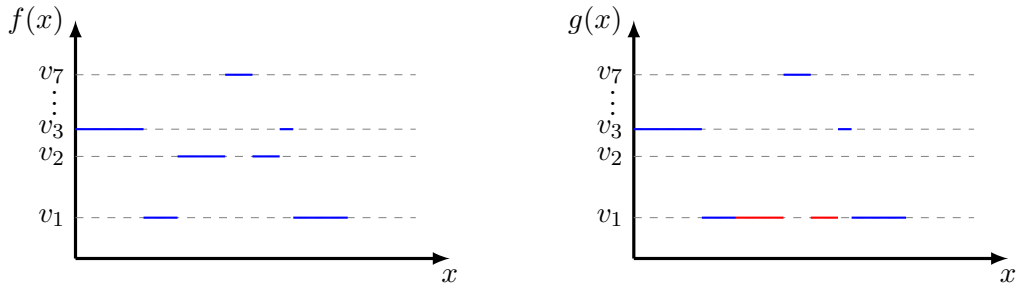

Figure 4.1: Illustration of Lemma `SFplus_cons`. The value v_2 taken by the simple function f (on the left) is replaced in g (on the right) by the value v_1 (in red).

Since f belongs to `SFplus` then we express f as,

$$f = \sum_{v \in \{v_1, v_2\} \cup \ell} v \times \mathbb{1}_{f^{-1}(\{v\})},$$

where v_1 and v_2 are distinct values such that $v_1 < v_2$, and neither v_1 nor v_2 is included in the list ℓ.

By defining g as $g := f + (v_1 - v_2) \times \mathbb{1}_{f^{-1}(\{v_2\})}$, the result is,

$$g = \sum_{v \in \{v_1\} \cup \ell} v \times \mathbb{1}_{f^{-1}(\{v\})} + v_1 \times \mathbb{1}_{f^{-1}(\{v_2\})},$$

showing that g belongs to `SFplus` with a reduced list of values.

4.1.3 Equivalent Inductive Types of `Mp`

To decompose the proof process of the equivalence between `Mp` and `Mplus` in Coq, we introduce and enumerate several inductive types that serve as equivalents to the inductive type `Mp`. More precisely, the first inductive type, `Mp1`, is constructed using `SFp`. Meanwhile, the second inductive type, `Mp2`, is proven to be equivalent to `Mp1`. The passage from `Mp2` to `Mp` goes by taking the supremum, which allows to recover functions that yield values in $\overline{\mathbb{R}}$, using the coercions between $\mathbb{R}$ and $\overline{\mathbb{R}}$ (see Section 2.2). The Coq code implementations corresponding to these inductive types are provided below,

```
Inductive Mp1 : (X → ℝ̄) → Prop :=
  | Mp1_SFp : ∀ f : X → ℝ, SFp genX f → Mp1 f
  | Mp1_sup : ∀ (f : nat → X → ℝ), incr_fun_seq f →
                (∀ n, Mp1 (f n)) → Mp1 (fun x ⇒ Sup_seq (fun n ⇒ f n x)).

Inductive Mp2 : (X → ℝ̄) → Prop :=
  | Mp2_charac : ∀A, measurable genX A → Mp2 (charac A)
  | Mp2_scal : ∀(a : ℝ) (f : X → ℝ), 0 ⩽ a → Mp2 f → Mp2 (fun x ⇒ a * f x)
  | Mp2_plus : ∀(f g : X → ℝ), Mp2 f → Mp2 g → Mp2 (fun x ⇒ f x + g x)
  | Mp2_sup : ∀ (f : nat → X → ℝ), incr_fun_seq f →
                (∀ n, Mp2 (f n)) → Mp2 (fun x ⇒ Sup_seq (fun n ⇒ f n x)).
```

We notice that `Mp2` shares similarities with the inductive type `Mp`, yet differs in their operational definitions for addition and multiplication: `Mp` uses $+_{\overline{\mathbb{R}}}$ and $*_{\overline{\mathbb{R}}}$, whereas `Mp2` adopts the real $+_{\mathbb{R}}$ and $*_{\mathbb{R}}$ operators. The path for proving the equivalence between `Mp` and these two inductive types is straightforward, using induction on the hypothesis of `f` and the construction of the adapted sequences (see Section 3.4.2).

4.1.4 Verifying `Mp` and `Mplus` Equivalence

With all essential elements established, we are now equipped to verify the correctness of the `Mp` definition. In other words, we aim to ensure that `Mp` accurately represents $\mathcal{M}_+$ just as `Mplus` does. This is captured in the following lemma.

```
Lemma Mp_correct : ∀f, Mp f ↔ Mplus genX f.
```

To establish the first implication of this equivalence, `Mp f → Mplus genX f`, the proof uses the inductive principle on the premise `Mp f`. This requires proving that `f`, as defined by each of the `Mp` constructors (refer to Section 4.1.1), satisfies `Mplus`. Conversely, the proof for `Mplus genX f → Mp f` is more direct, relying solely on the implications `Mp2 genX f → Mp f` and `Mplus genX f → Mp2 f`, as outlined in Section 4.1.3. Thus, the Lebesgue induction principle applies to `Mplus`.

4.2 Product Measure on a Product Space

In this section, we define the product measure for measurable subsets within a product space, allowing for integration over such a product space. The concepts and tools developed here are used for the subsequent proof of the Tonelli theorem in Section 4.3.

4.2.1 Specification of a Product Measure

Before delving into the specifics of the product measure, let us establish the concept of a product measure space. Consider two measure spaces (X_1, Σ_1, μ_1) and (X_2, Σ_2, μ_2), we define the *product measure space* as $(X_1 \times X_2, \Sigma_1 \otimes \Sigma_2, \mu_1 \otimes \mu_2)$, where $\Sigma_1 \otimes \Sigma_2$ represents the *product σ-algebra*, which will be further discussed in Section 4.2.2, and $\mu_1 \otimes \mu_2$ is the product measure that operates over this product σ-algebra and will be defined in Section 4.2.5.

The measure $\mu_1 \otimes \mu_2$ has to satisfy the properties of measures outlined in Section 3.2.1, and conforms to the *box property*. This property is expressed as follows,

$$\forall A_1 \in \Sigma_1,\ \forall A_2 \in \Sigma_2, \quad \mu_1 \otimes \mu_2(A_1 \times A_2) = \mu_1(A_1)\,\mu_2(A_2). \tag{4.8}$$

To guarantee the *existence* and *uniqueness* of such a product measure (see Section 4.2.5), we require that μ_1 and μ_2 be *σ-finite*. In other words, the full sets X_1 and X_2 are the nondecreasing

unions of subsets with finite measures. Informally, this can be written as: $X_1 = \bigcup_{n=1}^{\infty} A_n$ and $X_2 = \bigcup_{n=1}^{\infty} B_n$, where each A_n and B_n has a finite measure, i.e., $\mu_1(A_n) < \infty$ and $\mu_2(B_n) < \infty$.

The construction of a candidate product measure involves a three-step process, as illustrated in Figure 4.2. Initially, we establish X_1-*sections* (also known as *"vertical" cuttings*) of subsets to be Σ_2-measurable (refer to Section 4.2.3). Subsequently, we prove the measurability of the measure of these sections over (X_1, Σ_1), constructed in two stages, depending on whether μ is finite or σ-finite as will be detailed in Section 4.2.4. Finally, the candidate product measure is defined as the integral of these section measures, as will be outlined in Section 4.2.5. Ultimately, we verify that this candidate qualifies as a product measure, and we ensure its uniqueness is guaranteed. The main argument behind establishing the measurability of the measure of sections lies in the monotone class theorem [22, Section 1.6 p.40], which is applied twice in this context: first, to establish the measurability of the measure of sections, and second, to ensure the uniqueness of the resulting product measure.

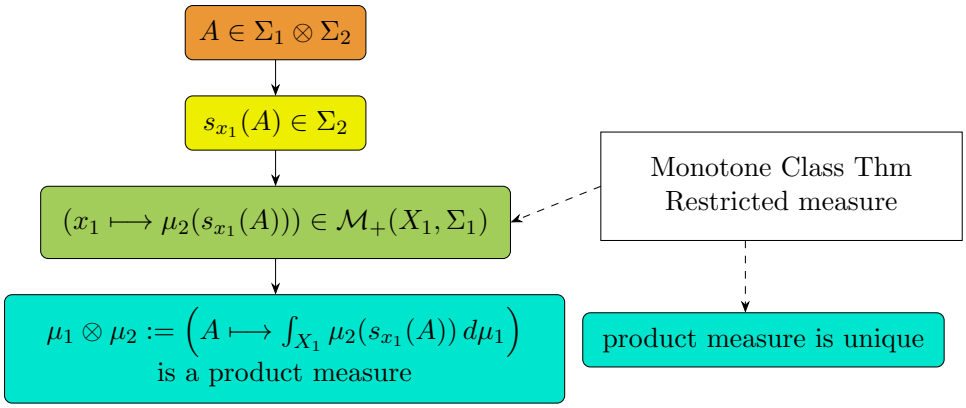

Figure 4.2: Flowchart illustrating the construction of the product measure. The filled colors represent different subsections: 4.2.2 in brown, 4.2.3 in yellow, 4.2.4 in green, and 4.2.5 in turquoise blue. Dashed lines indicate the application of the proof arguments.

4.2.2 Product σ-Algebra

In conjunction with the notion of product measure, the product σ-algebra, denoted as $\Sigma_1 \otimes \Sigma_2$ over the space $X_1 \times X_2$ is introduced in [11, Section 4.3]. This σ-algebra is the smallest σ-algebra generated by the Cartesian products of measurable subsets A_1 and A_2 that belong to the respective σ-algebras Σ_1 and Σ_2. Its informal mathematical definition is as follows,

$$\Sigma_1 \otimes \Sigma_2 \overset{\text{def.}}{=} \sigma\left(\Sigma_1 \overline{\times} \Sigma_2\right) \text{ where } \Sigma_1 \overline{\times} \Sigma_2 \overset{\text{def.}}{=} \{A_1 \times A_2 \mid A_1 \in \Sigma_1, A_2 \in \Sigma_2\}. \tag{4.9}$$

Here, $\sigma(\cdot)$ denotes the σ-algebra generated by a collection of sets $\Sigma_1 \overline{\times} \Sigma_2$ (refer to Section 3.1.1).

According to [11, Section 4.3] and Section 3.1.2, $\Sigma_1 \otimes \Sigma_2$ can be equivalently described as the σ-algebra generated by the expression,

$$(\text{gen}(\Sigma_1) \cup \{X_1\}) \overline{\times} (\text{gen}(\Sigma_2) \cup \{X_2\})$$

This generator is implemented in Coq as `Gen_Product genX1 genX2`, and is succinctly referred to as `genX1xX2` in subsequent sections. Similarly, reversing the variable order yields `genX2xX1`, which corresponds to `Gen_Product genX2 genX1`.

4.2.3 Section of Subsets

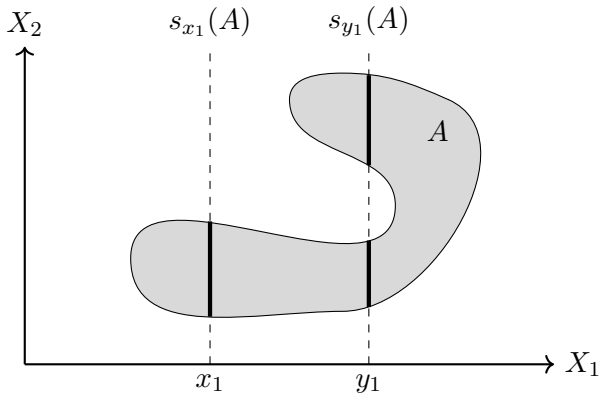

Figure 4.3: X_1-sections of a subset A of $X_1 \times X_2$ at points x_1 and y_1.

In the context of integrating over a product space, the concept of *sections of subsets* becomes relevant. This mathematical concept involves maintaining one variable constant while investigating a subset within the Cartesian product space. To illustrate, let us consider a Cartesian product space $X_1 \times X_2$ with a subset A contained within it, as shown in Figure 4.3.

Taking a specific point x_1 in X_1, the X_1-section of A at x_1 is the subset of X_2 formed by pairing each element x_2 in X_2 with x_1 in such a way that (x_1, x_2) belongs to A. This operation effectively *slices* the original subset A along the X_1-axis, resulting in a subset of X_2 that corresponds to all possible second coordinates for the fixed x_1. The X_1-section is defined mathematically and in the Coq language as follows:

$$s_{x_1}(A) := \{x_2 \in X_2 \mid (x_1, x_2) \in A\}.$$

```
Definition section (x1 : X1) (A : X1 * X2 → Prop) (x2 : X2) : Prop := A (x1, x2).
```

Similarly, one can define the X_2-section of A at a specific point x_2 in X_2 by fixing the second variable and considering all points in X_1 that, when paired with x_2, belong to A.

Sections exhibit compatibility with various set operations. For instance, they are compatible with the empty set ($s_{x_1}(\varnothing) = \varnothing$), the complement ($s_{x_1}(A^c) = s_{x_1}(A)^c$), countable union ($s_{x_1}\left(\bigcup_{i=1}^{\infty} A_i\right) = \bigcup_{i=1}^{\infty} s_{x_1}(A_i)$), countable intersection ($s_{x_1}\left(\bigcap_{i=1}^{\infty} A_i\right) = \bigcap_{i=1}^{\infty} s_{x_1}(A_i)$), and maintain monotonicity ($A \subset B \Rightarrow s_{x_1}(A) \subset s_{x_1}(B)$). Additionally, they satisfy the following property of sections,

$\forall A_1 \in X_1$, $\forall A_2 \in X_2$, and $\forall x_1 \in X_1$,

$$s_{x_1}(A_1 \times A_2) = \begin{cases} A_2 & \text{when } x_1 \in A_1, \\ \varnothing & \text{otherwise}, \end{cases} \tag{4.10}$$

Following this, the subsequent lemma establishes that if a subset A is $\Sigma_1 \otimes \Sigma_2$-measurable, then its X_1-sections at any point in X_1 are Σ_2-measurable. Given that the measurability of subsets is defined as an inductive type (see Section 3.1.1), the proof involves straightforward induction on the hypothesis.

```
Lemma section_measurable :
  ∀ A x1, measurable genX1xX2 A → measurable genX2 (section x1 A).
```

4.2.4 Measurability of Measure of Section

Given the measurability of sections (see Section 4.2.3), their measures can be determined. In Figure 4.2, the product measure is defined as the integral of the measures of sections. In Coq, the representation of the measure for X_1-sections is defined through the total function:

```
Definition meas_section (A : X1 * X2 → Prop) (x1 : X1) : ℝ := muX2 (section x1 A).
```

Here, `muX2` denotes the measure μ_2 on the space X_2.

Before proceeding with the integration of this function, it is essential to establish both its nonnegativity and measurability. Specifically, for every $\Sigma_1 \otimes \Sigma_2$-measurable subset A, let us demonstrate that the function $(x_1 \longmapsto \mu_2(s_{x_1}(A)))$ belongs to $\mathcal{M}_+(X_1, \Sigma_1)$.

The nonnegativity property of $(x_1 \longmapsto \mu_2(s_{x_1}(A)))$ is a direct consequence of the properties of measures. The proof of measurability unfolds in two stages: firstly, when the measure μ_2 is assumed to be *finite* (i.e., when $\mu_2(X_2)$ is finite); and secondly, in the more general σ-finite case (i.e., when the full set X_2 is the nondecreasing unions of subsets with finite measure μ_2). The first stage adopts a high-level approach, relying on the monotone class's theorem, and will be applied further to prove the second stage. The latter stage extends the first by incorporating restricted measures.

Let us now prove that the measure of X_1-sections is measurable.

(i) Assuming `muX2` is finite. The first step of the proof of the measurability of `meas_section` is stated in Coq as,

```
Lemma meas_section_Mplus_finite : ∀A, is_finite_measure muX2 →
  measurable genX1xX2 A → Mplus genX1 (meas_section A).
```

In this context, the assumption `is_finite_measure muX2` asserts the finiteness of the measure `muX2`. The lemma `meas_section_Mplus_finite` states that under this assumption and considering the measurability of the subset A with respect to the product space generated by `genX1xX2`, the measure of the section of this subset belongs to the space $\mathcal{M}_+(X_1, \Sigma_1)$.

Define $\mathcal{S}$ as the set of measurable subsets A of the product space $\Sigma_1 \otimes \Sigma_2$ for which the function mapping each x_1 to the measure of the sections $s_{x_1}(A)$ in the space X_2 under the measure μ_2 belongs to the space of nonnegative measurable functions,

$$\mathcal{S} := \left\{ A \in \Sigma_1 \otimes \Sigma_2 \,|\, \left(x_1 \longmapsto \mu_2(s_{x_1}(A))\right) \in \mathcal{M}_+(X_1, \Sigma_1) \right\}.$$

To establish the claim that $\mathcal{S} = \Sigma_1 \otimes \Sigma_2$, it is sufficient to show that $\Sigma_1 \otimes \Sigma_2 \subset \mathcal{S}$. Initially, we prove that $\mathcal{S}$ contains a generator $\overline{\Sigma} := \Sigma_1 \overline{\times} \Sigma_2$ of $\Sigma_1 \otimes \Sigma_2$ (refer to Section 4.2.2). Next, we demonstrate that $\mathcal{S}$ includes the algebra of sets generated by $\overline{\Sigma}$ (i.e., the closure of $\overline{\Sigma}$ under complement and finite union). Furthermore, we establish that $\mathcal{S}$ forms a monotone class, demonstrating closure under monotone countable union and intersection. This step relies on the finiteness assumption on μ_2, along with continuity from below and

from above (see Section 3.2.2).

Finally, we conclude by applying the following corollary of the monotone class theorem using `X` := `X1` * `X2`, `P` := $\mathcal{S}$, and `gen` := $\overline{\Sigma}$. This corollary asserts that if a monotone class `P` (i.e., `P` is closed under countable increasing unions and countable decreasing intersections of its subsets) contains the smallest algebra of sets containing `gen`, it also includes the smallest σ-algebra containing `gen` (see [22, Section 1.6 p.40].).

```
Theorem monotone_class_Prop : ∀(gen : (X → Prop) → Prop) (P : (X → Prop) → Prop),
  is_Monotone_class P → Incl (Algebra gen) P →
  Incl (Sigma_algebra gen) P.
```

In this context, the function `Incl` represents the inclusion relationship between subsets within the power set of X. Furthermore, `Algebra` denotes an algebra of sets generated by `gen`, characterized by closure under taking complements and finite unions. Meanwhile, `Sigma_algebra` extends the concept of an algebra by being closed under countable unions.

(ii) In the second stage of the proof of the measurability of `meas_section`, we presume the measure μ_2 to be σ-finite. The lemma is presented in Coq as follows,

```
Lemma meas_section_Mplus_sigma_finite :
 ∀ A, is_sigma_finite_measure muX2 →
   measurable genX1xX2 A → Mplus genX1 (meas_section A).
```

The assumption of σ-finitness implies the existence of a nondecreasing sequence $(B_n)_{n\in\mathbb{N}} \in \Sigma_2$ such that $X_2 = \bigcup_{n\in\mathbb{N}} B_n$, and $\mu_2(B_n)$ is finite for every $n \in \mathbb{N}$.

For each $n \in \mathbb{N}$, we establish the *restricted measure* as follows,

$$\mu_2^n := (A_2 \in \Sigma_2 \longmapsto \mu_2(A_2 \cap B_n) \in \overline{\mathbb{R}}_+)$$

Consequently, since μ_2^n is a finite measure on (X_2, Σ_2), and considering the initial stage of the proof (i) when dealing with finite measures, we deduce that,

$$\forall A \in \Sigma_1 \otimes \Sigma_2, \quad (x_1 \longmapsto \mu_2^n(s_{x_1}(A))) \in \mathcal{M}_+(X_1, \Sigma_1).$$

Moreover, considering that $s_{x_1}(A) \in \Sigma_2$ (refer to Section 4.2.3), and using the distributivity of countable intersection over union, we have,

$$s_{x_1}(A) = s_{x_1}(A) \cap \bigcup_{n\in\mathbb{N}} B_n = \bigcup_{n\in\mathbb{N}} (s_{x_1}(A) \cap B_n)\,.$$

By applying the continuity from below property of μ_2, we observe that for all $A \in \Sigma_1 \otimes \Sigma_2$ and $x_1 \in X_1$,

$$\mu_2(s_{x_1}(A)) = \mu_2\left(\bigcup_{n\in\mathbb{N}} (s_{x_1}(A) \cap B_n)\right) = \sup_{n\in\mathbb{N}} \mu_2^n(s_{x_1}(A)).$$

In conclusion, the proof is finished since $\mathcal{M}_+(X_1, \Sigma_1)$ is closed under supremum, as detailed in Section 3.1.3. Thus, $\mu_2(s_{x_1}(A)) \in \mathcal{M}_+(X_1, \Sigma_1)$. □

Let us now describe how the measure of a X_1-section of a product set is determined. Based on the Equation (4.10), the measure of the section of $A_1 \times A_2$ is expressed as:

$$\forall A_1 \in \Sigma_1,\ \forall A_2 \in \Sigma_2, \quad (x_1 \longmapsto \mu_2(s_{x_1}(A_1 \times A_2))) = \mu_2(A_2)\,\mathbb{1}_{A_1}. \tag{4.11}$$

4.2.5 Existence and Uniqueness of the Product Measure

Existence

Considering that the measures of sections are measurable as detailed in Section 4.2.4, their integration becomes eligible. The candidate product measure is defined as a function over the product σ-algebra $\Sigma_1 \otimes \Sigma_2$, a concept explored in Section 4.2.2. This function is expressed through the subsequent mathematical expression and is also implemented within Coq as outlined below,

$$(\mu_1 \otimes \mu_2)(A) := \int_{X_1} \mu_2(s_{x_1}(A))\, d\mu_1,$$

```
Definition meas_prod_meas (A : X1 * X2 → Prop) : R̄ :=
  LInt_p muX1 (meas_section muX2 A).
```

It is evident that this candidate function is both nonnegative and attains zero on the empty set. The σ-additivity property is proved through the σ-additivity of the measure `muX2` and that of the integral established by the Equation (3.6) in Section 3.4.1. This verification confirms this function satisfies the properties of a measure, enabling us to formalize the following Coq `record` defining the product measure `meas_prod` as an object of type *measure* and is associated with the product space generated by `genX1xX2` (see Section 3.2 and Section 4.2.2).

```
Definition meas_prod : measure genX1xX2 :=
  mk_measure genX1xX2 meas_prod_meas
    meas_prod_emptyset meas_prod_nonneg
      meas_prod_sigma_additivity.
```

This `record` is provided through the `mk_measure` constructor, where `meas_prod_meas` parameter represents the function that acts as the candidate product measure. This function has been validated to ensure it satisfies the properties of a measure. In Coq, the product measure is captured with the notation `muX1xX2 := meas_prod muX1 muX2`.

Uniqueness

Product measures are proved to preserve the finiteness or σ-finiteness property of the initial measures μ_1 and μ_2. Then, the proof of the uniqueness of the product measure follows a parallel course to the verification of the measurability of the measure of sections, as detailed in Section 4.2.4. Initially, when the measures μ_1 and μ_2 are finite, we introduce two finite product measures, denoted as m and $\tilde{m}$, induced by the measures μ_1 and μ_2, both satisfying the box property (4.8). We then consider the set $\mathcal{S} \overset{\text{def.}}{=} \{A \in \Sigma_1 \otimes \Sigma_2 \,|\, m(A) = \tilde{m}(A)\}$ and prove that it contains $\Sigma_1 \otimes \Sigma_2$ through the application of `monotone_class_Prop`, establishing uniqueness of the product measure (i.e., $m = \tilde{m}$). Following this, the proof is extended to σ-finite measures by using restricted measures.

4.3 Tonelli Theorem

Having defined the product measure in Section 4.2, we are now equipped to explore integration within a product space. Similar to Section 4.2, we assume that the measures are σ-finite, ensuring the existence and uniqueness of the product measure.

In this section, we delve into the proof of the Tonelli theorem, allowing the computation of a double integral on a product space by iteratively integrating with respect to each variable, in either order. Alongside this, the theorem also states measurability properties that guarantee the legitimacy of all integrals. This mathematical concept is expressed as follows,

Theorem 2 (Tonelli). *Let (X_1, Σ_1, μ_1) and (X_2, Σ_2, μ_2) be measure spaces. Assume that μ_1 and μ_2 are σ-finite. Let $f \in \mathcal{M}_+(X_1 \times X_2, \Sigma_1 \otimes \Sigma_2)$. Then, we have*

$$\left(\forall x_1 \in X_1,\ f_{x_1} \in \mathcal{M}_+(X_2, \Sigma_2)\right) \quad \wedge \quad \int_{X_2} f_{x_1}\, d\mu_2 \in \mathcal{M}_+(X_1, \Sigma_1), \tag{4.12}$$

$$\left(\forall x_2 \in X_2,\ f^{x_2} \in \mathcal{M}_+(X_1, \Sigma_1)\right) \quad \wedge \quad \int_{X_1} f^{x_2}\, d\mu_1 \in \mathcal{M}_+(X_2, \Sigma_2), \tag{4.13}$$

and we have

$$\int_{X_1 \times X_2} f\, d(\mu_1 \otimes \mu_2) = \int_{X_1} \left(\int_{X_2} f_{x_1}\, d\mu_2 \right) d\mu_1 \tag{4.14}$$

$$= \int_{X_2} \left(\int_{X_1} f^{x_2}\, d\mu_1 \right) d\mu_2. \tag{4.15}$$

where f_{x_1} and f^{x_2} represent sections of functions that will be defined in the following section.

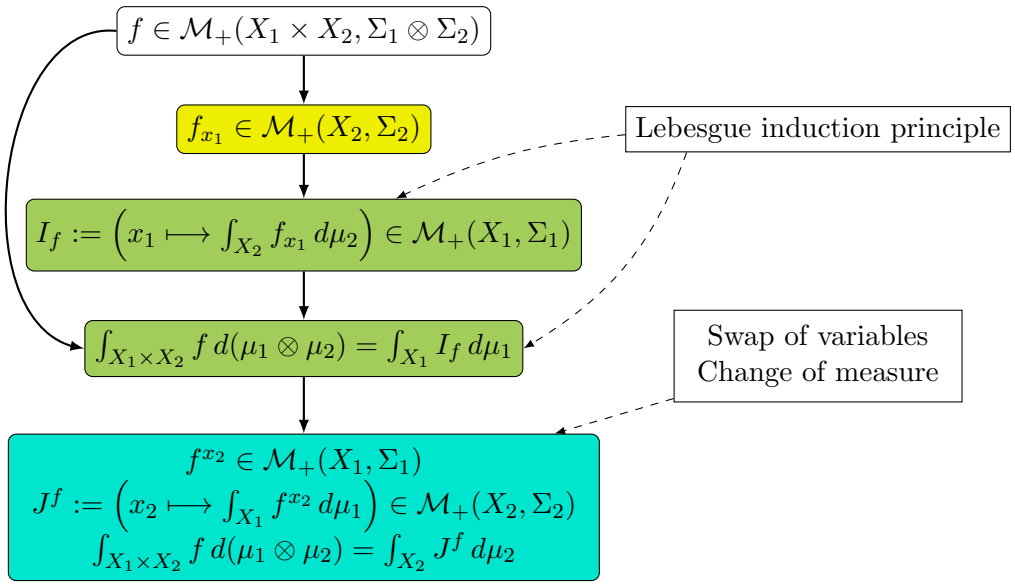

Figure 4.4: Flowchart illustrating the construction of iterated integrals on a product space. The filled colors correspond to Sections: 4.3.1 in yellow, 4.3.2 in green, and 4.3.3 in turquoise blue. Dashed lines indicate the application of the specified proof arguments.

Adopting the same methodology used in Section 4.2, the construction of the iterated integral (represented on the right-hand side of the Equation (4.14)) unfolds in three steps, as illustrated in Figure 4.4. In the first step, we establish the Σ_2-measurability of X_1-sections of functions (refer to Section 4.3.1 for further details about sections of functions). Next, we demonstrate the Σ_1-measurability of the integral computed over X_2 of these sections of functions. The iterated integral is then formed as the integral (in X_1) of the integral (in X_2) of the sections of functions as described in Section 4.3.2. Subsequently, Formula (4.14) is proven, and from this, (4.15) is derived by exchanging variables, using both a change of measure and the uniqueness of the

product measure (see Section 4.3.3).

The main argument driving this proof is the Lebesgue induction principle, as detailed in Section 4.1, applied twice: firstly, to guarantee the measurability of the integral of sections of functions in conjunction with the first Tonelli formula (4.14); and secondly, to establish the change-of-measure formula, which will be further explained in Section 4.3.3.

4.3.1 Section of Functions

Following the same approach for the section of set established in Section 4.2.3, consider a given extended real-valued function $f : X_1 \times X_2 \to \overline{\mathbb{R}}$. For each element x_1 fixed within X_1, the *X_1-section of f at x_1* is defined through partial application with respect to the first variable as:

$$f_{x_1} := (x_2 \mapsto f(x_1, x_2)).$$

```
Definition section_fun (x1 : X1) (f : X1 * X2 → R̄ ) (x2 : X2) : R̄ := f (x1, x2).
```

The mathematical relevance of this definition differs from that outlined in Section 4.2.3, as it focuses on sections of functions rather than sections of subsets.

Similarly, for each element x_2 fixed within the set X_2, the X_2-section of the function f, denoted as f^{x_2}, is defined through partial application with respect to the second variable,

$$f^{x_2} := (x_1 \mapsto f(x_1, x_2)).$$

The measurability of a function with respect to a product σ-algebra guarantees that its sections are measurable with respect to the σ-algebra of the variable that remains unfixed. More precisely, if we have a nonnegative measurable function f defined on $X_1 \times X_2$ with respect to $\Sigma_1 \otimes \Sigma_2$, then the X_1-sections of f will be measurable on X_2 with respect to Σ_2. This concept is formalized in Coq as follows:

```
Lemma section_fun_Mplus : ∀(f : X1 * X2 → R̄ ) (x1 : X1),
  Mplus genX1xX2 f → Mplus genX2 (section_fun x1 f).
```

The utility of sections of function is further enhanced by their compatibility with a range of mathematical operations, including addition and scalar multiplication.

Moreover, the integral of a function defined on a product space is the integral of a section of this function with respect to the unfixed variable. Given the complexity of integrating the function $f(x_1, x_2)$ over its entire domain due to its dependence on multiple variables, one can fix one variable to simplify the integration process. Consequently, for a fixed x_1, the integration of the X_1-section $f_{x_1}(x_2)$ over the domain X_2 can be represented as $\int_{X_2} f_{x_1}(x_2)\, d\mu_2 = \int_{X_2} f(x_1, x_2)\, d\mu_2$.

4.3.2 Iterated Integral and the First Formula of Tonelli Theorem

Given that X_1-sections of measurable functions are nonnegative and measurable with respect to Σ_2 as elaborated in Section 4.3.1, we can compute their integral over X_2. For any function f within the space $\mathcal{M}_+(X_1 \times X_2, \Sigma_1 \otimes \Sigma_2)$, we define the integral I_f as follows,

$$I_f := \left(x_1 \longmapsto \int_{X_2} f_{x_1}\, d\mu_2 \right),$$

This integral is encoded in Coq as,

```
Definition LInt_p_section_fun (f : X1 * X2 → R̄ ) (x1 : X1) : R̄ :=
    LInt_p muX2 (section_fun x1 f).
```

The process of iterated integration introduces an extra step of integration over X_1, requiring a prior confirmation that I_f belongs to $\mathcal{M}_+(X_1, \Sigma_1)$. The nonnegativity property of I_f is a straightforward consequence of the integral's monotonicity property, as discussed in Section 3.4.1. Moreover, the measurability property of I_f, along with the formulation and proof of the first Tonelli formula (4.14), will be established by the use of the Lebesgue induction principle elaborated in Section 4.1.

We express the first Equation of the Tonelli theorem (4.14) as below,

```
Lemma Tonelli_aux1 : ∀f : X1 * X2 → Rbar, Mplus genX1xX2 f →
  Mplus genX1 (LInt_p_section_fun f) ∧
  LInt_p meas_prod f = LInt_p muX1 (LInt_p_section_fun f).
```

In this code snippet, `meas_prod` denotes the product measure as defined in Section 4.2. We initiate the proof of the lemma by introducing a predicate `P` that states the nonnegativity and measurability of I_f and Equation (4.14) for a given function f,

```
Let P (f : X1 * X2 → R̄ ) : Prop :=
  Mplus genX1 (LInt_p_section_fun f) ∧
      LInt_p meas_prod f = LInt_p muX1 (LInt_p_section_fun f).
```

The property `P` will be proven compatible with indicator functions, positive linearity, and the supremum operation for an increasing sequence of measurable functions $(f_n)_{n\in\mathbb{N}}$. This derives from the compatibility of the integral I_f with the same properties, along with previously established results such as the positive linearity property of the Lebesgue integral, the closure properties of $\mathcal{M}_+$, as discussed in Sections 3.1.3 and 3.4.1, and the Beppo Levi theorem, as outlined in Section 3.4.3. In more detail, through the application of the Lebesgue induction principle we get the following steps.

(i) **`P` holds on $\mathcal{IF}(X_1 \times X_2, \Sigma_1 \otimes \Sigma_2)$.**

Our first step in proving the lemma `Tonelli_aux1` (see Equation (4.14)) involves demonstrating that the predicate `P` holds on the set of measurable characteristic functions $\mathcal{IF}$. To achieve this, we verify the following equality for a measurable subset A and any given $x_1 \in X_1$,

$$I_{\mathbb{1}_A}(x_1) = \int_{X_2} \mathbb{1}_{s_{x_1}(A)}(x_1)\, d\mu_2 = \mu_2(s_{x_1}(A)) \in \mathcal{M}_+(X_1, \Sigma_1),$$

where $\mathbb{1}_A$ denotes the characteristic function of a subset A. This equality is a consequence of Equation (3.5) page 29, and the measurability of the subset of sections as detailed in Section 4.2.3, as well as the measurability of their measures. This is useful to validate the Equation (4.14) specifically for characteristic functions within the Tonelli theorem, illustrated by the Equation below,

$$\int_{X_1\times X_2} \mathbb{1}_A\, d(\mu_1 \otimes \mu_2) = \int_{X_1} I_{\mathbb{1}_A}(x_1)\, d\mu_1$$

(ii) **`P` holds on $\mathcal{SF}_+(X_1 \times X_2, \Sigma_1 \otimes \Sigma_2)$.**

In the second step, we extend the validity of the predicate `P` to the set of nonnegative simple functions $\mathcal{SF}_+$. The integral of the mapping $I := (f \mapsto I_f)$, is proven to be positively linear. We verify the following lemmas,

```
Lemma LInt_p_section_fun_meas_prod_plus : ∀(f g : X1 * X2 → R̄ ),
  Mplus genX1xX2 f → Mplus genX1xX2 g →
    P f → P g → P (fun x ⇒ f x +_R̄ g x).

Lemma LInt_p_section_fun_meas_prod_scal : ∀(a : ℝ) (f : X1 * X2 → R̄ ),
  0 ⩽ a → Mplus genX1xX2 f → P f → P (fun x ⇒ a *_R̄ f x).
```

The proof of these lemmas is straightforward, deriving from the application of the positive linearity of the integral I_f.

(iii) `P` **holds on** $\mathcal{M}_+(X_1 \times X_2, \Sigma_1 \otimes \Sigma_2)$**.**

Subsequently, we demonstrate that the predicate `P` applies to the set of nonnegative measurable functions $\mathcal{M}_+$. This means that `P` is compatible with the supremum, in addition to the first two steps (i) and (ii) of this proof. According to the theorem of Beppo Levi (monotone convergence) (referenced in Section 3.4.3), we prove that the mapping I is compatible with the supremum. For any nondecreasing sequence $(f_n)_{n\in\mathbb{N}}$ within $\mathcal{M}_+(X_1 \times X_2, \Sigma_1 \otimes \Sigma_2)$,

$$I_{\sup_{n\in\mathbb{N}} f_n} = \sup_{n\in\mathbb{N}} I_{f_n}$$

Therefore, within Coq, we establish the following lemma, which validates the compatibility of `P` with the supremum of nondecreasing, nonnegative simple functions,

```
Lemma LInt_p_section_fun_meas_prod_Sup_seq :
  ∀ f, incr_fun_seq f → Mplus_seq genX1xX2 f →
    (∀ n, P (f n)) → P (fun x ⇒ Sup_seq (fun n ⇒ f n x)).
```

We thus conclude that property `P` holds on $\mathcal{M}_+$, which in turn confirms the validity of the first formula (Equation (4.14)) of the Tonelli theorem.

4.3.3 Change of Measure, Second Formula, and Tonelli Theorem

The proof of the second Equation of the Tonelli theorem, as referenced in Equation (4.15), can be proved using the same path as the first Tonelli formula (4.14). This involves using the sections of function with respect to the second variable, introducing J^f as,

$$J^f := \left(x_2 \longmapsto \int_{X_1} f^{x_2}\, d\mu_1\right)$$

demonstrating that this function belongs to $\mathcal{M}_+(X_2, \Sigma_2)$ as illustrated in Figure 4.4, and establishing the stated equality through the application of the Lebesgue induction principle.

Although this strategy is easy, it is quite lengthy and redundant. Instead, we simply switch the roles of the two variables, expressing the previous result for functions of the type `X2 * X1 → R̄`. Subsequently, the challenging aspect lies in a change of measure that brings back the original function type `X1 * X2 → R̄`.

The concept of changing measures uses the idea of an *image measure*, which is also referred to as a *pushforward measure*. This concept is detailed, in [22, Section 2.6]. In this process, the measure is shifted between σ-algebras, moving specifically from $\Sigma_2 \otimes \Sigma_1$ to $\Sigma_1 \otimes \Sigma_2$.

Change of Measure

In the setting of measurable spaces (X_1, Σ_1) and (X_2, Σ_2), consider a measurable function $h : X_1 \to X_2$. Given a measure μ_1 on (X_1, Σ_1), the *image measure of* μ_1 *under the map* h is constructed as a measure on (X_2, Σ_2) defined by,

$$h\#\mu_1 \stackrel{\text{def.}}{=} \mu_1 \circ h^{-1}.$$

In this context, h^{-1} refers to the pre-image operation of measurable subsets under h, not the inverse of the function h. In Coq, this concept is represented as `meas_image h Mh mu`. The verification of the properties of the image measure $h\#\mu_1$ relies on the properties of the measure μ_1, and the measurability of h.

Given $f \in \mathcal{M}_+(X_2, \Sigma_2)$, the compatibility of measurability with function composition ensures that $f \circ h \in \mathcal{M}_+(X_1, \Sigma_1)$. This premise allows the formulation of the change-of-measure formula, which is articulated in both mathematical expression and within Coq as follows:

$$\int_{X_2} f \, d(h\#\mu_1) = \int_{X_1} f \circ h \, d\mu_1. \tag{4.16}$$

```
Lemma LInt_p_change_meas : ∀(h : X1 → X2) (Mh : measurable_fun genX1 genX2 h)
  (f : X2 → R̄ ), Mplus genX2 f →
  LInt_p (meas_image h Mh muX1) f = LInt_p muX1 (fun x : X1 ⇒ f (h x)).
```

The proof of this lemma uses the principle of induction based on the hypothesis of `f`. This approach is done as `Mplus` is shown to be equivalent to `Mp` in Section 4.1.4. More specifically, we demonstrate that this equality is compatible with characteristic functions, positive linearity, and the supremum operation. These properties are derived from the integral properties, including positive linearity, and the theorem (1) (Beppo Levi, monotone convergence) in Section 3.4.3.

Swap Function

We define the product measure, denoted by $\mu_{12} := \mu_1 \otimes \mu_2$ on the product space $(X_1 \times X_2, \Sigma_1 \otimes \Sigma_2)$, as induced by μ_1 and μ_2, according to Section 4.2. Conversely, by interchanging the roles of the two spaces, we pose $\mu_{21} := \mu_2 \otimes \mu_1$ as the product measure on $(X_2 \times X_1, \Sigma_2 \otimes \Sigma_1)$, which is represented in Coq as,

$$\texttt{muX2xX1} := \texttt{meas\_prod muX2 muX1}.$$

We consider the function of the swap of variables $h := (x_2, x_1) \longmapsto (x_1, x_2)$ and prove its measurability `Mh`, we then prove that the image measure $h\#\mu_{21}$, is indeed a product measure on the product space $(X_1 \times X_2, \Sigma_1 \otimes \Sigma_2)$, induced by μ_1 and μ_2. In the Coq proof assistant, this is represented as `meas_prod_swap` := `meas_image h Mh muX2xX1`.

Second Formula of Tonelli Theorem

Having gathered all the necessary components, we are now ready to establish the second formula of the Tonelli theorem (4.15). For a function $f \in \mathcal{M}_+(X_1 \times X_2, \Sigma_1 \otimes \Sigma_2)$, we have $f \circ h \in$

$\mathcal{M}_+(X_2 \times X_1, \Sigma_2 \otimes \Sigma_1)$. Using the section of functions with respect to x_2 (as detailed in Section 4.3.1), we obtain for any x_2 in X_2,

$$f^{x_2} := (x_1 \longmapsto f(x_1, x_2)) = (x_1 \longmapsto f \circ h(x_2, x_1)) = (f \circ h)_{x_2}. \tag{4.17}$$

This leads us to the equalities,

$$\int_{X_1 \times X_2} f \, d\mu_{12} \overset{(a)}{=} \int_{X_1 \times X_2} f \, d(h\#\mu_{21}) \overset{(b)}{=} \int_{X_2 \times X_1} f \circ h \, d\mu_{21}$$
$$\overset{(c)}{=} \int_{X_2} \left(\int_{X_1} (f \circ h)_{x_2} \, d\mu_1 \right) d\mu_2 \overset{(d)}{=} \int_{X_2} \left(\int_{X_1} f^{x_2} \, d\mu_1 \right) d\mu_2.$$

The uniqueness of the product measure, as established in Section 4.2, implies that the image measure under the swap function satisfies $h\#\mu_{21} = \mu_{12}$, thereby establishing assertion (a). Assertion (b) follows from the application of the change-of-measure formula (4.16). The application of the first formula of Tonelli's theorem (4.14) to the space $X_2 \times X_1$ leads to (c), while Equation (4.17) yields result (d).

In Coq, we use `swap f` to express the composition $f \circ h$. The second Equation of Tonelli's theorem, as articulated in Equation (4.15), is expressed in the Coq environment as follows,

```
Lemma Tonelli_aux2 : ∀f, Mplus genX1xX2 f →
  Mplus genX2 (LInt_p_section_fun muX1 (swap f)) ∧
  LInt_p meas_prod_swap f = LInt_p muX2 (LInt_p_section_fun muX1 (swap f)).
```

This completes the demonstration of the second formula of the Tonelli Theorem.

Statement of Tonelli Theorem

Lastly, consider X_1 and X_2 any type, and that μ_1 and μ_2 are σ-finite measures, we obtain a comprehensive theorem that legitimates all integrals, as presented in Tonelli's Theorem,

```
Context {X1 X2 : Type}.
Context {genX1 : (X1 → Prop) → Prop}.
Context {genX2 : (X2 → Prop) → Prop}.

Let genX1xX2 := Gen_Product genX1 genX2.

Variable muX1 : measure genX1.
Variable muX2 : measure genX2.
Hypothesis HmuX1 : is_sigma_finite_measure genX1 muX1.
Hypothesis HmuX2 : is_sigma_finite_measure genX2 muX2.

Let muX1xX2 := meas_prod muX1 muX2 HmuX2.

Theorem Tonelli : ∀(f : X1 * X2 → R̄ ), Mplus genX1xX2 f →
   (Mplus genX1 (LInt_p_section_fun muX2 f) ∧
   (Mplus genX2 (LInt_p_section_fun muX1 (swap f)) ∧
     LInt_p muX1xX2 f = LInt_p muX1 (LInt_p_section_fun muX2 f)) ∧
     LInt_p muX1xX2 f = LInt_p muX2 (LInt_p_section_fun muX1 (swap f))).
```

In this code snippet, `Gen_Product` is a function that constructs a generator for the product of two measurable spaces X_1 and X_2, as defined in Section 4.2.2, and `meas_prod` denotes the product measure as defined in Section 4.2.

Part II

Formalization of Simplicial Finite Elements

Chapter 5

Algebra

This chapter covers some algebraic developments necessary for the formalization of finite elements and the formal proof of their properties. These complements are mainly based on the Coquelicot and math-comp libraries, as discussed in Section 2.2 and 2.3, respectively. Coquelicot is essential for its hierarchy of canonical structures that represent mathematical algebraic structures, while math-comp is primarily used for handling finite types (ordinals), iterated operators (`bigop`), and binomial coefficients. The content and structure of this chapter are partly inspired by the works of Gostiaux [41].

This chapter is organized into various sections exploring distinct concepts and structures within algebra. Section 5.1 discusses subsets, images, pre-images, and bijective functions, along with the bijectivity property restricted on subsets. Section 5.2 highlights the principle of double induction, addressing finite families and ordinal numbers. Section 5.3 delves into algebraic structures, covering monoids, groups, module spaces, and affine spaces, including the concept of barycenters. Section 5.4 focuses on finite dimensional subspaces, introducing topics such as linear span, basis families, affine independence, and the dual space concept. Finally, Section 5.5 provides an overview of binomial coefficients.

From this chapter through to the end of the report, certain notations will be presented in boldface. This indicates that they represent families (such as vectors in $\mathbb{R}^n$) containing multiple elements, distinguishing them from scalars in $\mathbb{R}$ or $\mathbb{N}$, for example.

5.1 Functions and restrictions

This section starts with an introduction of the concept of subsets, as detailed in Section 5.1.1. Following this, Section 5.1.2 focuses on the direct and reciprocal images of subsets under mappings, and explores the concept of function composition. The discussion then progresses to bijective functions in Section 5.1.3, including the existence of bijective inverse functions. Finally, Section 5.1.4 provides the properties of bijectivity when restricted to subsets, demonstrating how bijective functions can retain their characteristics within specific subsets of a given type.

5.1.1 Subsets

In Coq, we define a generic set `U : Type`. The subsets of `U` can be represented as a predicate by the type `U → Prop`. This sets the stage for discussing properties and operations on subsets of `U`. We start by defining a full set within the set `U`, which includes all possible elements of `U`, and always returns `True`.

```
Definition fullset : U → Prop := fun _ ⇒ True.
```

We address the definition of the inclusion relation between two subsets, `A` and `B` of `U`. The relation is described mathematically as follows:

$$A \subset B \stackrel{\text{def.}}{=} \forall x \in U,\ x \in A \Longrightarrow x \in B.$$

This concept is implemented in Coq for subsets `A, B : U → Prop` by the following definition:

```
Definition incl (A B : U → Prop) : Prop := ∀x : U, A x → B x.
```

Furthermore, for any element x in U, there exists a set known as a *singleton*, denoted by $\{x\}$, which consists solely of the element x. This concept is formalized in Coq as:

```
Definition singleton : U → U → Prop := fun x y : U ⇒ x = y.
```

This defines a predicate that takes two arguments `x` and `y` of type `U`, and returns a proposition that states whether `x` is equal to `y`.

5.1.2 Image, Pre-image and Composition of Functions

We now explore the notions of direct and reciprocal image (pre-image) of a subset ([41, Section 1.3 p.16]). Consider `U1` and `U2` two types. A function, denoted as `f : U1 → U2`, maps each element `x1` of the domain `U1` to an element `x2` in the codomain `U2`. This mapping is also referred to as an *application*, and the terms are used interchangeably.

Let A_1 be a subset of U_1. The *direct image* of A_1 under the function f, denoted $f(A_1)$, is defined mathematically as:

$$f(A_1) \stackrel{\text{def.}}{=} \{\ x_2 \in U_2 \mid \exists x_1 \in A_1,\ f(x_1) = x_2\}.$$

In Coq, this concept is captured through the following inductive definition:

```
Inductive image {U1 U2 : Type} (f : U1 → U2) (A1 : U1 → Prop) : U2 → Prop :=
  Im : ∀ x1 : U1, A1 x1 → image f A1 (f x1).
```

Conversely, the *pre-image* (or *reciprocal image*) of a subset A_2 of U_2 under f is defined as:

$$f^{-1}(A_2) \stackrel{\text{def.}}{=} \{x_1 \in U_1 \mid f(x_1) \in A_2\}.$$

The Coq implementation of the pre-image is straightforward:

```
Definition preimage := fun {U1 U2 : Type} (f : U1 → U2) (A2 : U2 → Prop) (x1 : U1) ⇒ A2 (f x1).
```

Let us now discuss the composition of two functions. Consider `U1`, `U2`, and `U3` three sets of any `Type`, and two functions, `f : U1 → U2` and `g : U2 → U3`. Function composition involves creating a new function that maps the output of one function into the input of another ([41, Section 1.4 p.18]). Specifically, we define a function that first applies `f` and then `g` to the result of `f`, forming a new function denoted `g ∘ f` from `U1` to `U3`. It is important to note the sequence in which `f` and `g` are applied.

```
Definition compose : U1 → U3 := fun x1 : U1 ⇒ g (f x1).
```

5.1.3 Bijective Functions

Consider two types, `U1` and `U2`. We present the properties of functions between these sets, specifically focusing on injectivity, surjectivity, and bijectivity ([41, Section 1.4 p.17]).

A function `f : U1 → U2` is defined as *injective* if it does not map two distinct elements of `U1` to the same element in `U2`. This property guarantees that each element of the domain `U1` corresponds uniquely to an element in the codomain `U2`. The formal definition in Coq is:

```
Definition injective := ∀x1 x2 : U1, f x1 = f x2 → x1 = x2.
```

In simpler terms, f is injective if and only if for every singleton $\{x_2\}$ of `U2`, the pre-image $f^{-1}(\{x_2\})$ is either empty or a singleton.

A function `f : U1 → U2` is *surjective* if every element in `U2` has at least one pre-image in `U1`. This property ensures that the image of the function covers the entire codomain `U2` (i.e., `f(U1)` = `U2`). It is expressed in Coq as:

```
Definition surjective (f : U1 → U2) : Prop := ∀x2 : U2, ∃x1 : U1, f x1 = x2.
```

The definition of `injective` is already present in the standard library of Coq, while `surjective` was added during the development of this work.

We now state that a function `g : U2 → U1` satisfies the cancellation property, which means that composing a function `f : U1 → U2` with `g` yields the identity function.

```
Definition cancel := fun (f : U1 → U2) (g : U2 → U1) ⇒ ∀x1 : U1, g (f x1) = x1.
```

This leads to the bijectivity property. A function is considered *bijective* if it admits a left and right inverse for the composition of functions. This dual property establishes a one-to-one correspondence between all elements of `U1` and `U2`. The Coq definition of a bijective function is:

```
Variant bijective (A B : Type) (f : B → A) : Prop :=
    Bijective : ∀g : A → B, cancel f g → cancel g f → bijective f.
```

In this context, the `Variant` type is similar to the `Inductive` type, but it cannot be used to perform a proof by induction.

We can now transform the inductive type `bijective`, which involves weak existential quantification, into a strong existential quantification as established through this lemma:

```
Lemma bij_EX : bijective f → {g : U2 → U1 | cancel f g ∧ cancel g f}.
```

From the bijective property of f, we deduce the existence of the inverse function, denoted as f^{-1}, which maps each element of U_2 back to its unique pre-image in U_1.

```
Definition f_inv {f : U1 → U2} (Hf : bijective f) : U2 → U1 := proj1_sig (bij_EX Hf).
```

Here, `proj1_sig` is used to extract the witness function component from the proof that such an inverse exists under the assumption that `f` is bijective. Thus, we deduce that the inverse mapping f^{-1} is also bijective.

```
Lemma f_inv_bij : ∀{f : U1 → U2} (Hf : bijective f), bijective (f_inv Hf)
```

It is important to distinguish between the notation $f^{-1}(A_2)$ for the pre-image and the inverse function f^{-1}. While both use similar symbols, the pre-image represents a set of elements in U_1, whereas the inverse function maps each element of U_2 directly back to U_1.

5.1.4 Bijective Functions on Subsets

In this section, we focus on applying the bijectivity property specifically to subsets rather than entire sets. This discussion serves as a continuation and a more focused examination of the general concepts of bijectivity introduced earlier. Throughout this section, we append the terms used with the suffix "S" to indicate that the definitions, properties, and lemmas are exclusively related to restrictions of functions to subsets.

We initiate by defining two types, `U1` and `U2`, and their subsets, `P1` and `P2`, respectively. We then consider a function `f : U1 → U2` and present its properties when limited to these specific subsets. The reason we need to explicitly address bijectivity restricted to the subsets `P1` and `P2`, rather than the entire sets `U1` and `U2`, is that we will later study the bijectivity of certain linear forms restricted to polynomial spaces, instead of the full space of functions of infinite dimension.

The following function computes the image of `f` over a subset defined by `P1`. It is defined as:

```
Definition RgS := fun (P1 : U1 → Prop) (f : U1 → U2) ⇒ image f P1.
```

This set contains all the images of elements from `P1` produced by the function `f` (refer to Section 5.1.2).

We further proceed to check whether the range of `f` from type `U1` to `U2`, when restricted to a subset `P1`, is a subset of `P2`. This inquiry is formalized in Coq with the following definition:

```
Definition funS (f : U1 → U2) : Prop := incl (RgS P1 f) P2.
```

Here, `incl` stands for inclusion (refer to Section 5.1.1). The function `funS` formalizes the idea of a restricted function on a subset.

A function `f` is considered injective when restricted to a subset `P1` if

```
Definition injS : Prop := ∀x1 y1 : U1, P1 x1 → P1 y1 → f x1 = f y1 → x1 = y1.
```

This definition ensures that no two distinct elements within `P1` map to the same element in `U2`, thereby preserving the injectivity of the function within the subset.

A function `f` is surjective on a subset `P2`, restricted to `P1`, if

```
Definition surjS (f : U1 → U2) : Prop := ∀x2 : U2, P2 x2 → ∃x1 : U1, P1 x1 ∧ f x1 = x2.
```

This ensures every element of `P2` is an image of some element from `P1`, covering `P2` entirely with images from `P1`.

As for the bijectivity, we state that for every element `x1` in the subset `P1`, applying `f : U1 → U2` followed by `g : U2 → U1` results in the identity. This definition ensures that `g` is a left inverse of `f` on the subset `P1`.

```
Definition canS (P1 : U1 → Prop) (f : U1 → U2) (g : U2 → U1) : Prop :=
  ∀ x1 : U1, P1 x1 → g (f x1) = x1.
```

This defines the concept of a cancellation property between two functions `f` and `g`, which are presumed to act as inverses on the subset defined by `P1`.

Putting all parts together, we now establish the bijective relationship of two functions `f : U1 → U2` and `g : U2 → U1` between the subsets `P1` and `P2` of `U1` and `U2`, respectively. This is stated as follows:

```
Definition bijS_spec := fun (P1 : U1 → Prop) (P2 : U2 → Prop) (f : U1 → U2) (g : U2 → U1) ⇒
  funS P1 P2 f ∧ funS P2 P1 g ∧ canS P1 f g ∧ canS P2 g f.
```

This specification states that the entire image of `f`, when applied to elements that satisfy `P1`, falls within `P2`, and similarly, the image of `g`, when applied to elements that satisfy `P2`, falls within `P1`. It also ensures that `f` and `g` are proper inverses within these subsets, effectively maintaining a mutual inclusion of function images and establishing a bijection between the subsets `P1` and `P2`.

Ultimately, a function `f` is asserted to be bijective if there exists a potential inverse function `g` that satisfies `bijS_spec`. This relationship is given by:

```
Variant bijS (P1 : U1 → Prop) (P2 : U2 → Prop) (f : U1 → U2) : Prop :=
    BijS_ : ∀ g :  U2 → U1, bijS_spec P1 P2 f g → bijS P1 P2 f.
```

The `bijS` predicate in Coq is used to check if a function `f` between two types, `U1` and `U2`, is a partial bijection that meets the conditions specified by the predicates `P1` and `P2` in the definition `bijS_spec`.

Since `bijS` is defined as an inductive type equivalent to weak existential quantification, we can transform it into strong existential quantification by employing the bijective specifications as follows:

```
Lemma bijS_EX : ∀(P1 :  U1 → Prop) (P2 : U2 → Prop) (f : U1 → U2),
  bijS P1 P2 f → {g : U2 → U1 | bijS_spec P1 P2 f g }.
```

This lemma directly follows from `ex_EX`, an alias for the axiom `constructive_indefinite_description`. For more details on this axiom, refer to Section 2.1.

Once we assume that f is bijective on subsets, the next step is to extract the actual inverse function f^{-1} from the proof provided by `bijS_EX`.

```
Definition f_invS := fun (P1 : U1 → Prop) (P2 : U2 → Prop) (f : U1 → U2) (Hf : bijS P1 P2 f) ⇒
  proj1_sig (bijS_EX Hf).
```

This inverse function f^{-1} is constructed from the bijection f and is verified to be bijective from the subset $P_2 \subseteq U_2$ back to $P_1 \subseteq U_1$:

```
Lemma f_invS_bijS : ∀(P1 : U1 → Prop) (P2 : U2 → Prop) (f : U1 → U2) (Hf : bijS P1 P2 f),
  bijS P2 P1 (f_invS Hf).
```

5.2 Ordinals and Finite Families

Section 5.2.1 introduces the principle of double induction, an extension of mathematical induction that is applied simultaneously to two variables. The study of finite ordinal numbers is the focus of Section 5.2.2. Building on this, Section 5.2.3 explores the concept of finite families, which can be understood as vectors of known size, defined as functions that map a finite ordinal type to any other type, providing some concepts necessary to manage these families.

5.2.1 Principle of Double Induction

Double induction is a mathematical technique that essentially involves using the principle of mathematical induction simultaneously on two variables (see Figure 5.1). While simple induction begins by proving a property for a base case, when $n = 0$, it then assumes the property holds for an arbitrary `n` and proves it for `n`+1, double induction extends this concept to two dimensions. We study the double induction from 1 instead of 0, as this approach will

be useful later for proving the unisolvence property of Lagrange finite elements in Section 10.3.7.

We formalize this principle in Coq as follows:

```
Lemma nat_ind2_alt_11 : ∀(P : nat → nat → Prop),
  (∀ m, 0 < m → P m 1) → (∀ n, 0 < n → P 1 n) →
  (∀ m n, 0 < m → 0 < n → P m.+1 n → P m n.+1 → P m.+1 n.+1) →
   ∀ m n, 0 < m → 0 < n → P m n.
```

This lemma consists of proving a property `P m n` for all positive integers `m` and `n` using a stepwise approach. We first establish the *base cases* for the lines `n`=1 and `m`=1. Then, we use the inductive hypothesis to extend these base cases to all (`m`, `n`) pairs by a kind of "staircase" logic: if the property holds at the next step horizontally or vertically, it can be extended diagonally.

In more detail, the figure 5.1 represents a graphical depiction of a double induction method used to prove a property `P m n` for all positive `m` and `n`, as described in the lemma `nat_ind2_alt_11`. The lemma establishes two base cases: $(\forall\ \mathtt{m},\ 0 < \mathtt{m} \to \mathtt{P\ m\ 1})$ and $(\forall\ \mathtt{n},\ 0 < \mathtt{n} \to \mathtt{P\ 1\ n})$. In the diagram, these are illustrated by the green horizontal line at $\mathtt{n} = 1$ and the green vertical line at $\mathtt{m} = 1$. Each orange dot on these lines represents that `P m 1` and `P 1 n` hold for all positive `m` and `n`, respectively. The inductive step in the lemma states: $(\forall\ \mathtt{m\ n},\ 0 < \mathtt{m} \to 0 < \mathtt{n} \to \mathtt{P\ m.+1\ n} \to \mathtt{P\ m\ n.+1} \to \mathtt{P\ m.+1\ n.+1})$. In the figure, this is depicted by showing that once `P m.+1 n` (a step to the right on the grid, shown as an orange dot to the left of the green dot) and `P m n.+1` (a step up on the grid, shown as an orange dot below the green dot) are true, then `P m.+1 n.+1` (the green dot) is also true. The arrows show the direction of the induction, moving rightward and upward to establish `P m.+1 n.+1`.

We have also formalized other lemmas analogous to `nat_ind2_alt_11` for nonnegative integers; however, these are not used in subsequent developments.

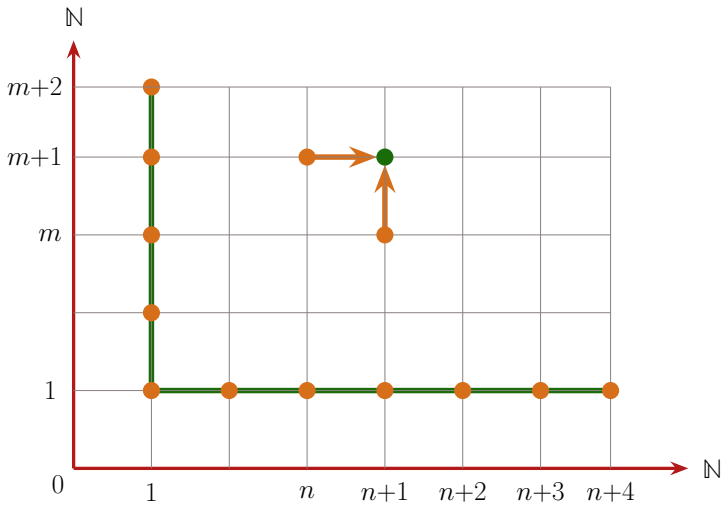

Figure 5.1: Visual representation of the lemma `nat_ind2_alt_11` with `P(m, n)` being proven for all positive `m` and `n` through base cases and inductive steps as double induction.

5.2.2 Ordinals

In this thesis, we are interested in the study of finite ordinals, which are defined to represent elements strictly smaller than a given natural number `n`. In this section, we give some specific utility functions and properties designed to manipulate and transform ordinal types,

complementary to those of the math-comp library [56, Section 7.4] discussed in Section 2.3. We will need to study ordinals as they will be used later in our work to construct finite families mapping from some ordinal type to any other type.

We recall that the finite type 'I_n represents the set of ordinals strictly less than n. In other words, 'I_n corresponds to the set of natural numbers $[0..n-1]$. The notation 'I_{m,n} is defined to represent functions from one ordinal to another, specifically 'I_m → 'I_n. This means that any element of 'I_{m,n} is a function that maps elements from the set $[0..m-1]$ to the set $[0..n-1]$. Similarly, 'I_[n] is a simplified notation. It denotes the set of functions 'I_{n,n}, which includes all functions from $[0..n-1]$ to itself.

We present operations designed to manipulate finite ordinals, enabling both increment and decrement of ordinal values.

The lower_S operation is designed to decrease the value of an ordinal by one by shifting it down from its current set 'I_n.+1 to the smaller set 'I_n, provided that it is not the smallest ordinal, ord0 (see Section 2.3).

```
Definition lower_S : ∀{n : nat} {i : 'I_n.+1}, i ≠ ord0 → 'I_n.
```

This function constructs a new ordinal value, explicitly in the set 'I_n. We confirm the correctness of this operation, by verifying that the value of the result is indeed equal to i.−1.

```
Lemma lower_S_correct : ∀{n} {i : 'I_n.+1} (H : i ≠ ord0), lower_S H = i.−1 :> nat.
```

The notation :> is used for type coercion, which means that lower_S H is converted from its type 'I_n to a natural number type (see Section 2.1 for more details on coercions).

Conversely, the lift_S function increments the value of an ordinal by shifting it from a set of size n to a set of size $n+1$.

```
Definition lift_S {n} (i : 'I_n) : 'I_n.+1 := lift ord0 i.
```

Essentially, lift_S is a specialized case of the more general lift function defined in Section 2.3, that increments the value of ordinal i from the beginning.

Again, we verify the correctness of this definition as the value of the result is equal to i.+1.

```
Lemma lift_S_correct : ∀{n} (i : 'I_n), lift_S i = i.+1 :> nat.
```

Progressing further, we explore an interesting aspect of ordinal manipulation through the skip_ord function. This function maps an index from a smaller set, 'I_n, to a larger one, 'I_n.+1, while intentionally *skipping* over a specified position, denoted by i0. This operation is delineated as follows:

```
Definition skip_ord {n} (i0 : 'I_n.+1) (j : 'I_n) : 'I_n.+1 := lift i0 (cast_ord (pred_Sn n) j).
```

The functions lift and cast_ord are defined in Section 2.3. This precise adjustment is supported by the proof pred_Sn n, which validates that n = n.+1.−1 for the casting function to operate correctly. Consider the case where n = 3, which means 'I_3 includes the elements {0, 1, 2} and 'I_4 extends to {0, 1, 2, 3}. Selecting i0 ordinal of value 2 in 'I_4 as the position to skip, we apply the skip_ord function to map each ordinal from 'I_3 to 'I_4. This mapping process ensures that all elements of 'I_3 find corresponding positions in 'I_4, except for the skipped position i0 of value 2, which does not correspond to any original element from 'I_3.

Transitioning to operations involving concatenated sets, the concat_l_ord function manages the canonical projection from a larger set, 'I_(n1 + n2), onto its first segment, 'I_n1, given the proof that the index is within the bounds of n1.

```
Definition concat_l_ord : ∀{n1 n2 : nat} {i : 'I_(n1 + n2)}, i < n1 → 'I_n1.
```

Conversely, the concat_r_ord function handles the projection of indices from the concatenated set 'I_(n1 + n2) onto the second segment 'I_n2, with an appropriate downshift in index values.

```
Definition concat_r_ord : ∀{n1 n2 : nat} {i : 'I_(n1 + n2)}, ¬ i < n1 → 'I_n2.
```

5.2.3 Finite Family

This section builds upon the previous one and is dedicated to finite families that are functions from some ordinal type 'I_n to any type E. For instance, these families can represent vectors or matrices when E is a module space. The notation 'E^n of 'I_n → E refers to the set of functions from 'I_n to E. In other words, it represents the type of finite families of elements from E, where the indices are elements of 'I_n. Furthermore, 'E^0 denotes the type of functions from the empty set 'I_0 to E. Thus, 'E^0 is a type with a single element, called unit type. The names of statements in this section will have the suffix "F" to indicate families.

Constructors of Finite Families

Throughout this section, we declare a context where E, F, and G represent any type. A finite family A, denoted as an n-family, contains n items of type E, and is typically expressed as A : 'I_n → E or A : 'E^n, where 'I_n represents the set of indices.

Let us provide some specific types of finite families. A *constant* family is a vector of n-items where every element is identical. The implementation of such a family can be illustrated by the following Coq definition:

```
Definition constF n (x : E) : 'E^n := fun _ ⇒ x.
```

This function takes the size n of the family, and an element x of type E, and returns a family of type 'E^n where each of the n elements is the element x. This function uses a lambda expression fun _ ⇒ x to create a function that ignores its argument (indicated by '_') and consistently returns x. Consider an example when n=4 and x=2 we obtain constF 4 2 = (2,2,2,2).

The correctness of this definition is verified by the lemma:

```
Lemma constF_correct : ∀n (x : E) (i : 'I_n), constF n x i = x.
```

Single-element families are a specialized type of constant family constF that contains only one item x0. The definition in Coq can be represented as:

```
Definition singleF (x0 : E) : 'E^1 := constF 1 x0.
```

This function differs from the predicate singleton defined in Section 5.1.1, as it is a function that returns a finite family of size 1, rather than a predicate comparing two elements.

Operations on Finite Families: Casting and Resizing in Coq

We turn our attention to several useful operations that manipulate finite families, building on the properties developed for ordinals in Section 5.2.2.

Finite families can interact and relate to each other through inclusion operation, ensuring for example that elements of one family are contained within another. Consider a predicate PE defining a subset of the set E, we define a predicate asserting that all items of the n-family A belong to PE. This can be formalized in Coq with the following definition:

```
Definition inclF {n} (A : 'E^n) (PE : E → Prop) : Prop := ∀i, PE (A i).
```

By using this predicate, we can establish a relationship between two families A1 and A2 of potentially different sizes, n1 and n2 respectively, where all items of one must appear in the other,

```
Definition invalF {n1 n2} (A1 : 'E^n1) (A2 : 'E^n2) : Prop :=
  inclF A1 (fun x : E ⇒ ∃i : 'I_n2, x = A2 i)
```

An important aspect to note is that if A1 contains duplicate items, it is possible for the definition invalF to hold true even if $n2 < n1$. For example, we define two families $A1 = (3, 3, 5)$ and $A2 = (3, 5)$. Since every element in A1 has a corresponding element in A2, we can say that the proposition invalF A1 A2 holds true, indicating that all elements of A1 are included in A2, despite A2 being shorter than A1.

Revisiting the operator cast_ord of type casting of ordinals explored in Section 2.3, the castF function extends the cast_ord operation to entire families. Given a proof that the two sizes n1 and n2 are equal, castF transforms a family A1 of type 'E^n1 into a family of type 'E^n2, while maintaining the order of elements.

```
Definition castF {n1 n2} (H : n1 = n2) (A1 : 'E^n1) : 'E^n2 :=
  fun i2 : 'I_n2 ⇒ A1 (cast_ord (eq_sym H) i2).
```

Expanding upon the ordinal functions lift_S and skip_ord outlined in Section 5.2.2, we adapt these operations to manipulate finite families. The liftF_S function removes the first item from an n.+1-family, shifting all its subsequent items.

```
Definition liftF_S {n} (A : 'E^n.+1) : 'E^n := fun i ⇒ A (lift_S i).
```

Similarly, the skipF function creates a new family by omitting the i0-th item from an n.+1-family.

```
Definition skipF {n} (A : 'E^n.+1) (i0 : 'I_n.+1) : 'E^n := fun (j : 'I_n) ⇒ A (skip_ord i0 j).
```

Another quite similar operation is the replaceF function, which creates a new family by replacing the element at a specific index i0 with a new value x0, while leaving all other elements unchanged. This function is formalized as follows:

```
Definition replaceF {n} (A : 'E^n) (x0 : E) (i0 : 'I_n) : 'E^n :=
  fun i : 'I_n ⇒ match (ord_eq_dec i i0) with
    | left _ ⇒ x0
    | right _ ⇒ A i
    end.
```

Let us illustrate with a simple example. Consider a family A = (2, 5, 8) of type 'nat^3. Assuming we want to replace the element at index i0 of value 1 in 'I^3, with a new value x0 = 10. The replaceF A x0 i0 outputs (2, 10, 8).

Additionally, we can concatenate two families A1 and A2 by combining them into a unified family $A = A_1 \oplus A_2$ through the concatF function defined as a patern matching, preserving the original order of elements from both.

```
Definition concatF {n1 n2} (A1 : 'E^n1) (A2 : 'E^n2) : 'E^(n1 + n2) := fun i: 'I_(n1 + n2) ⇒
  match (lt_dec i n1) with
    | left H ⇒ A1 (concat_l_ord H)
```

```
  | right H ⇒ A2 (concat_r_ord H)
  end.
```

This function uses the `lt_dec` decision function to determine whether `i < n1` or not, using ordinal operations `concat_l_ord` and `concat_r_ord` defined in Section 5.2.2 to correctly index into `A1` and `A2`. For instance, we consider two families `A1` = (1, 3) in `'nat^2` and `A2` = (5, 7, 9) in `'nat^3`. The `concatF` function operates as follows:

$$\texttt{concatF A1 A2} = \texttt{fun i : 'I}_{2+3} \Rightarrow \begin{cases} \texttt{A1 i} & \text{if } \texttt{i} < 2, \\ \texttt{A2 (i} - 2\texttt{)} & \text{else.} \end{cases}$$

Thus, the new concatenated family will be $(1, 3, 5, 7, 9)$.

Last but not least, when involving ordered data, we construct a sortedness predicate of the elements of a finite family.

```
Definition sortedF (op : E → E → Prop) {n} (A : 'E^n) : Prop := ∀(i j : 'I_n),
  i < j → op (A i) (A j).
```

This predicate is used to determine whether a family of elements `A` is sorted with respect to a given ordering operation `op`.

Concluding this section, we consider cases where the elements of a family are functions. Each item in the family can itself be a function, which is expressed as `f : '(E → F)^n`, meaning the family consists of `n` functions from `E` to `F`. We define a function that, for each index `i` within the set `'I_n`, applies the `i`-th function from the family of function `f` to the `i`-th element of the family `A`, resulting in a new element of type `F`. The Coq definition is given by:

```
Definition mapiF {n} (f : '(E → F)^n) (A : 'E^n) : 'F^n := fun i : 'I_n ⇒ f i (A i).
```

Building on this definition, the `mapF` function represents an `n`-family made of the images by `f` of items of `A`. The Coq definition represents this function as follows:

```
Definition mapF {n} (f : E → F) (A : 'E^n) : 'F^n := mapiF (fun⇒ f) A.
```

here, `(fun ⇒ f)` creates an anonymous function, that is equivalent to writing `(fun _ ⇒ f)`, which ignores its arguments and simply returns `f`.

Extending the concept of mapping to the binary function `f`, the `map2F` applies the function `f` to corresponding elements from two `n`-families `A` and `B`.

```
Definition map2F {n} (f : E → F → G) (A : 'E^n) (B : 'F^n) : 'G^n :=
  fun i : 'I_n ⇒ f (A i) (B i).
```

5.3 Algebraic Structures

Section 5.3.1 focuses on abstract monoids, exploring their structure and the operation of finite iterations. Section 5.3.2 discusses the structure of multiplicative monoids, including the study of finite product operations within this structure. Section 5.3.3 introduces the structure of module spaces [41, Section 6.1 p.163], beginning with the definition and properties of additive group and submodule spaces. Section 5.3.4 studies linear combinations within module spaces. Section 5.3.5 introduces the Kronecker delta function and outlines its properties. Finally, Section 5.3.6 concludes by extending the discussion to affine spaces, and introducing the concept of barycenters.

5.3.1 Abstract Monoid and Finite Iterations of the Law

A monoid G is defined as a set equipped with an operation that is associative and has an identity element, typically denoted as zero when the operation is denoted `plus`. For a structure to be classified as a monoid, it must satisfy the following axioms:

1. Closure: $\forall a, b \in G, \quad \text{op } a\ b \in G$,
2. Associativity: $\forall a, b, c \in G,\ (\text{op } (\text{op } a\ b)\ c) = \ \text{op } (a \text{ op } (b\ c))$.
3. Identity Element: $\forall a \in G, \quad \text{op } e\ a = \ \text{op } a\ e = a$.

An abelian monoid builds upon the concept of a monoid by introducing the requirement for the operation to be commutative. This means that for all elements $a, b \in G$, the equation $\text{op } a\ b = \ \text{op } b\ a$ must hold. In Coquelicot, this algebraic structure is denoted as `AbelianMonoid`, and it is the entry point in the hierarchy of algebraic structures.

In Coq, we can define the summation over elements of an abelian monoid G using the following:

```
Definition sum {n} (u : 'G^n) : G := \big[plus_cm/0]_(i < n) u i.
```

This definition employs the big operator from the math-comp library [7]. Here, `plus_cm` is an abstract operation carrying the proof that the abstract operation `plus` of Coquelicot is indeed a commutative monoid law in the world of math-comp, which is used to compute the sum of families indexed from 0 to $n-1$. The initial value of 0 in the summation indicates that we start with the identity element `zero` of the monoid, to ensure the operation is well-defined even when the set being summed is empty. We use the symbole + as the notation for the abstract operation `plus`.

The properties of `plus_cm`, such as associativity and commutativity, ensure that the order of addition does not affect the outcome of the sum. Furthermore, if $n = 0$, the summation of zero elements results in the identity element `zero`, demonstrating that the sum is well-defined even for an empty index set. This is outlined in Coq as:

```
Lemma sum_nil : ∀(u : 'G^0), sum u = 0.
```

We recall that `'G^0` denotes the type of functions from the empty set `'I_0` to `G`. Thus, `'G^0` is a type with a single element, called unit type. Moreover, it is also an abelian monoid, as a functional type towards the abelian monoid `G`.

We also establish that if all elements of a family u are zero (i.e., u is the identity element of the monoid `'G^n`), the resultant sum is zero.

```
Lemma sum_zero_compat : ∀{n} (u : 'G^n), u = 0 → sum u = 0.
```

We observe that the first occurrence of 0 is of type `'G^n`, and the second 0 is of type `G`.

Summation can be understood recursively: the sum of a non-empty family u decomposes into the sum of its first element u_0 and the sum of the remaining elements of the family.

```
Lemma sum_ind_l : ∀{n} (u : 'G^n.+1), sum u = u ord0 + sum (liftF_S u).
```

where, `liftF_S` is the shift function of finite families defined in Section 5.2.3.

In scenarios involving families, where most elements are zeros except for a possibly non-zero entry, the `itemF` function constructs such families using the replacement function `replaceF` established in Section 5.2.3.

```
Definition itemF n (x : G) (i0 : 'I_n) : 'G^n := replaceF 0 x i0.
```

This function initializes all elements to the identity element of the monoid (zero), except for the element at index `i0`, which is set to `x`. The following lemma states that the sum of this family equals the element `x`.

```
Lemma sum_itemF : ∀{n} (x : G) (i0 : 'I_n), sum (itemF n x i0) = x.
```

Expanding upon the concept of family concatenation, the summation over concatenated families confirms that the sum of a concatenated family equals the combined sums of the original families.

```
Lemma sum_concatF : ∀{n1 n2} (u1 : 'G^n1) (u2 : 'G^n2),
    sum (concatF u1 u2) = sum u1 + sum u2.
```

where, `concatF` is the concatenation function of finite families defined in Section 5.2.3.

Furthermore, we provide a Coq definition of the concept of concatenating a family of families. Here is a detailed explanation of this definition. We start by introducing the function `concatnF_aux`, which uses lists instead of families. The list of lists is concatenated into a single list:

```
Definition concatnF_aux {n} {b : 'nat^n} (u : ∀i : 'I_n, 'G^(b i)) : list G :=
  List.concat (to_listF (fun i ⇒ to_listF (u i))).
```

In this definition, `n` indicates the number of families, `b` is the family of natural numbers indicating the size of each family `u`, and `u i` is a family of type `G` with length `b i` for each index `i`. To explain how this function works, we use the list `concat` function which flattens a list of lists into a single continuous list, by appending each head of the list to the recursively concatenated tail. Here `to_listF` converts a family into a list.

We validate that the total length of this concatenated list `concatnF_aux` is equal to the sum of the lengths of the individual families `b i`.

```
Lemma concatnF_aux_length : ∀{n} {b : 'nat^n} (u : ∀i: 'I_n, 'G^(b i)),
  length (concatnF_aux u) = sum b.
```

This leads to the definition of the `concatnF` function, which casts the concatenated list `concatnF_aux` back into a single family of size `sum b`.

```
Definition concatnF {n} {b : 'nat^n} (u : ∀i: 'I_n, 'G^(b i)) : 'G^(sum b) :=
  castF (concatnF_aux_length u) (of_listF (concatnF_aux u)).
```

where, `castF` is the casting function of types of finite families defined in Section 5.2.3, and `of_listF` is a function that converts a list into a family whose length is the length of the list.

To better understand the process of concatenating a family of families, let us consider a practical example where we have three families: $u_0 = (a, b, c) \in$ 'G^3, $u_1 = (d, e) \in$ 'G^2, and $u_2 = (f, g, h, i) \in$ 'G^4. The family $b = (3, 2, 4)$ represents the lengths of these families. Initially, each family u_i is transformed into a list using the function `to_listF`. This results in the separate lists $[a, b, c]$, $[d, e]$, and $[f, g, h, i]$. These lists are then compiled into a single list of lists: $[[a, b, c], [d, e], [f, g, h, i]]$. Following this, the concatenation function processes the list of lists, combining them into a single, flattened list: $[a, b, c, d, e, f, g, h, i]$. Notably, the total length of this list is 9, which precisely corresponds to the summed lengths in b. Finally, the function `concatnF` takes this unified list and casts it back into a family of type 'G^9.

Given that the `concatnF` function merges multiple families, defined by `u`, into a unified family, the value at any particular index `k` can be retrieved using the lemma:

```
Lemma concatnF_splitn_ord : ∀{n : nat} {b : 'nat^n} (u : ∀i : 'I_n, 'G^(b i)) (k : 'I_(sum b)),
    concatnF u k = u (splitn_ord1 k) (splitn_ord2 k).
```

Where `splitn_ord1 k` and `splitn_ord2 k` respectively output `i : 'I_n` and `j : 'I_(b i)` such that `concatnF u k = u i j` refers to the k-th element in the concatenated family of `u`. Note that the type of `k` carries the sizes `b`.

For illustration, consider the previous example with three blocks defined by $b = (3, 2, 4)$, and the concatenated function sequence given by $u = (u_0, u_1, u_2)$. If `k` $= 3$ (representing the 4-th element in the concatenated family using 0-based indexing), this falls within the second block (block index `i` $= 1$). Consequently, `splitn_ord1 k` evaluates to 1. Within this block, `k` $= 3$ represents the first position. Thus, `splitn_ord2 k` evaluates to 0, pinpointing the exact location in the sequence as `concatnF u` $3 =$ `u` $1\ 0$.

5.3.2 Multiplicative Monoid and Monomials

Finite Product in a Multiplicative Monoid

The real number type, $\mathbb{R}$, is already equipped with an additive monoid structure. We introduce the alias `R_mul` $:=$ `R` with coercions to assign it an additional multiplicative monoid structure, whereby the operation `sum` is interpreted as a product where 1 is an identity element of this monoid. In this section, we will outline the definition of the big product operation, and its inherent properties.

The big product operation, denoted in Coq by `prod_R`, aggregates elements through multiplication. Mathematically, this operation can be expressed as:

$$\prod_{i=0}^{n-1} u_i = u_0 \cdot u_1 \cdot \ldots \cdot u_{n-1}.$$

Here, u_i represents the elements of a family $\boldsymbol{u}$ belonging to $\mathbb{R}^n$.

A special case of the product operation indicates that a family consisting solely of the identity element ($(1, \ldots, 1)$ in this case) results in one, irrespective of the size of the family $\boldsymbol{u}$.

```
Lemma prod_R_one_compat : ∀{n} (u : 'R_mul^n), u = 1 → prod_R u = 1.
```

The first occurrence of 1 is of type `'R_mul^n`, while the second occurrence of 1 is of type `R_mul`.

Lastly, we state that if any element of the family u is zero, the entire product evaluates to zero, showcasing the absorbing element property of zero in multiplication. This is captured as,

```
Lemma prod_R_zero : ∀{n} (u : 'R_mul^n), (∃ i : 'I_n, 0 = u i)→ prod_R u = 0.
```

Monomials

A multi-variate monomial is defined as a product of powers of variables, with each variable possibly raised to a non-negative integer exponent. It has the general form:

$$\prod_{i=0}^{n-1} u_i^{\alpha_i} = u_0^{\alpha_0} \times u_1^{\alpha_1} \times \ldots \times u_{n-1}^{\alpha_{n-1}}. \tag{5.1}$$

In the expression, $(u_0, u_1, \ldots, u_{n-1}) \in \mathbb{R}^n$ represents a family of variables in $\mathbb{R}$. The family $(\alpha_0, \alpha_1, \ldots, \alpha_{n-1}) \in \mathbb{N}^n$ corresponds to their respective non-negative integer exponents. For

instance, if a monomial is defined in three variables x, y, and z, and $\boldsymbol{\alpha} = (2, 1, 3)$, the monomial can be expressed as x^2yz^3.

We first define the family $(u_i^{\alpha_i})_{i\in[0..n-1]} \in \mathbb{R}^n$ through the `map2F` function defined in Section 5.2.3. In Coq, this operation is implemented as follows using the `pow` power function from the real numbers library:

```
Definition powF {n} (u : 'R^n) (a : 'nat^n) : 'R^n := map2F pow u a.
```

Then we compute the product of the resulting elements in Equation (5.1). The formal definition in Coq is presented as follows:

```
Definition powF_P {n} (a : 'nat^n) (u : 'R^n) : R := prod_R (powF u a).
```

This function effectively creates a monomial where each term of the product is a component of the family $\boldsymbol{u}$, each raised to the corresponding exponent in the family $\boldsymbol{a}$.

5.3.3 Group and Module Space

A *group* $(G, +)$ is a set equipped with a monoid structure and an inverse operation that adheres to the following axiom:

- Inverse: $\forall x \in G,\ x + (-x) = (-x) + x = 0$.

A group is further classified as an abelian (or commutative) group if it satisfies the commutative property, meaning that for all x and y in the group, $x + y = y + x$. In Coquelicot, this structure is denoted as `AbelianGroup` (see Figure 2.1).

Given an abelian group $(E, +)$, we can define G as a subgroup of E using the following Coq definition:

```
Definition compatible_g {E : AbelianGroup} (G : E → Prop) : Prop :=
  (∀ (x y : E), G x → G y → G (plus x (opp y))) ∧ (∃ (x : E), G x).
```

This definition specifies that G is closed under substraction, and is not the empty set. Namely, it is compatible with the group structure.

A *module space* $(E, +, \cdot)$ over a commutative ring K, is a generalization of vector spaces, denoted in Coquelicot as `ModuleSpace`. Moreover, a ring K is an algebraic structure consisting of a set equipped with two operations: addition $+$, which forms an abelian group with an additive identity 0, and multiplication $\cdot$, which is associative and distributes over addition.

A module space E is a mathematical structure consisting of a set equipped with two operations: vector addition $+$ and scalar multiplication $\cdot$. The elements of E are referred to as vectors, and those of K are known as scalars. These operations must adhere to the following axioms to establish a module space:

1. $(E, +)$ is an abelian group.
2. Distributivity of Vectors: $\forall \nu \in K, \forall (x, y) \in E^2,\ \nu.(x + y) = \nu.x + \nu.y$,
3. Distributivity of Scalars: $\forall (\nu, \mu) \in K^2, \forall x \in E,\ (\nu + \mu).x = \nu.x + \mu.x$,
4. Associativity of Scalars: $\forall (\nu, \mu) \in K^2, \forall x \in E,\ \nu.(\mu.x) = (\nu\mu).x$,

5. Identity Element: $\forall x \in E,\ 1_K.x = x$, where 1_K denotes the multiplicative identity in K.

In this context, the dot operation $\cdot$ is of type: `K → E → E` in Coq.

Specifically, since a field is an abelian ring in which every non-zero element is invertible, module spaces extend the concept of vector spaces by expanding the set of scalars from a field to a ring.

A *submodule space* of a module space $(E, +, \cdot)$ over the ring K is any subset F of E that is an additive subgroup of E and satisfies scalar multiplication closure, such that for every scalar $\mu \in K$ and every vector $x \in F$, the product μx remains in F ([41, Section 6.2 p.165]). The formal definition in Coq is presented as follows:

```
Definition compatible_ms {K : Ring} {E : ModuleSpace K} (F : E → Prop) :=
  compatible_g F ∧ (∀ (x : E) (a : K), F x → F (scal a x)).
```

The space F qualifies as a module space over the ring K, satisfying the essential axioms required for a module space where K is a ring. It is noteworthy that both the zero vector $\{0\}$ and the entire set E naturally form submodule spaces of E.

A series of lemmas further elucidates the basic properties of submodule spaces:

```
Lemma compatible_ms_zero: ∀F : E → Prop, compatible_ms F → F zero.
Lemma compatible_ms_plus: ∀(F : E → Prop) (x y : E), compatible_ms F →
  F x → F y → F (plus x y).
```

Linearity forms the core principle of linear algebra, combining the two operation laws that define module spaces. This concept is required not only for defining linear applications but also will be useful in subsequent sections to construct, for instance, linear combinations within module spaces.

Consider two module spaces, E and F, over a ring K. A function f from E to F is termed a linear application if it satisfies linear properties formally defined in Coq as follows:

```
Definition lin_map {K : Ring} {E F : ModuleSpace K} (f: E → F) : Prop :=
   (∀ (x y : E), f (plus x y) = plus (f x) (f y))
 ∧ (∀ (x : E) (a : K), f (scal a x) = scal a (f x)).
```

Note that the first occurrence of `plus` refers to the addition operation in E, while the second occurrence pertains to the addition operation within F.

From this definition, we can deduce that:

$$f : E \to F \text{ is linear } \iff \forall (x, y) \in E^2, \forall (\nu, \mu) \in K^2, \quad f(\nu x + \mu y) = \nu f(x) + \mu f(y).$$

If f is linear and bijective, it is called an *isomorphism*.

Let E and F be two module spaces over a ring K. The set of linear applications from E to F forms a submodule space of the space of all applications from E to F. We denote this set by $\mathcal{L}(E, F)$.

```
Lemma cms_lm: ∀E F : ModuleSpace R_Ring, compatible_ms lin_map.
```

We note that the space of all applications from E to F is a module space itself, as it aligns with the `fct_ModuleSpace` function, which establishes a module space over functions from any type `E` to any module space `F` over the same ring `K`.

5.3.4 Linear Combination in a Module Space

Within a module space E over a ring K, consider a finite family $(B_i)_{i\in[0..n-1]}$ and scalars $(L_i)_{i\in[0..n-1]}$. The linear combination of these families is defined mathematically as:

$$\sum_{i=0}^{n-1} L_i B_i.$$

Here, L_i represents the coefficients in the ring K.

To formalize this definition in Coq, we start by constructing the scalar multiplication across elements of two sequences using the `map2F` function, defined in Section 5.2.3. Let a scalar L from K^n and a family B from E^n:

```
Definition scalF {n} (L : 'K^n) (B : 'E^n) : 'E^n := map2F scal L B.
```

Then, summing the result of this function to produce the linear combination in E:

```
Definition lin_comb {n} (L : 'K^n) (B : 'E^n) : E := sum (scalF L B).
```

In this definition, the finite summation `sum` is established in Section 5.3.1.

One important lemma of linear combinations is that any linear combination where the coefficients equal zero results in the zero family.

```
Lemma lc_zero_compat_l : ∀{n} (L : 'K^n) (B : 'E^n), L = 0 → lin_comb L B = 0.
```

For higher-dimensional spaces of dimension $n+1$, a linear combination can be recursively broken down into a scalar multiplication of their first elements, plus the linear combination of the rest of their elements (after "lifting"). This is established through this lemma:

```
Lemma lc_ind_l : ∀{n} (L : 'K^n.+1) (B : 'E^n.+1),
  lin_comb L B = scal (L ord0) (B ord0) + lin_comb (liftF_S L) (liftF_S B).
```

We recall that `ord0` refers to the first index in a non-empty family (refer to Section 2.3), and the `liftF_S` function as referenced in Section 5.2.3 shifts the indices of each family by one, effectively "lifting" the sequence to exclude the first element and then proceeding with the next part of the family. This recursive approach allows to perform proofs involving linear combination by induction on the number of vectors. This lemma is a consequence of the `sum_ind_l` lemma.

Moving to the linearity aspect of mappings, we have for any given index `i` within `'I_n`, the function that takes an element `B` from the space `'E^n` and returns the `i`-th component of `B` is a linear mapping.

```
Lemma lm_component : ∀{n} (i : 'I_n), lin_map (fun B : 'E^n ⇒ B i).
```

Here, `lin_map` refers to the property of being a linear mapping (see Section 5.3.3), which means that the i-th component B_i must satisfy the two assertions of linearity: preservation of addition, and scalar multiplication.

Furthermore, considering we have two module spaces E and F over a ring K. We describe a more complex scenario involving a family of scalar valued linear maps on K and their interaction with linear combinations. Assuming that each function f_i in the family is a linear map, we conclude that the linear combination of the maps $x \longmapsto \sum_{i=0}^{n-1} f_i(x)B_i$ where B_i are elements of F, remains a linear map.

```
Lemma fct_lc_l_lm : ∀{n} (f : '( E → K)^n) (B : 'F^n), (∀ j : 'I_n, lin_map (f j)) →
  lin_map (fun x : E ⇒ lin_comb (fun i : 'I_n ⇒ f i x) B).
```

5.3.5 Kronecker Delta Function

In the context of real numbers $\mathbb{R}$, the Kronecker delta function is a widely used tool in mathematical expressions involving for example vectors. It is defined precisely for all $i, j \in \mathbb{N}$ by the following:

$$\delta_{i,j} \stackrel{\text{def.}}{=} \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases} \tag{5.2}$$

The function is implemented in Coq using the decidability lemma `Nat.eq_dec` to handle decision about equality in `nat`, as shown in the following pattern matching:

```
Definition kronecker (i j : nat) : R :=
  match (Nat.eq_dec i j) with
    | left _  ⇒ one
    | right _ ⇒ zero
    end.
```

The Kronecker delta function is symmetric confirmed within Coq as:

```
Lemma kronecker_sym : ∀(i j : nat), kronecker i j = kronecker j i.
```

By Equation (5.2), the Kronecker delta function equals 1 when its indices match $i = j$.

```
Lemma kronecker_is_1 : ∀(i j : nat), i = j → kronecker i j = 1.
```

Conversely, the Kronecker delta function equals 0 when its indices do not match $i \neq j$.

```
Lemma kronecker_is_0 : ∀(i j : nat), i ≠ j → kronecker i j = 0.
```

A key utility of the Kronecker delta function is observed when it is summed over one of its indices. In such cases, the sum across any row or column of a matrix representation equates to one. The corresponding lemmas in Coq are:

```
Lemma sum_kronecker_l : ∀{n} (j : 'I_n), sum (fun i : 'I_n ⇒ kronecker i j) = 1.
Lemma sum_kronecker_r : ∀{n} (i : 'I_n), sum (fun j : 'I_n ⇒ kronecker i j) = 1.
```

Additionally, we recall from Section 5.3.1 that `itemF` is a function that creates a family in which all entries are initialized to a zero element of an abelian monoid $\mathbb{R}$, except for one specified index where the value is the number $x \in \mathbb{R}$. In this context, we state that for any dimension d, the family produced by the `itemF` function at index i with value x is equal to the family where each element is the product of x and the Kronecker delta evaluated at i and j.

```
Lemma itemF_kronecker_eq : ∀{d} (x : R) (i : 'I_d), itemF d x i = (fun j ⇒ x * kronecker i j).
```

One important thing to note is that decrementing both indices of the Kronecker delta function does not change their equality status as long as both indices are non-zero.

```
Lemma kronecker_pred_eq : ∀i j, i ≠ 0 → j ≠ 0 → kronecker (i − 1) (j − 1) = kronecker i j.
```

5.3.6 Affine Spaces and Barycenter

Affine spaces are a fundamental concept in geometry and algebra that extend the ideas of module spaces (see Section 5.3.3). While a module space focuses on vectors that abstract the concepts of direction and magnitude, an affine space emphasizes points and the transformations between them. Any module space may be equipped with an affine space structure by choosing

some origin that may differ from the null vector. For instance, in $\mathbb{R}^3$, every plane is an affine subspace whether it contains $(0,0,0)$ or not.

An affine space is denoted in Coq as `AffineSpace`. Mathematically, if `V` is a module space over a ring `K`, then an associated affine space `E` consists of a non-empty set of points that is equipped with a function `vect : E → E → V`, denoted by $\overrightarrow{AB} :=$ `vect A B` for all `A` and `B` in `E`. This function satisfies the following conditions:

1. Chasles relation: $\forall A, B, C \in E, \overrightarrow{AB} + \overrightarrow{BC} = \overrightarrow{AC}$.
2. Bijectivity: $\forall (A : E)(\overrightarrow{u} : V), \exists! B : E, \overrightarrow{AB} = \overrightarrow{u}$.

Another key operation is the construction of affine combinations of points in `E`, which is an extension of linear combinations in module spaces (refer to Section 5.3.4). It is defined through the following predicate:

$$\sum_{i=0}^{n-1} L_i \, \overrightarrow{GA_i} = \overrightarrow{0}. \tag{5.3}$$

When $\sum_{i=0}^{n-1} L_i$ is invertible in the ring `K` (i.e., the sum of the weights has an inverse in the ring `K`), `G` is called the affine combination of points $(A_i)_{i\in[0..n-1]}$ with coefficients $(L_i)_{i\in[0..n-1]}$. This is translated in Coq as:

```
Context {K : Ring}.
Context {V : ModuleSpace K}.
Context {E : AffineSpace V}.
Definition aff_comb {n} (L : 'K^n) (A : 'E^n) (G : E) : Prop := lin_comb L (vectF G A) = 0.
```

Here, `vectF G` is a function that converts a family of points in `E` into a family of vectors in `V` starting from `G`.

The Equation (5.3) implies the following equation using the Chasles relation with any point O:

$$\left(\sum_{i=0}^{n-1} L_i\right) \overrightarrow{OG} = \sum_{i=0}^{n-1} L_i \, \overrightarrow{OA_i}. \tag{5.4}$$

When $\sum_{i=0}^{n-1} L_i$ is invertible, G is also called barycenter of points $(A_i)_{i\in[0..n-1]}$ with weights $\{L_i\}_{i\in[0..n-1]}$. Moreover, when the sum is one and the points $(A_i)_{i\in[0..n-1]}$ are affinely independent (see Section 5.4.3), the weights are called the barycentric coordinates of G in the affine system $(A_i)_{i\in[0..n-1]}$.

We guarantee the existence of a barycenter `G` as an affine combination of points in `A` using weights `L` through this lemma,

```
Lemma baryc_EX : ∀{n} {L : 'K^n} (A : 'E^n), invertible (sum L) → { G | aff_comb L A G }.
```

The precondition `invertible (sum L)` specifies that the sum of the weights has an inverse in the ring `K`, and the Equation (5.4) may be simplified. If `K` is a field, this means that $\sum_{i=0}^{n-1} L_i \neq 0$.

The barycenter `G` is then defined as a total function based on whether the sum of the weights `L` is invertible or not as follows:

```
Definition barycenter {n} {L : 'K^n} (A : 'E^n) : E := match invertible_dec (sum L) with
  | left HL ⇒ proj1_sig (baryc_EX A HL)
  | right _ ⇒ point_of_as E
  end.
```

When the sum of the weights is invertible, the function is the affine combination of the points `A` with weights `L` extracted from the lemma `baryc_EX`. When the sum is not invertible, the function returns a default point from the affine space `E` (the witness of non-emptiness of `E`).

Now, we determine a special kind of barycenter, named *isobarycenter*, which corresponds to equal weights, e.g. 1.

```
Definition isobarycenter {n} (A : 'E^n) : E := barycenter ones A.
```

In the sequel, we consider `V1` and `V2` as module spaces over a ring `K`, and associated affine spaces `E1` and `E2`. A function `f : E1 → E2` preserving barycenters of any family of points with any weights of invertible sum is called an *affine mapping*. It is defined as follows:

```
Definition aff_map (f : E1 → E2) : Prop := ∀n (L : 'K^n) (A1 : 'E1^n),
    invertible (sum L) → f (barycenter L A1) = barycenter L (mapF f A1).
```

A significant property of affine mappings is their behavior under the bijective property. A function is bijective if it is both injective and surjective, as established in Section 5.1.3. If `f : E1 → E2` is an affine and bijective map, then its inverse is also an affine map, stated by:

```
Lemma am_bij_compat : ∀{f : E1 → E2} (Hf : bijective f), aff_map f → aff_map (f_inv Hf).
```

Furthermore, let us assume `E1` and `E2` are two module spaces over a ring `K`, and each is considered as an affine space over itself. In this setting, the vector from point `A` to point `B` is defined by `vect A B := B − A`. Within this framework, affine mappings can be expressed as the sum of a linear map and a constant vector. This decomposition is formalized as follows:

```
Lemma am_lm_ms : ∀{lf : E1 → E2} (c2 : E2), lin_map lf → aff_map (lf + (fun⇒ c2)).
```

This lemma demonstrates that adding a constant vector to any linear map creates an affine map. Conversely, by subtracting the image of the zero vector in `E1` from the map `f`, one can revert it to a linear map. For more details on the linearity of mappings, refer to Section 5.3.3.

```
Lemma am_lm_0_rev : ∀{f : E1 → E2}, aff_map f → lin_map (f − (fun⇒ f 0)).
```

5.4 Finite Dimensional Subspaces

5.4.1 Linear Span

Consider `E` a module space over a ring `K`, and `B` a family of vectors of `E`. The linear span of `B` is the subset of `E` of all possible linear combinations of `B`. This is formally captured in a Coq environment as an inductive type:

```
Inductive lin_span {n} (B : 'E^n) : E → Prop := Lin_span : ∀L, lin_span B (lin_comb L B).
```

In this definition, `lin_span` is a predicate that specifies which vectors of `E` are in the span of a given family of vectors `B` (refer to Section 5.1.1).

A vector $x \in E$ is in the linear span of `B` if there exists a family of scalars `L` from `K`, such that x can be expressed as the linear combination of `B` with coefficients `L`, i.e.,

$$x \in \text{ span } (B_0, \ldots, B_{n-1}) \stackrel{\text{def.}}{=} \exists L \in K^n,\ x = \sum_{i=0}^{n-1} L_i B_i. \tag{5.5}$$

From this Equation (5.5), since the span of `B` encompasses all possible linear combinations of the vectors within the set, then each individual vector in `B` is included in its linear span when it is multiplied by the scalar coefficient 1, while all other coefficients are set to 0.

```
Lemma lin_span_inclF_diag : ∀{n} (B : 'E^n), inclF B (lin_span B).
```

Here, `inclF` represents the inclusion property of finite families in a subset discussed in Section 5.2.3.

Moreover, when comparing two families of vectors, `B1` and `B2`, with different dimensions, their linear spans are equal if each family is included in the span of the other.

```
Lemma lin_span_ext : ∀{n1 n2} (B1 : 'E^n1) (B2 : 'E^n2),
  inclF B1 (lin_span B2) → inclF B2 (lin_span B1) → lin_span B1 = lin_span B2.
```

The linear span of a family of vectors qualifies as a submodule space in the module space `E` (see Section 5.3.3).

```
Lemma lin_span_cms : compatible_ms (lin_span B).
```

Consequently, the linear span is also closed under addition and scalar multiplication that naturally extends to the closure under all linear combinations.

```
Lemma lin_span_lc_closed : ∀(n : nat) (L : 'K^n) (A B : 'E^n),
  inclF A (lin_span B) → lin_span B (lin_comb L A).
```

5.4.2 Generating, Free, Basis Families

Let `E` be a module space over a ring `K`. After confirming that the linear span of a family of vectors `B` of `E` qualifies as a module space and includes all linear combinations of vectors from `B`, we can define a predicate stating that `B` is a generating family for a subset `PE` of `E`. This relationship is formally captured by the following definition:

```
Definition lin_gen (PE : E → Prop) {n} (B : 'E^n) : Prop := PE = lin_span B.
```

This definition underlines that the subset `PE` is entirely spanned by the vectors $(B_i)_{i\in[0..n-1]}$. It ensures that for B to qualify as a generating family, every element within `PE` must be representable as a linear combination of the vectors in B.

We now address the inclusion relationship between a generating family and the subset it spans. If B forms a generating family for a subset `PE` of `E`, it implies that each vector in B must be an element of `PE`, expressed through the inclusion function `inclF` defined in Section 5.2.3.

```
Lemma lin_gen_inclF : ∀{PE : E → Prop} {n} {B : 'E^n}, lin_gen PE B → inclF B PE.
```

Next, we delve into the concept of *linear independence* of vectors. The vectors $(B_i)_{i\in[0..n-1]}$ are linearly independent (or that the family of vectors B is free [41, Section 6.6 p.188]) if the only zero linear combination is the trivial one.

```
Definition lin_indep {n} (B : 'E^n) := ∀(L : 'K^n), lin_comb L B = 0 → L = 0.
```

This guarantees that no vector in B is redundant; none can be represented as a linear combination of others.

Having established B as both a generating family and as linearly independent, we are positioned to declare that B forms a basis for PE. This is specified as follows:

```
Definition basis (PE : E → Prop) {n} (B : 'E^n) := lin_gen PE B ∧ lin_indep B.
```

Ultimately, we tie together the basis with the notion of dimension. The dimension of a submodule space PE of the module space E is defined as the number of vectors in its bases. This is formulated as follows:

```
Inductive has_dim (PE : E → Prop) n : Prop :=
  Dim : ∀ (B : 'E^n), basis PE B → has_dim PE n.
```

This dimension describes the minimum number of vectors required to span the entire subset PE, and the maximum number of vectors that may be linearly independent. We prove that all bases of PE have the same dimension, thereby ensuring the correctness of this definition.

We articulate that in a module space E, if the vectors $(B_i)_{i\in[0..n-1]}$ are linearly independent, then they also form a basis for its linear span, and vice versa.

```
Lemma basis_lin_span_equiv : ∀{n} {B : 'E^n}, basis (lin_span B) B ↔ lin_indep B.
```

To establish a family as a basis in a submodule space, it must satisfy the condition of having the correct size (matching the known dimension n), and must be linearly independent.

```
Lemma lin_indep_basis : ∀{PE : E → Prop} {n} {B : 'E^n},
  has_dim PE n → inclF B PE → lin_indep B → basis PE B.
```

We now present a specific lemma that explores the relationship between bijections and injections within the framework of a module space E over the ring of real numbers $\mathbb{R}$. This lemma is very important for subsequent discussions, particularly in establishing the unisolvence property of the Lagrange finite element. The details of its application are thoroughly discussed in Section 10.3.7.

Within the module space E, we consider a subset PE of the type of functions from E to $\mathbb{R}$. We assume PE to be finite-dimensional of dimension nPE, and acknowledge its structure as a submodule space.

Consider a family of vectors A of type 'E^n, and define a family of linear functions $(u(A_i))_{i\in[0..n-1]} \in \mathbb{R}^n$. Then, if the number of vectors in A matches the dimension nPE of PE where PE is a subset of the function space $\mathcal{F}(E, \mathbb{R})$, then there exists a bijection between PE and the full set $\mathbb{R}^n$ if and only if the function u is injective on the module space corresponding to PE. The formal lemma is expressed as follows:

```
Lemma lmS_bijS_val_gather_equiv : ∀{E : ModuleSpace R_Ring} {PE : (E → R) → Prop}
  {nPE : nat} (HPE : compatible_ms PE) {n : nat} (A : 'E^n),
   n = nPE → bijS PE fullset (fun (u : E → R) (i : 'I_n) ⇒ u (A i)) ↔
    (∀ u : sub_ModuleSpace HPE, (∀ i : 'I_n, val u (A i) = 0) → u = 0).
```

This lemma has slightly been simplified from its original version in Coq to enhance clarity and readability. The term bijS signifies bijectivity within subsets, which is explored in detail in Section 5.1.4. The sub_ModuleSpace declaration is associated with the construction of the module space PE, based on the proof HPE that it is closed under linear operations. Meanwhile, val serves as the canonical injection from sub_ModuleSpace HPE into E.

5.4.3 Affine independence

Building on the principle of linear independence discussed in Section 5.4.2, we introduce the notion of *affine independence* within the context of affine spaces. Let E be a module space, considered as an affine space over itself (see Section 5.3.6). A family of points $(A_i)_{i\in[0..n]}$ is considered affine independent if the family of vectors $(A_i - A_0)_{i\in[1..n]}$ is linearly independent. The formal definition in Coq is presented as follows:

```
Definition affine_independent {n} (A : 'E^n.+1) := lin_indep (liftF_S A - constF n (A ord0)).
```

In simpler terms, no point in the set can be expressed as an affine combination of the others, where an affine combination involves coefficients whose sum must be invertible. Further details on affine combinations can be found in Section 5.3.6. This definition uses the operations `liftF_S` and `constF` of finite families, as described in Section 5.2.3. The application `liftF_S A` shifts each element of `A` by one position, while the `constF` function creates a constant vector where every element is the first element of `A`.

An important property of an affine independent family is highlighted by the lemma `affine_independent_lc` stating that for two families of scalars, if they sum to the same value and produce the same affine combination of points, then they must be identical.

```
Lemma affine_independent_lc : ∀{n} (L1 L2 : 'R^n.+1) (A : 'E^n.+1),
  affine_independent A → sum L1 = sum L2 → lin_comb L1 A = lin_comb L2 A → L1 = L2.
```

Additionally, the concept of an *affine generator* is introduced, which describes a set of points that represent any point in an affine space through an affine combination where the coefficients sum to one. This sum condition ensures that the affine combination remains within the same affine space as the points $(A_i)_{i\in[0..n-1]}$. Consider a module space `E` over the ring $\mathbb{R}$, we have:

```
Definition affine_generator {n} (A : 'E^n) : Prop := ∀(x : E), ∃ (L :  'R^n),
  sum L = 1 ∧ x = lin_comb L A.
```

Finally, a relationship between affine independence and affine generators is established. If a family of $n+1$ points A is affine independent in a space of dimension n, it also qualifies as an affine generator for that space.

```
Lemma affine_independent_generator : ∀{n} (A : 'E^n.+1),
   has_dim fullset n → affine_independent A → affine_generator A.
```

5.4.4 Dual Space, Duality

In this section, we introduce the concept of the dual space of a module space E over the ring $\mathbb{R}$, as detailed in Gostiaux's work [41, Section 6.8 p.210]. The dual space, denoted $\mathcal{L}(E, \mathbb{R})$ or alternatively E', comprises all linear forms mapping elements from E to $\mathbb{R}$. Notably, E' itself constitutes a module space over $\mathbb{R}$, equipped with operations that conform to the linear structure of module spaces.

To concretely understand the structure of E', let E be a finite dimensional module space and $B = (e_i)_{i\in[0..n-1]}$ a basis of E. The concept of a dual basis is introduced for E'. A *dual basis* is a family of linear forms $B' = (e'_i)_{i\in[0..n-1]}$ such that each e'_i uniquely corresponds to e_i from the original basis B. The action of these dual basis vectors on E is defined by the relation:

$$e'_i(e_j) = \delta_{i,j} \quad \text{(Kronecker delta)} \tag{5.6}$$

where δ is the Kronecker delta function given in Section 5.3.5. This definition ensures that each linear form e'_i in the dual basis B' maps its corresponding basis vector e_i in E to 1, and

all other basis vectors to 0.

The family of linear forms B' is both free and spans the entire dual space E', hence it forms a basis for E'. This is possible because in finite dimensions, the dual space E' has the same dimension as E. And so, in this case, E' is isomorphic to E.

Having briefly discussed the concept of dual basis of a basis in E, we now turn our attention to the corresponding reverse concept: the *predual basis* of a basis in E'. Specifically, for any given basis B' of the dual space E', there exists a unique basis B in the finite dimensional module space E such that B' is the dual basis of B. This relationship is mathematically characterized by the condition:

$$e'_i(e_j) = \delta_{i,j}.$$

In Coq, the concept of a predual basis is formulated for a finite dimensional subspace `PE` of the module space `E` over the ring $\mathbb{R}$. This construction requires that the mapping from `PE` (a subset of `E`) to $\mathbb{R}^n$ defined by `B'` is a bijection. This relationship is given by the following definition:

```
Definition predual_basis : ∀{E : ModuleSpace R_Ring} {PE : E → Prop} {n : nat}
  {B' : '( E → R)^n}, bijS PE fullset (gather B') → 'E^n.
```

here, the function `gather` is used here to transform the family `B'` of linear forms of type `'( E → R)^n`, into a single function that maps any input from `E` to a vector in `'R^n`.

The requirement for bijectivity of the dual basis `B'` here, stems from the need to establish a reversible mapping between the space `E` and its dual space `E'`.

Moreover, we specify that for a finite dimensional subspace `PE` of a module space `E` over a ring $\mathbb{R}$, if `PE` is equipped with a basis `B'` of n linear maps from `E` to $\mathbb{R}$, then there exists a unique corresponding basis in `E`, the predual basis. This is formalized as follows:

```
Lemma predual_basis_basis : ∀{E : ModuleSpace R_Ring} {PE : E → Prop} {n : nat},
  has_dim PE n → ∀B' : '( E → R)^n, (∀ i : 'I_n, lin_map (B' i)) →
    ∀ HB' : bijS PE fullset (gather B'), basis PE (predual_basis HB').
```

5.5 Binomials

In this section, we give a brief introduction of binomial coefficients, commonly represented as $\binom{n}{k}$ and denoted in math-comp library[1] by `'C`. The binomial coefficients are the positive integers that provide the number of ways to choose a set of k elements from n elements without regard to the order, where $n, k \in \mathbb{N}$. In this thesis, binomials are needed for constructing various elements required for the triplet of the Lagrange finite element. This includes, for example, the family of multi-indices (refer to Section 8.1) and the dimension of the polynomial approximation space (detailed in Section 8.2).

In this context, we introduce `pbinom`, a Coq function that computes a modified binomial coefficient. Specifically, `pbinom m n` calculates $\binom{m+n}{m} - 1$. This is given in Coq as:

```
Definition pbinom (m n : nat) : nat := ('C ((m + n), m)).-1.
```

The standard binomial coefficient $\binom{m+n}{m}$ can be recovered directly from the successor of the `pbinom` function as it is always positive:

[1] https://math-comp.github.io/htmldoc/mathcomp.ssreflect.binomial.html

```
Lemma pbinomS_eq : ∀m n, (pbinom m n).+1 = 'C (m + n, m).
```

Given that $\binom{m+n}{m}$ is always positive, the expression (`pbinom m n`).+1 structurally reinforces this property. This adjustment enables the application of functions such as `liftF_S` (refer to Section 5.2.3), which require a family size of 'something + 1' to be defined structurally, without imposing further assumptions on size.

We present an interesting property connecting sums of binomial coefficients (see Section 5.3.1). This property is mathematically expressed as:

$$\sum_{i=0}^{n} \binom{i+m}{i} = \binom{n+m+1}{n}.$$

This is translated in Coq as:

```
Lemma pbinomS_rising_sum_r : ∀m n,
  sum (fun i : 'I_n.+1 ⇒ (pbinom i m).+1) = (pbinom n m.+1).+1.
```

Chapter 6

Mathematical Presentation of Finite Elements

The Finite Element Method (FEM)[1] is a widely-used computational technique in engineering and mathematical modeling for approximating solutions to boundary value problems associated with partial differential equations [19, 34, 35, 36, 66]. Its application is particularly useful since deriving analytical solutions is almost always impossible. The principle of the FEM involves dividing a complex domain into smaller, simpler parts called geometrical finite elements (see Figure 6.1). The equations formulated within each of these finite elements are then assembled to form a larger system of equations, representing the entire problem.

This chapter introduces the fundamental principles of the Finite Element Method (see Figure 6.1), using the Galerkin method [66]. These principles are essential for the comprehension of the topics discussed in the subsequent chapters. Although not every aspect covered in this chapter will be formalized using Coq in the present document, it provides detailed steps necessary for the numerical implementation of these principles. A case study illustrating these steps is the Poisson problem (6.1), a second-order linear elliptic partial differential equation within a d-dimensional domain ([48, p.26]). We will demonstrate the construction of a weak discrete formulation for this problem as described in Equation (6.4) in Section 6.2.1 and develop the related linear system through discretization [40]. Additionally, this chapter also defines the geometric mapping transforming the reference geometric element into current geometric elements in Section 6.2.2. Following this, we introduce the general definition of finite elements as a triplet in Section 6.3, (e.g. see [59]). Finally, the principle of unisolvence, necessary for ensuring that the studied problem has a unique solution, is explained in Section 6.4.

We recall that, similar to the previous chapter, certain notations will be presented in boldface. This indicates that they represent families (such as vectors in $\mathbb{R}^n$) containing multiple elements, distinguishing them from scalars in $\mathbb{R}$ or $\mathbb{N}$, for example.

6.1 Continuous Problem: Strong and Weak Formulation

6.1.1 Definitions and Notations

Consider Ω, a non-empty, open, bounded, and connected subset of $\mathbb{R}^d$, with $d \geq 1$ is the spatial dimension and $\Gamma \overset{\text{def.}}{=} \partial\Omega$ representing its regular boundary (i.e., the boundary is smooth,

[1] https://jschoeberl.github.io/iFEM/FEM/finiteelements.html

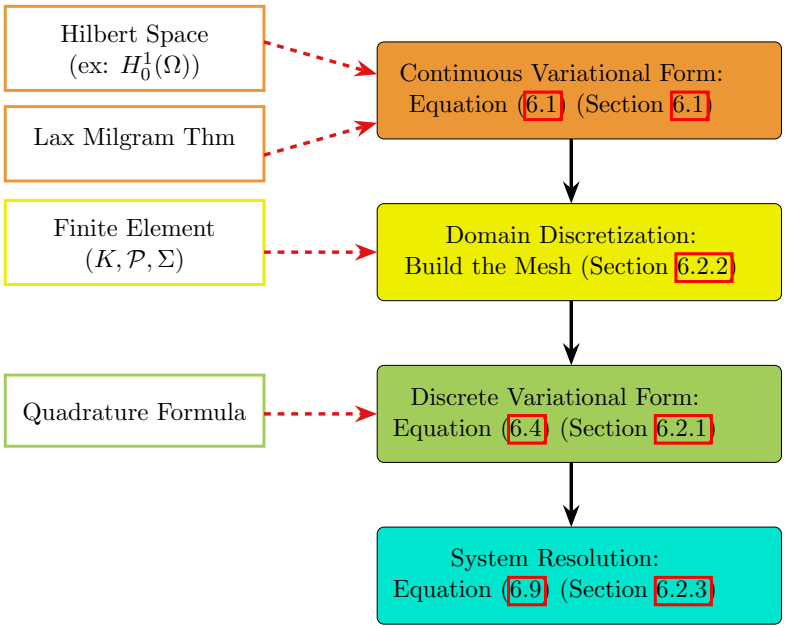

Figure 6.1: A brief overview of the components in a typical stationary FEM. The thick black arrows illustrate the standard steps in solving a problem using FEM, while the dashed red arrows indicate the application of theorems or numerical tools. The filled colors correspond to Sections: 6.1.2 in brown, 6.2.2 in yellow, and 6.2.1 in green, 6.2.3 in turquoise blue.

which implies that it is continuously differentiable to some degree) where certain conditions (like Dirichlet or Neumann conditions [3]) are imposed. To elucidate further:

- An *open* subset is one where, around every point, a small sphere can be drawn such that its entire interior falls within the subset.
- A *bounded* subset means all the points fit within a specific distance from a central point.
- A subset is termed *connected* if there is a continuous path within the subset that links any two points in Ω.

In this section, we define various spaces of functions, ranging from Lebesgue spaces to Sobolev spaces, which will be useful in subsequent sections for a better understanding and application of the finite element method.

Lebesgue Space

The Lebesgue space $L^2(\Omega)$ is a space of functions defined with respect to the measure space $(\Omega, \mathcal{B}, \mu)$, where $\mathcal{B}$ is the Borel sigma-algebra that is generated by the open subsets of the domain Ω, and μ is the Lebesgue measure [18] (refer to Sections 3.1.1 and 3.2). The space $L^2(\Omega)$ includes all measurable functions $f : \Omega \to \mathbb{R}$ that are *square-integrable* over Ω. A function f is square-integrable if the integral of the square of f over Ω is finite, i.e.,

$$\int_\Omega |f(x)|^2 \, d\mu < \infty.$$

This space is a *Hilbert space* (i.e., complete normed vector space with an associated inner product), with the norm defined by $\|f\|_{L^2(\Omega)} = \left(\int_\Omega |f(x)|^2 \, d\mu\right)^{1/2}$. *Completeness* here means that every Cauchy sequence of vectors in the Lebesgue space $L^2(\Omega)$ converges to a limit that is

also in $L^2(\Omega)$.

The construction of $L^2(\Omega)$ heavily relies on the theory of the Lebesgue integral provided in Chapter 3.

Sobolev Space

The *Sobolev space* $H^1(\Omega)$ is a subset of the Lebesgue space $L^2(\Omega)$, designed to include functions that are square integrable and also requires that the first derivatives of these functions, interpreted in the weak sense, are square-integrable (see eg [17, 51, 64, 1]). A *weak derivative* allows us to extend the concept of differentiation to functions that might not be differentiable in the traditional sense. Specifically, a function $u : \Omega \to \mathbb{R}$ has a weak derivative $\mathbf{v} : (\Omega \to \mathbb{R})^d$ denoted by $\mathbf{v} = (v_0, \ldots, v_{d-1})$, with respect to x_i if, for every infinitely differentiable function ϕ with compact support in Ω (i.e., $\phi \in \mathcal{D}(\Omega)$), the following integral equation holds:

$$\forall i \in [0..d-1], \quad \int_\Omega u \, \partial_{x_i}\phi \, d\mu = -\int_\Omega \mathbf{v} \, \phi \, d\mu,$$

where $\partial_{x_i}\phi$ (or $\frac{\partial \phi}{\partial x_i}$) is the partial derivative of ϕ with respect to the i-th coordinate.

The gradient operator, denoted by $\boldsymbol{\nabla} u$, collects the first-order weak derivatives into a family. This operator is used to compute the vector of partial derivatives of a function, representing the vector of weak partial derivatives of u. In mathematical notation, it is expressed as follows:

$$\boldsymbol{\nabla} u = \left(\frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}, \ldots, \frac{\partial u}{\partial x_d} \right) \in \mathbb{R}^d.$$

Additionally, a function has *compact support* means it is nonzero only within a bounded region and zero everywhere else.

In essence, $H^1(\Omega)$ is a Hilbert space that consists of all functions $u \in L^2(\Omega)$ for which the weak derivatives $\boldsymbol{\nabla} u$ also belong to $L^2(\Omega)$. In mathematical terms,

$$H^1(\Omega) = \{u \in L^2(\Omega) : \boldsymbol{\nabla} u \in (L^2(\Omega))^d\},$$

The norm in this space is given by,

$$\|u\|_{H^1(\Omega)} = \left(\|u\|^2_{L^2(\Omega)} + \|\boldsymbol{\nabla} u\|^2_{L^2(\Omega)} \right)^{1/2}.$$

The space $H^1_0(\Omega)$ is a subspace of $H^1(\Omega)$ and includes those functions in $H^1(\Omega)$ that vanish at the boundary Γ. The norm $\|\cdot\|$ on $H^1_0(\Omega)$ is given by:

$$\|u\|_{H^1_0(\Omega)} = \|\boldsymbol{\nabla} u\|_{L^2(\Omega)}.$$

6.1.2 Strong Formulation

Consider a function f in the space $L^2(\Omega)$. The Poisson equation is a boundary-value problem expressed in the strong formulation as follows:

$$\text{find } u : \Omega \to \mathbb{R} \text{ such that:} \quad \begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{on } \Gamma, \end{aligned} \tag{6.1}$$

where the Laplace operator, denoted by Δ, is a second-order differential operator defined as $\Delta u \overset{\text{def.}}{=} \sum_{i=1}^{d} \frac{\partial^2 u}{\partial x_i^2} : \Omega \to \mathbb{R}$, where u is the function to be determined.

The Poisson equation is widely used across various scientific and engineering disciplines, notably in modeling phenomena such as steady-state heat distribution within mediums that have internal heat sources. For instance, the equation $-\Delta u = f$ can represent how temperature is distributed within a material, where f corresponds to heat produced by electronic components.

6.1.3 Weak Formulation

The Finite Element Method (FEM) for solving PDEs is written using a *weak, variational formulation*. For instance, if we look at the Poisson problem in its strong formulation, the solution u is expected to have second derivatives and thus be at least $\mathcal{C}^1(\Omega)$ (i.e., the function u is continuously differentiable over the domain Ω, meaning both u and its first derivatives are continuous across Ω). However, in many cases, there is no solution with such regularity (for domains with geometrical or physical irregularities for example). The correct mathematical setting is the weak, variational formulation. It requires u to be only in $H_0^1(\Omega)$, which means u and its (weak) first derivatives need only be square-integrable, not necessarily continuous. The Lax-Milgram theorem states indeed the existence and the uniqueness of the weak solution in this space.

To transition to the *weak formulation* of the Poisson equation, we begin with multiplying both sides of the strong form of the Poisson Equation (6.1) by a test function $v \in H_0^1(\Omega)$. We then integrate over the domain Ω,

$$-\int_\Omega \Delta u \; v \; d\mu = \int_\Omega f v \; d\mu.$$

Following this, we apply the integration by parts formula [16], to the left-hand side of the equation, and take the boundary condition into account, we obtain the weak form of the Poisson equation,

find $u \in H_0^1(\Omega)$, such that:

$$\forall v \in H_0^1(\Omega), \quad \int_\Omega \boldsymbol{\nabla} u \cdot \boldsymbol{\nabla} v \; d\mu = \int_\Omega f v \; d\mu. \tag{6.2}$$

The *strong formulation* (6.1) is so called because solutions are required to satisfy the partial differential equation at every point within the domain Ω explicitly, and this requires that all the necessary derivatives exist and be continuous as specified by the equation (6.1). On the other hand, the *weak formulation* is termed weak because it weakens these requirements: the solution is required to satisfy the integral of the PDE multiplied by any test functions in a specified space (the integral is often integrated by parts). The PDE in this form is also called a *variational formulation*. There are fewer requirements on the existence and continuity of the derivatives, thus the weak solution lives in a larger space than the strong solution: in many cases, the weak solution exists, whereas the strong solution may not be properly defined.

6.1.4 Algebraic Form and Lax-Milgram Theorem.

Before we proceed with the approximation of the problem described in Equation (6.2) using the Finite Element Method (FEM), it is essential to verify that the problem is well-posed.

This entails confirming that there exists a unique solution. To establish this, we rely on the Lax-Milgram theorem, as referenced in [10, 52].

We generalize the components of Equation (6.2) into a Hilbert space setting. Specifically, we use the space $(V, \|.\|_V)$ to represent $(H_0^1(\Omega), \|.\|_{H_0^1(\Omega)})$, the bilinear form a to represent $(u, v) \longmapsto \int_\Omega \boldsymbol{\nabla} u \cdot \boldsymbol{\nabla} v$, and the linear form ℓ to represent $v \longmapsto \int_\Omega fv$, for all u and v in V. This abstract formulation becomes the algebraic problem:

$$\text{find } u \in V, \text{ such that:} \qquad \forall v \in V, \quad a(u, v) = \ell(v). \tag{6.3}$$

To ensure the applicability of the Lax-Milgram theorem, we need to verify that the bilinear form a meets the following conditions:

(i) boundedness : there exists a constant $c > 0$ such that for all $u, v \in V$,

$$|a(u, v)| \leqslant c\|u\|_V\|v\|_V.$$

(ii) coercivity : there exists a constant $\alpha > 0$ such that for all $u \in V$,

$$a(u, u) \geq \alpha\|u\|_V^2.$$

With these conditions met, the Lax-Milgram theorem guarantees the existence of a unique solution $u \in V$ for the weak formulation of the Poisson equation as specified in Equation (6.2). The requisite conditions for applying this theorem are indeed satisfied in the case of the Equation (6.2).

6.2 Discrete Problem

In this section, we aim to find an approximate solution to problem (6.3), by using the Galerkin method (see [34, p.30]), which involves formulating the discrete Poisson problem.

6.2.1 Approximate Problem

The Galerkin method approximates the infinite-dimensional space V with a finite-dimensional subspace V_h of V, which is referred to as the approximation space. The space V_h is defined as the space of continuous functions whose restriction to any cell of the mesh $\mathcal{T}_h$ (refer to Section 6.2.2 below) belongs to an approximation space $\mathcal{P}$ of finite dimension, which will be characterized further in detail in Section 8.2, usually a space of polynomials.

$$V_h \overset{\text{def.}}{=} \{v_h \in \mathcal{C}^0(\mathbb{R}^d, \mathbb{R}) \mid \forall K \in \mathcal{T}_h, v_{h|K} \in \mathcal{P}\} \subset V.$$

Consequently, by constructing an appropriate approximation space V_h, the formulation for the weak discrete Poisson problem is

$$\text{find } u_h \in V_h, \text{ such that:} \qquad \forall v_h \in V_h, \quad a(u_h, v_h) = \ell(v_h). \tag{6.4}$$

The Lax Milgram theorem applies directly to Equation (6.4) when computing approximate solutions. The next question to consider is the accuracy of the approximate solution u_h with respect to the exact solution u. To address this, we refer to the Lemma of Céa, which provides

a bound on the error of the Galerkin approximation [19, 48], and states that the error between the continuous solution u and the discrete solution u_h is bounded by

$$\forall v_h \in V_h,\ \|u - u_h\|_V \leqslant \frac{c}{\alpha}\ \|u - v_h\|_V\,,$$

where $c > 0$ and $\alpha > 0$ are the continuity and the coercivity constants of the bilinear form a (see Section 6.1.4). More specifically, this means that the error in the finite element approximation $(u - u_h)$ is bounded by a constant $\frac{c}{\alpha}$ times the smallest error of the best possible approximation of u achievable by any function v_h within the subspace V_h. Moreover, a smaller value of the factor $\frac{c}{\alpha}$ leads to a smaller error bound, which in turn indicates that the solution u_h produced by the FEM is closer to the true solution u.

6.2.2 Building the Mesh

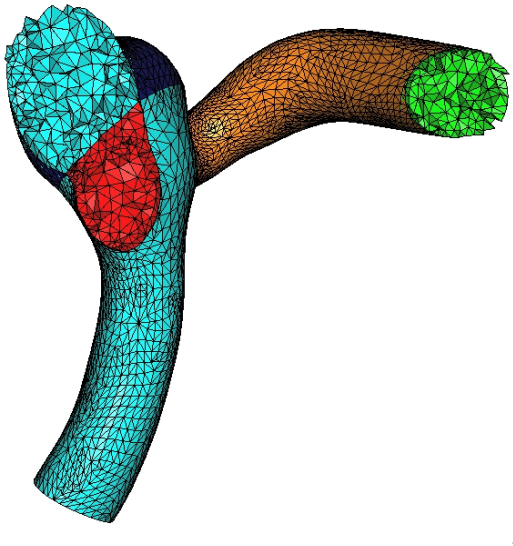

Figure 6.2: Visual representation of a 3D model of a cerebral aneurysm with a mesh made of tetrahedrons [38].

The discrete problem (6.4) constitutes a linear system. This latter is obtained by discretizing the domain Ω into finite elements as depicted in Figure 6.2 (see eg, [40] and [66, Chapter 3 p.73], and expressing the approximate solution u_h and a test function v_h in terms of basis functions associated with these elements as will be established further in Section 6.2.3. Specifically, we consider a mesh $\mathcal{T}_h$, which partitions the domain $\overline{\Omega}$ into a collection of closed subsets known as the cells of the mesh (see [34, p.54]), or geometric elements $\{K_m\}_{m=1}^{N_c}$, where N_c is the total number of cells. Each cell K_m within the mesh $\mathcal{T}_h$ has a nonempty interior (i.e., the largest open set that is completely contained within K_m), and these interiors are mutually disjoint, thereby ensuring complete coverage of the domain Ω,

$$\left(\forall m \in [1..N_c], \overset{\circ}{K}_m \neq \varnothing\right) \wedge \left(\forall m, n \in [1..N_c], m \neq n \Rightarrow \overset{\circ}{K}_m \cap \overset{\circ}{K}_n = \varnothing\right) \wedge \left(\bigcup_{m=1}^{N_c} K_m = \overline{\Omega}\right).$$

In this context, $\overset{\circ}{K}_m$ represents the interior of the cell K_m. Typically, the geometric shapes of K_m are intervals in one-dimensional space, triangles or quadrilaterals in two-dimensional spaces, and tetrahedra or cuboids in three-dimensions (see Figure 1.2). For the purposes of this thesis, we will focus exclusively on working with *simplices*, which are a generalization of the

concept of a triangle or tetrahedron to arbitrary dimensions.

The parameter $h > 0$ of $\mathcal{T}_h$ denotes the degree of the mesh refinement. A smaller h value indicates a finer mesh with smaller, more numerous cells, hopefully leading to more precise results. Specifically, h is defined as the diameter of the largest cell, as visually depicted in Figure 6.3.

$$h = \max_{1 \leqslant m \leqslant N_c} \operatorname{diam}(K_m).$$

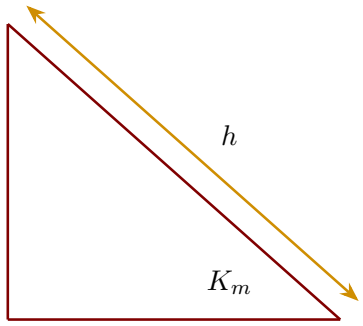

Figure 6.3: Geometric parameter h for a triangle K_m, which represents the diameter of the cell K_m.

To simplify the notation, the index m of the cells K_m is omitted in further developments.

A mesh is said to be *conforming* (see [34, p.61]) when, for example, in dimension $d = 2$, the intersection of two distinct cells is either empty, or consists of a vertex, an edge, as illustrated in Figure 6.4.

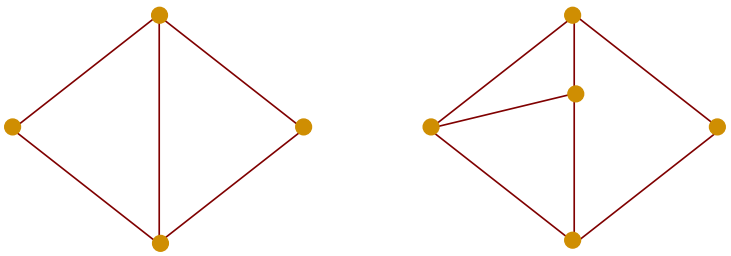

Figure 6.4: On the left, two triangular cells in a conforming mesh, where the intersection of the two cells is an entire edge, ensuring continuity across the mesh. On the right, three triangular cells form a non-conforming mesh because the vertex of one cell lies along the interior of an edge of another cell.

In the framework of FEM, two types of geometric configurations exist: the *reference geometric element* $\hat{K}$ and the *current geometric element* K. The *reference element*, denoted as $\hat{K}$, is a unit right simplex. i.e., it is the interval $[0, 1]$ in one-dimensional space, the unit isosceles right triangle in two-dimensional spaces, and the unit right tetrahedron in three-dimensional spaces.

In a space of dimension $d \geq 1$, the reference cell $\hat{K}$ is formed by $d+1$ points in $\mathbb{R}^d$, known as the reference vertices $(\hat{\mathbf{v}}_i)_{i=0}^d$ which will be discussed further in Section 9.1.1. It is characterized as a compact, connected polytope with a non-empty interior. Specifically, $\hat{K}$ is represented as the unit right simplex (see [35, p.21]), defined by,

$$\hat{K} \overset{\text{def.}}{=} \left\{ \hat{\mathbf{x}} \in \mathbb{R}^d \mid 0 \leqslant \hat{\mathbf{x}}_i \leqslant 1 \text{ for } i = [0..d-1], \text{ and } \sum_{i=0}^{d-1} \hat{\mathbf{x}}_i \leqslant 1 \right\} \subset \mathbb{R}^d. \tag{6.5}$$

By convention, the reference vertices $(\hat{\mathbf{v}}_i)_{i=0}^d$ are defined by $\hat{\mathbf{v}}_0 = 0$, and for all $\forall i \in [1..d], \hat{\mathbf{v}}_i = \boldsymbol{\delta}_{i-1}$ (see Section 9.1.1).

From the reference cell $\hat{K}$, we generate each cell of the mesh $\mathcal{T}_h$, referred to as the *current cells* K within the physical domain where the problem is defined. This process involves a family of C^1-diffeomorphic geometric mappings, denoted as T_{geo}^K, which transforms $\hat{K}$ onto each individual cell K in $\mathcal{T}_h$. A simplex K in $\mathbb{R}^d$ is defined as the convex hull of a set of $d+1$ current vertices. These vertices, represented as $(\mathbf{v}_i)_{i\in[0..d]}$, are points in $\mathbb{R}^d$. The convex hull of these points, K, is the smallest convex set that contains all the vertices. It can be mathematically expressed as follows,

$$K \overset{\text{def.}}{=} \left\{ \mathbf{x} = \sum_{i=0}^{d} \mu_i \mathbf{v}_i \in \mathbb{R}^d \mid \forall i \in [0..d],\ \mu_i \geq 0 \wedge \sum_{i=0}^{d} \mu_i = 1 \right\}. \tag{6.6}$$

The geometric transformation T_{geo}^K is mathematically defined as,

$$T_{\text{geo}}^K : \hat{K} \ni \hat{\mathbf{x}} \quad \longmapsto \quad \sum_{i=0}^{d} \hat{\psi}_i(\hat{\mathbf{x}}) \mathbf{v}_i \in K. \tag{6.7}$$

The $\hat{\psi}_i$ are known as local shape functions, polynomial functions specified on $\hat{K}$. Each shape function $\hat{\psi}_i$ is uniquely defined to fulfill the condition $\hat{\psi}_i(\hat{\mathbf{v}}_j) = \delta_{i,j}$ for $i, j \in [0..d]$, where $\delta_{i,j}$ is the Kronecker delta function defined in Section 5.3.5. This condition ensures that $T_{\text{geo}}^K(\hat{\mathbf{v}}_i) = \mathbf{v}_i$ for each vertex $\hat{\mathbf{v}}_i$ of the reference cell $\hat{K}$. The transformation T_{geo}^K is an affine map in $\mathcal{C}^\infty(\mathbb{R}^d)$ associated with an invertible Jacobian matrix denoted by $\boldsymbol{J}_{\text{geo}}^K$ (see e.g. [42]). For additional examples and a detailed discussion of geometric mappings, refer to Section 9.3 and see [36, p.90].

6.2.3 Building the Linear System

This section outlines the process of computing the global matrix by solving a system of linear equations. Currently, this part has not been formalized in Coq; it is included here solely to offer a conceptual understanding of how the Finite Element Method (FEM) operates and how the approximate solutions are computed (see Figure 6.1).

We recall from Section 6.2.1, that the weak discrete formulation of the Poisson problem is defined as follows:

$$\text{find } u_h \in V_h, \text{ such that for all } v_h \in V_h, \quad a(u_h, v_h) = \ell(v_h).$$

With the construction of a conforming mesh, we are now equipped to derive a corresponding system of linear equations. Let $(\phi_j)_{j\in[1..N_{\text{dof}}]}$ represent the *global shape functions* forming a basis for V_h, where N_{dof} indicates the total number of the global *degrees of freedom* in the mesh $\mathcal{T}_h$, which is also equal to the dimension of the space V_h. However, the detailed construction of the global shape functions $(\phi_j)_{j\in[1..N_{\text{dof}}]}$ will not be discussed. By setting $v_h = \phi_i$ for all $i \in [1..N_{\text{dof}}]$ and expressing the approximate solution u_h in terms of this basis, we obtain the following formulation,

find $u_h = \sum_{j=1}^{N_{\text{dof}}} U_j \phi_j$ such that:

$$\forall i \in [1..N_{\text{dof}}], \quad \sum_{j=1}^{N_{\text{dof}}} a(\phi_j, \phi_i) U_j = \ell(\phi_i). \tag{6.8}$$

This problem can also be represented as,

$$\begin{array}{l} \text{find } U \overset{\text{def.}}{=} [U_1, \ldots, U_{N_{\text{dof}}}]^\top \text{ in } \mathbb{R}^{N_{\text{dof}}}, \text{ such that:} \\ AU = b, \qquad \text{where } \left\{ \begin{array}{rcll} A_{ij} & = & a(\phi_j, \phi_i), & i,j \in [1..N_{\text{dof}}], \\ b_i & = & \ell(\phi_i), & i \in [1..N_{\text{dof}}]. \end{array} \right. \end{array} \tag{6.9}$$

Here, the matrix A is often called the *stiffness matrix*, U is the vector of unknown coefficients that are the components of the discrete solution u_h in the basis $(\phi_j)_{j\in[1..N_{\text{dof}}]}$, and b is the vector representing the integral of the source term f multiplied by the basis functions. The final task of finding the solution involves solving this linear system through numerical linear algebra techniques. The computation of the matrix A involves evaluating the bilinear form a for every pair of basis functions.

$$\begin{aligned} a(\phi_j, \phi_i) &= \int_\Omega \boldsymbol{\nabla}\phi_j \cdot \boldsymbol{\nabla}\phi_i \\ &\overset{(1)}{=} \sum_{K\in\mathcal{T}_h} \int_K \boldsymbol{\nabla}\phi_j \cdot \boldsymbol{\nabla}\phi_i \\ &\overset{(2)}{=} \sum_{K\in\mathcal{T}_h} \int_{\hat{K}} (\boldsymbol{J}^K_{\text{geo}})^{-T} \hat{\boldsymbol{\nabla}}\hat{\phi}_j \cdot (\boldsymbol{J}^K_{\text{geo}})^{-T} \hat{\boldsymbol{\nabla}}\hat{\phi}_i \, |\det \boldsymbol{J}^K_{\text{geo}}| \\ &\overset{(3)}{\approx} \sum_{K\in\mathcal{T}_h} \sum_{i_g=1}^{N_q} \left((\boldsymbol{J}^K_{\text{geo}})^{-T} \hat{\boldsymbol{\nabla}}\hat{\phi}_j \cdot (\boldsymbol{J}^K_{\text{geo}})^{-T} \hat{\boldsymbol{\nabla}}\hat{\phi}_i \, |\det \boldsymbol{J}^K_{\text{geo}}| \right) (\hat{\boldsymbol{\xi}}_{i_g}) \, \hat{\omega}_{i_g}. \end{aligned} \tag{6.10}$$

The integral over the entire domain Ω is first represented as a sum of integrals over each mesh element, as shown by the assertion (1).

Following this, in assertion (2), we transform each integral over the current element K into an integral over the reference element $\hat{K}$, (defined above in the Equation (6.6) and (6.5), respectively) by a change of variables. This transformation uses the inverse transpose of the Jacobian matrix [36, Section 8.1, p.89] of the geometric mapping T^K_{geo}, which maps gradient vectors from the reference to the current element. We also use the gradients of the basis functions on the reference element, $\hat{\nabla}\hat{\phi}_i$ and $\hat{\nabla}\hat{\phi}_j$, scaled by the absolute value determinant of the Jacobian matrix, $|\det \boldsymbol{J}^K_{\text{geo}}|$, to adjust for the deformation between the reference element and the current element.

Finally, assertion (3) approximates the integral over the reference element $\hat{K}$ using the numerical quadrature method ([34, p.213]). This approximation is evaluated at specific quadrature points $\hat{\boldsymbol{\xi}}_{i_g}$, multiplied by the corresponding quadrature weights $\hat{\omega}_{i_g}$, for all basis functions $(\hat{\phi}_j)_{j\in[1..N_{\text{dof}}]}$ of the mesh.

6.3 General Definition of a Finite Element

This section introduces the general definition of the finite element triplet, which establishes the structure and mathematical formulation of finite elements. The formalization of this triplet

in Coq will be presented in detail in Section 7.1, where essential proofs and definitions will support its implementation. For instance, the unisolvence property, discussed in the following Section 6.4, will also be addressed as part of this formalization.

At the heart of finite element analysis lies the concept of a finite element defined as a triple $(K, \mathcal{P}, \Sigma)$ (see e.g [19, p.93]), where each component plays a role in defining how a function is approximated and represented on a cell K of the mesh $\mathcal{T}_h$.

(i) K denotes a geometric element that is a compact, connected subset of $\mathbb{R}^d$ with a non-empty interior. This geometric element can take various forms—such as a triangle, quadrilateral, tetrahedron, or hexahedron, depending on the complexity and requirements of the simulation (see Section 6.2.2).

(ii) $\mathcal{P}$ represents a finite-dimensional vector space of functions defined on each geometric element K. The dimension of this space, indicated by n_{dof}, represents the number of local degrees of freedom within each element. $\mathcal{P}$ serves as the approximation space for functions, typically polynomials of bounded degree. It provides a space for representing the solution within each geometric element, where the choice of polynomial degree (linear, quadratic, etc.) can be increased in order to adjust the accuracy of the solution.

(iii) Σ is a family of n_{dof} linear forms acting on functions within the space $\mathcal{P}$. This family is denoted as $\Sigma \overset{\text{def.}}{=} \{\sigma_i\}_{i\in[0..n_{\text{dof}}-1]}$. These linear forms represent the local degrees of freedom within a finite element and are designed to ensure that the mapping $\Phi_\Sigma : \mathcal{P} \to \mathbb{R}^{n_{\text{dof}}}$, given by

$$\Phi_\Sigma(p) := (\sigma_i(p))_{i\in[0..n_{\text{dof}}-1]},$$

establishes an isomorphism. This isomorphism ensures a bijective correspondence between the function space $\mathcal{P}$ and the vector space $\mathbb{R}^{n_{\text{dof}}}$, thereby Σ forms a basis for the space of linear forms $\mathcal{L}(\mathcal{P}, \mathbb{R})$, which is the dual space of $\mathcal{P}$. This property is known as the *unisolvence property*, and will be described in more detail in the following Section 6.4.

While several families of finite elements exist, such as Crouzeix–Raviart, Raviart–Thomas, and Nédélec [34, 35, 36, 55], this thesis focuses on the Lagrange finite elements [34, 16], which are the most classical and widely used. Within the Lagrange family, for each $i \in [0..n_{\text{dof}}-1]$, σ_i is the evaluation at evenly distributed points within each geometric element K. These points, known as Lagrange nodes, include the vertices and will be further elaborated upon in Section 9.1. More specifically, the unisolvence property ensures that each family of coefficients of polynomials in $\mathcal{P}$ is uniquely determined by the values of the polynomial at specified nodes. These coefficients are the unknowns that need to be computed within the finite element method. Once these coefficients are accurately determined, they precisely define the polynomial that approximates the solution to the partial differential equation in each geometric element. When the total degree $k = 1$ or the spatial dimension $d = 1$, the Lagrange finite elements use Lagrange bases, which are polynomial functions defined on each element of the mesh. These polynomials ensure interpolation at these nodes, and their definition will be given in Section 9.5.

6.4 Unisolvence Principle

This section details the unisolvence principle (see e.g. [36, Section 7.4 p.79]) and outlines the necessary conditions to establish it in the context of the finite element method. This property is fundamental to the reliability of the finite element method, ensuring that the solution to

the mathematical problem model is uniquely determined by the values taken by the degrees of freedom in each element of the mesh.

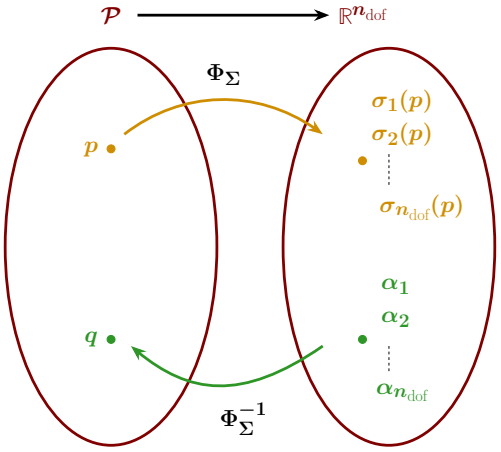

Figure 6.5: The bijective relationship between the approximation space $\mathcal{P}$ and the real number space $\mathbb{R}^{n_{\text{dof}}}$.

As discussed previously in Section 6.3, a finite element is defined as a triplet $(K, \mathcal{P}, \Sigma)$. We recall that unisolvence refers to the bijectivity of the mapping Φ_Σ, satisfying condition (iii) from Section 6.3 (see Figure 6.5). This bijectivity ensures a unique correspondence between the function space $\mathcal{P}$ and the vector space $\mathbb{R}^{n_{\text{dof}}}$, expressed as:

$$\forall(\alpha_0, \ldots, \alpha_{n_{\text{dof}}-1}) \in \mathbb{R}^{n_{\text{dof}}},\ \exists! p \in \mathcal{P}, \forall i \in [0..n_{\text{dof}} - 1],\ \sigma_i(p) = \alpha_i.$$

To verify this condition, one must equivalently demonstrate the following properties:

$$\dim \mathcal{P} = \text{card } \Sigma = n_{\text{dof}}. \tag{6.11}$$

$$\forall p \in \mathcal{P},\ (\forall i \in [0..n_{\text{dof}} - 1], \sigma_i(p) = 0) \Rightarrow (p = 0). \tag{6.12}$$

When these properties are satisfied, the finite element is referred to as unisolvent. The first condition (6.11) ensures the dimension of the polynomial space $\mathcal{P}$ and the cardinal of the family of linear forms are the same. The second condition (6.12), expresses that the kernel of Φ_Σ reduces to the zero vector (i.e., $\text{Ker}(\Phi_\Sigma) = \{p \in \mathcal{P} \mid \Phi_\Sigma(p) = 0\}$), hence its injectivity.

This section on the unisolvence property of finite elements will be very useful in the later chapters, as it establishes the unique correspondence between the degrees of freedom and the functions in the approximation space. This property relies on a series of algebraic concepts, including bijectivity, injectivity, and their equivalence, along with related properties. All these algebraic results are thoroughly formalized in Coq in Section subsec:fun:bijS.

Chapter 7

Formalization of Finite Elements and Related Properties

This chapter specifically builds upon the work done in Chapter 6 concerning the general definition of a finite element, represented as a triplet within individual cells K of the mesh $\mathcal{T}_h$. In Section 7.1, we represent the formalization of this triplet in the Coq proof assistant, using a record-based approach that includes the foundational components of a finite element. In Section 7.2, we define the shape functions as a basis for the polynomial space $\mathcal{P}$. This latter was briefly introduced in Section 6.3. Following this, in Section 7.3, we develop the local interpolation operator associated with this finite element.

7.1 Formalization of Finite Elements: A Record-Based Approach

In this section, we introduce a record type named `FE`. The fields of this record provide a comprehensive representation of finite elements, building upon concepts discussed in Sections 6.3 and 6.4. It includes the necessary components required to approximate solutions to partial differential equations (PDEs) within each mesh cell. This is achieved by exploring the interconnections among geometric elements, function spaces, and linear forms within the finite element method. However, the structure of this record is not limited to these three elements; it also includes additional significant assumptions, which will be detailed subsequently.

Let us consider the structure of the finite element `FE` record and highlight the specifics of each field within this record, which will be described afterwards to understand their contributions, as demonstrated in the following code snippet.

```
Record FE : Type := mk_FE {
  d : nat;                                  (* spatial dimension, e.g., 1, 2, 3 *)
  ndof : nat;                               (* number of degrees of freedom *)
  d_pos : 0 < d;                            (* dimension is positive *)
  ndof_pos : 0 < ndof;                      (* degrees of freedom are positive *)
  shape : shape_type;                       (* geometric shape of the element *)
  nvtx : nat := nvtx_of_shape d shape;      (* number of vertices *)
  vtx : '(' R^d)^nvtx;                      (* vertices of geometrical element *)
  K_geom : 'R^d → Prop := convex_envelop vtx; (* geometric element *)
  P_approx : FRd d → Prop;                  (* polynomial approximation space *)
  P_approx_has_dim : has_dim P_approx ndof; (* subspace of dimension ndof *)
```

```
  Sigma : '( FRd d → R)^ndof;                (* linear forms over the approximation space *)
  Sigma_lm : ∀ i, lin_map (Sigma i);         (* linearity of each form *)
  unisolvence : bijS P_approx fullset (gather Sigma); (* bijectivity *)
}.
```

This record type encapsulates various properties relevant to a finite element. In subsequent developments, when we need to access any of these properties or fields, we refer to them through an instance `fe` of type `FE`.

The record begins by defining `d` as the spatial dimension, and `ndof` as the number of degrees of freedom, which corresponds to the dimension d and the number n_{dof} as mentioned in Section 6.3. The constraints `d_pos` and `ndof_pos` ensure that these values are positive, highlighting that there is at least one degree of freedom.

The shape of the element is described by the `shape` field, defined through the enumeration `shape_type`, which identifies different geometric shapes—specifically, a `Simplex` or a `Quad` as illustrated in Figure 7.1 in 2 dimensions. The choice of shape affects the `nvtx` computation, which determines the number of vertices of the shape by the function `nvtx_of_shape` depending on the dimension `d` and the selected `shape`. The corresponding code implementing these type and function is given below,

```
Inductive shape_type := Simplex | Quad.
Definition nvtx_of_shape (d : nat) (shape : shape_type) : nat :=
  match shape with | Simplex ⇒ d.+1 | Quad ⇒ 2^d end.
```

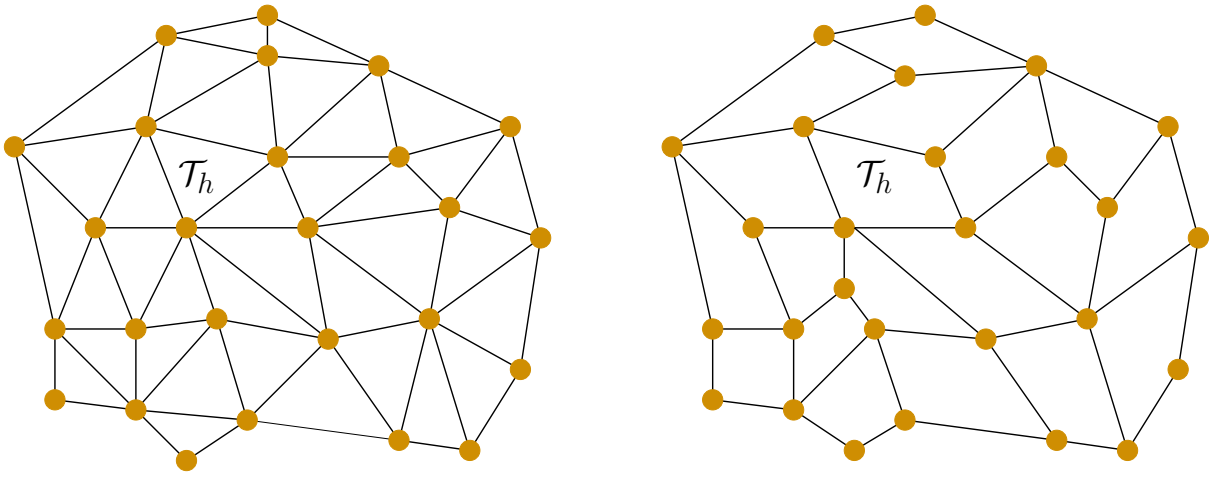

Figure 7.1: Illustration of a mesh $\mathcal{T}_h$ with two configurations: the right mesh is divided into 2D quadrangles, and the left mesh is divided into 2D triangles. The orange points represent the vertices of the mesh.

The vertices `vtx` of a geometric element are represented as a family of points within the space `'R^d`, with a size of `nvtx`. We recall from Section 5.2.3, that the notation `'(' R^d)^nvtx` represents the type of a family of `nvtx` elements where each element is of type `'R^d`. The value `nvtx` is calculated based on the shape of the geometric element and does not represent a parameter of the type `FE`. Additionally, the field `K_geom` is derived from the subset of `'R^d` describing the geometric element itself and is also a calculated attribute rather than a parameter of type FE. The element K represents the first component of the finite element triple, which is mathematically expressed in Section 6.2.2 through Equation (6.6) and discussed in Section 6.3. Examples of geometric elements include segments in $\mathbb{R}$, triangles in $\mathbb{R}^2$, and tetrahedra in $\mathbb{R}^3$. The

K_geom field is defined as the convex hull of the vertices, constructed using the convex_envelop predicate (see Equation (6.6) in Section 6.2.2). This predicate is defined in the Coq as an inductive type, as illustrated below:

```
Context {E : ModuleSpace R_Ring}.
Inductive convex_envelop {n} (V : 'E^n) : E → Prop :=
  | Cvx : ∀ (L : 'R^n), (∀ i, 0 ⩽ L i) → sum L = 1 →
      convex_envelop V (lin_comb L V).
```

Here, the function lin_comb computes the linear combination of elements in the module space E with coefficients from the ring ℝ. This function is detailed in Section 5.3.4.

This definition implies that any point in 'R^d belongs to K_geom if it can be expressed as a convex combination of the vertices vtx of a finite element within the mesh. Specifically, a convex combination, specified by the constructor Cvx, requires that the coefficients (weights) L, which are used to combine these vertices, are between 0 and 1, and sum up to one. This condition ensures that every resulting point lies within the geometric element or is situated on its boundary (inside the triangle in 2D for example). In this record, we have not formally established that the interior of the geometrical element K_geom is non-empty.

We now turn our attention to the second component of the finite element triplet, the approximation space P_approx, as detailed in Section 6.3. P_approx is a subspace of the vector space FRd d := 'R^d → R, which is the functional space of real-valued functions defined over 'R^d (refer to Sections 5.3.3 and 5.2.3 for further details on modules and families of functions). The dimension of P_approx is determined by the property P_approx_has_dim, which corresponds precisely to the number of degrees of freedom, denoted as ndof. For further details on the definition of the dimension, refer to Section 5.4.2.

Lastly, the third component of the finite element triplet, outlined in Section 6.3, is the field Sigma : '(FRd d → R)^ndof. This field represents a collection of ndof linear forms, which are collectively denoted as Σ. These forms correspond to the local degrees of freedom (DOF) within a geometric element of the mesh. These DOFs refer to the independent values that define the shape of the solution in each element of the mesh. These might include values such as function values at the mesh nodes (nodal values), derivatives at nodes, or integral values over the elements. Each function Sigma i within this family is a linear map, assumed in Sigma_lm and specified by lin_map property, a concept thoroughly detailed in Section 5.3.3. Furthermore, the unisolvence field specifies that gather Sigma : FRd d → 'R^ndof is bijective, denoted by the predicate bijS. This bijectivity exists between the approximation space P_approx and the full vector space 'R^ndof, denoted here as fullset. For a more detailed discussion on the unisolvence property, refer to Section 6.4, and for further insights into the nature of bijS, consult Section 5.1.4. The role of gather Sigma is to transform the family Sigma of linear forms into a single function that maps any input from FRd d to a vector in 'R^ndof.

7.2 Shape Functions of Finite Element

This section covers the concept of shape functions, as elaborated in [35], which are used to approximate the solution to PDEs. We associate the shape functions with a finite element represented by the triplet $(K, \mathcal{P}, \Sigma)$. As discussed in Section 6.3 (item iii), the functions within Σ serve as a basis for the dual space of linear forms $\mathcal{L}(\mathcal{P}, \mathbb{R})$. Consequently, there exists a family of functions $(\theta_i)_{i \in [0..n_{\text{dof}}-1]}$, referred to in Coq as theta, that belongs to the polynomial

space $\mathcal{P}$, with Σ forming the dual basis to θ. For more detail about duality, refer to Section 5.4.4.

We assume a variable `fe` of type `FE`. The conditions for the functions, θ, to qualify as local shape functions for a finite element `fe` are formally specified as follows:

```
Definition is_local_shape_fun := fun (theta : '(FRd (d fe))^(ndof fe)) ⇒
  (∀ i : 'I_(ndof fe), P_approx fe (theta i)) ∧
  (∀ i j : 'I_(ndof fe), Sigma fe i (theta j) = kronecker i j).
```

Here, `'I_(ndof fe)` is a notation used in Coq to represent an ordinal (see Sections 2.3 and 5.2.2). The `theta` input consists of a family of `ndof fe` functions, each of which is defined within the function space `FRd (d fe)`. We recall that `ndof fe` indicates the number of degrees of freedom associated with the finite element `fe`, and `FRd (d fe)` refers to the vector space of functions mapping from the real space $\mathbb{R}^d$ to $\mathbb{R}$, where `d fe` indicates the spatial dimension. Specifically, for the family $(\theta_i)_{i\in[0..n_{\text{dof}}-1]}$ to be considered as local shape functions, they must satisfy two conditions. Firstly, each function θ_i must belong to the approximation space `P_approx fe` associated with `fe`. Secondly, each function θ_j must fulfill the Kronecker delta property (see Section 5.3.5) according to the degrees of freedom specified by the linear forms `Sigma fe`. This requirement is mathematically articulated as:

$$\forall i, j \in [0..n_{\text{dof}} - 1], \quad \sigma_i(\theta_j) = \delta_{i,j}. \tag{7.1}$$

The construction of local shape functions $(\theta_i)_{i\in[0..n_{\text{dof}}-1]}$ is represented as the predual basis of $(\sigma_i)_{i\in[0..n_{\text{dof}}-1]}$ within each finite element (refer to Section 5.4.4), using the unisolvence condition.

```
Definition shape_fun : '(FRd (d fe))^(ndof fe) := predual_basis (unisolvence fe).
```

The concept of the `predual_basis` is discussed in Section 5.4.4. By this definition, we say that the defined `shape_fun` indeed qualifies as local shape functions in accordance with the `is_local_shape_fun` definition,

```
Lemma shape_fun_correct : is_local_shape_fun shape_fun.
```

Subsequently, we can verify that `shape_fun` forms a basis for the polynomial approximation space $\mathcal{P}$ of a finite element `fe`.

```
Lemma shape_fun_basis : basis (P_approx fe) shape_fun.
```

This proof is concise and directly follows from the lemma `predual_basis_basis`, which is detailed in Section 5.4.4.

7.3 Construction of a Local Interpolation Operator

The finite element method employs interpolation to approximate solutions to partial differential equations within the finite element mesh, such as the Poisson equation discussed in Chapter 6. The objective of this section is to construct a local interpolation operator, as outlined in [34], using the elements $(K, \mathcal{P}, \Sigma)$. This operator aims to approximate functions defined on the geometric element K (a subset of Ω) by using elements from $\mathcal{P}$ whose degrees of freedom are suitably chosen.

For a function v, the local interpolation process is described by the operator $\mathcal{I}_h$, defined as,

$$\mathcal{I}_h : \mathcal{F}(\mathbb{R}^d, \mathbb{R}) \ni v \mapsto \sum_{i=0}^{n_{\text{dof}}-1} \sigma_i(v)\ \theta_i \ \in \mathcal{P}. \tag{7.2}$$

The notation $\mathcal{F}(\mathbb{R}^d, \mathbb{R})$ refers to a space of functions from $\mathbb{R}^d$ to $\mathbb{R}$.

In the Coq proof assistant, we define the local interpolation operation as:

```
Definition local_interp : FRd (d fe) → FRd (d fe) :=
  fun v ⇒ lin_comb (fun i ⇒ Sigma fe i v) shape_fun.
```

where `lin_comb` is a function that implements the mathematical concept of a linear combination in a module space (refer to Section 5.3.4 for more detail). By definition, the interpolation operator is a linear mapping that takes its values in the approximation space $\mathcal{P}$.

```
Lemma local_interp_is_poly : ∀(v : FRd (d fe)), P_approx fe (local_interp v).
```

This lemma is established since $\mathcal{P}$, a subspace of $\mathcal{F}(\mathbb{R}^d, \mathbb{R})$, is a module space, it is closed under addition and scalar multiplication (refer to Section 5.3.3). This closure property ensures that any linear combination of the functions $(\theta_i)_{i\in[0..n_{\text{dof}}-1]}$ within $\mathcal{P}$ also remains within $\mathcal{P}$. Consequently, it follows that the interpolation operator $\mathcal{I}_h(v)$ belongs to $\mathcal{P}$.

Furthermore, when the interpolation operator $\mathcal{I}_h$ is applied to a function v, it produces a new function known as the interpolated function $\mathcal{I}_h v$, which serves as an approximation of v within the finite element mesh. Consequently, we can verify that the degrees of freedom for $\mathcal{I}_h v$ exactly match those of the function v. The relevant lemma is stated as follows:

```
Lemma Sigma_local_interp : ∀(i : 'I_(ndof fe)) (v : FRd (d fe)),
    Sigma fe i (local_interp v) = Sigma fe i v.
```

The proof of this lemma is straightforward and proceeds as follows, based on the linearity of the functions σ_i, Equation (7.1), and the properties of the Kronecker delta function $\delta_{i,j}$. For each $j \in [0..n_{\text{dof}} - 1]$ we have:

$$\sigma_j(\mathcal{I}_h v) = \sigma_j\left(\sum_{i=0}^{n_{\text{dof}}-1} \sigma_i(v)\theta_i\right) = \sum_{i=0}^{n_{\text{dof}}-1} \sigma_i(v)\sigma_j(\theta_i) = \sum_{i=0}^{n_{\text{dof}}-1} \sigma_i(v)\delta_{i,j} = \sigma_j(v).$$

□

Another noteworthy result is that the interpolation operator $\mathcal{I}_h$ is a projection from $\mathcal{F}(\mathbb{R}^d, \mathbb{R})$ to $\mathcal{P}$. This implies that for any polynomial p within the space $\mathcal{P}$, the operator $\mathcal{I}_h$ effectively preserves p without any changes. The corresponding lemma is formally stated as follows:

```
Lemma local_interp_proj : ∀p : FRd (d fe), (P_approx fe) p → local_interp p = p.
```

Let us verify this. Since $p \in \mathcal{P}$, then p is expressed as $p = \sum_{j=0}^{n_{\text{dof}}-1} x_j\theta_j$, as $(\theta_i)_{i\in[0..n_{\text{dof}}-1]}$ is proved to be a basis for $\mathcal{P}$ (see Section 7.2). Then, applying $\mathcal{I}_h$ yields,

$$\mathcal{I}_h p = \sum_{i=0}^{n_{\text{dof}}-1} \sigma_i(p)\ \theta_i = \sum_{i=0}^{n_{\text{dof}}-1}\sum_{j=0}^{n_{\text{dof}}-1} x_j\ \sigma_i(\theta_j)\ \theta_i = \sum_{i=0}^{n_{\text{dof}}-1}\sum_{j=0}^{n_{\text{dof}}-1} x_j\ \delta_{i,j}\ \theta_i = \sum_{j=0}^{n_{\text{dof}}-1} x_j\ \theta_j = p.$$

Furthermore, applying $\mathcal{I}_h$ twice to any function v in the space $\mathcal{F}(\mathbb{R}^d, \mathbb{R})$ results

$$\mathcal{I}_h(\mathcal{I}_h v) = \mathcal{I}_h v.$$

This demonstrates the idempotency of the interpolation operator $\mathcal{I}_h$.

Chapter 8

Constructing the Polynomial Space $\mathcal{P}_k^d$

In this chapter, we construct a polynomial space, denoted as $\mathcal{P}_k^d$ of polynomials of degree at most k on $\mathbb{R}^d$ [34, 35, 36, 59], and define one of its bases. The main concept of this study is the family of multi-indices, denoted as $\mathcal{A}_k^d$, essential for constructing polynomials for the approximation space $\mathcal{P}_k^d$ (refer to Section 5.2.3 for more details on finite families). We will particularly study cases where the degree of approximation is $k = 1$ and the spatial dimension is $d = 1$. The reason behind focusing on these cases will become apparent in subsequent chapters (see Section 10.3.7), particularly when we address the proof of the unisolvence property of the Lagrange finite element. This proof employs double induction on both d and k, starting from 1.

This chapter is structured into four sections: Section 8.1 introduces multi-indices and their applications; Section 8.2 discusses the structure of the $\mathcal{P}_k^d$ polynomial space and outlines a basis of this space; Section 8.3 details the specifics of $\mathcal{P}_1^d$ space, focusing on affine polynomials in d dimensions; and Section 8.4 deals with $\mathcal{P}_k^1$ space, addressing polynomial functions in a single variable.

8.1 Multi-Indices

This section introduces the concept of *multi-indices* [36, Section 7.3, p.78] and discusses their role in constructing multivariate polynomial spaces, particularly in constructing bases for the polynomial approximation space. Specifically, the family $\mathcal{A}_k^d$ is a collection of multi-indices used to construct polynomials up to a certain degree k and within a certain number of variables d. Thus, $\mathcal{A}_k^d$ provides a systematic way to enumerate all possible monomials up to a given order in a polynomial space and consists in defining the indices of the nodes within a geometrical element (see Figure 8.2). Here is a formal definition and explanation of how this family is constructed.

A multi-index $\boldsymbol{\alpha}$ is a tuple of nonnegative integers for all d,

$$\boldsymbol{\alpha} \overset{\text{def.}}{=} (\alpha_0, \alpha_1, \ldots, \alpha_{d-1}) \in \mathbb{N}^d,$$

where each α_i represents the power or degree of the variable x_i in the multivariate monomial $(x_0, \ldots, x_{d-1}) \longmapsto \prod_{i=0}^{d-1} x_i^{\alpha_i}$ (For more detailed information about monomials, refer to Section 5.3.2). The dimension d represents the number of variables, and the elements of $\boldsymbol{\alpha}$ specify

the degree to which each variable is raised. We define $|\boldsymbol{\alpha}| \overset{\text{def.}}{=} \alpha_0 + \alpha_1 + \ldots + \alpha_{d-1}$ as the total degree of the monomial.

8.1.1 Definition of $\mathcal{A}_k^d$, $\mathcal{C}_k^d$ and $\check{\mathcal{S}}_{k,k-i}^d$ Families

The family of multi-indices $\mathcal{A}_k^d$ is defined for all d as:

$$\mathcal{A}_k^d \overset{\text{def.}}{=} \{\boldsymbol{\alpha} \in \mathbb{N}^d \mid |\boldsymbol{\alpha}| \overset{\text{def.}}{=} \sum_{i=0}^{d-1} \alpha_i \leqslant k\}. \tag{8.1}$$

This family includes all multi-indices $\boldsymbol{\alpha}$ for which the sum of the indices does not exceed k.

The family $\mathcal{A}_k^d$ is a disjoint union of sub-families. Each sub-family corresponds to a specific *layer* $\mathcal{C}_\ell^d$, which is the family of multi-indices $\boldsymbol{\alpha}$ that includes indices where the sum precisely equals ℓ for $\ell \leqslant k$. This construction helps organize polynomial terms of varying degrees, where each layer includes all exponents of monomial terms in d variables that sum exactly to ℓ for $\ell \leqslant k$.

$$\mathcal{A}_k^d = \biguplus_{\ell=0}^{k} \mathcal{C}_\ell^d, \quad \text{with} \quad \mathcal{C}_\ell^d \overset{\text{def.}}{=} \{\boldsymbol{\alpha} \in \mathbb{N}^d \mid |\boldsymbol{\alpha}| = \ell\}. \tag{8.2}$$

To determine the cardinality (i.e., the number of distinct elements) of the family $\mathcal{A}_k^d$, it is essential to first verify that $\mathcal{A}_k^d$ is injective, as this family may contain repetitions. A detailed discussion of injectivity will be provided in Section 8.1.3. For now, we assume that the cardinality of $\mathcal{A}_k^d$ is computed as follows:

$$\operatorname{card} \mathcal{A}_k^d = \binom{d+k}{k} \overset{\text{def.}}{=} \frac{(d+k)!}{d!\ k!}.$$

In Coq, the cardinality of $\mathcal{A}_k^d$ is equal to the binomial expression (`pbinom d k`) + 1. For further details about binomial coefficients, refer to Section 5.5. Building on this, we will construct the family $\mathcal{A}_k^d$ of this length.

The family of *layers* $\mathcal{C}_k^d$ is structured as a disjoint union of *vertical slices*, each denoted by $\check{\mathcal{S}}_{k,k-i}^d$. For a visual representation of how these families are organized, refer to Figure 8.1.

$$\mathcal{C}_k^d = \biguplus_{i=0}^{k} \check{\mathcal{S}}_{k,k-i}^d \quad \text{with} \quad \check{\mathcal{S}}_{k,k-i}^d \overset{\text{def.}}{=} \{(k-i, \boldsymbol{\alpha}) \in \mathbb{N}^d \mid \boldsymbol{\alpha} \in \mathcal{C}_i^{d-1}\}. \tag{8.3}$$

The slices $\check{\mathcal{S}}_{k,k-i}^d$ are represented by the `Slice_fun` function and implemented through the `Slice_op` function as follows:

```
Definition Slice_op {d : nat} (i : nat) (alpha : 'nat^d) : 'nat^(d.+1) :=
  castF (add1n d) (concatF (singleF i) alpha).
Definition Slice_fun {d n} (i:nat) (a : 'I_n → 'nat^d) : 'I_n → 'nat^(d.+1) :=
  mapF (Slice_op i) a.
```

Here, a casting function, `castF`, is required to adjust the data type based on a function related to the property (`add1n d`) which asserts that adding 1 to any natural number `n` results in `n.+1`. We recall from Section 2.3, that the `.+1` notation represents the successor of a number `n`. For an in-depth explanation of the `castF` function, refer to Section 5.2.3. To maintain brevity and focus in this thesis, detailed descriptions of all casting functions are omitted. Instead, a more concise expression format is used, with details filled in using underscores.

The function `Slice_op` concatenates a single-element vector containing `i` to the multi-index `alpha`. Building upon `Slice_op`, the `Slice_fun` maps this operation across a collection of multi-indices `a`.

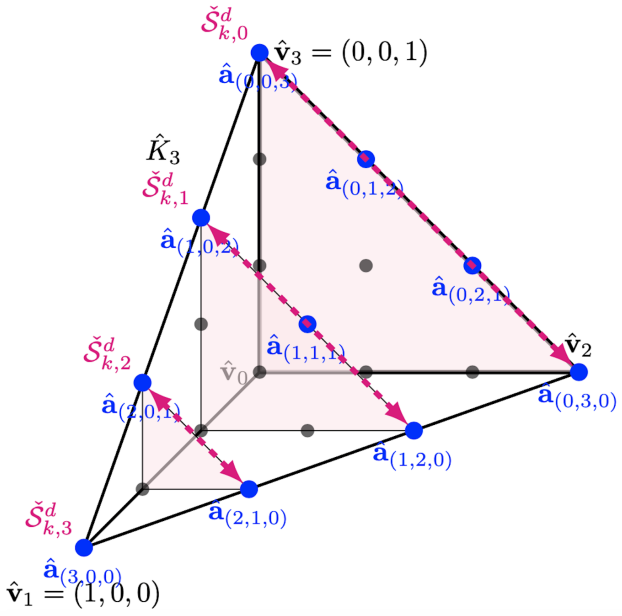

Figure 8.1: Vertical slices $\check{\mathcal{S}}^d_{k,k-i}$ in the case $d = k = 3$ (see Equation (8.3)). The reference Lagrange nodes $\hat{a}_{(\alpha_0,\alpha_1,\alpha_2)}$ in blue correspond to the element $(\alpha_0, \alpha_1, \alpha_2) \in \mathcal{C}^d_k$ (refer to Section 9.1.1). For instance, the family $\check{\mathcal{S}}^d_{k,1} = \{(1,0,2),(1,1,1),(1,2,0)\}$ is depicted by the indices of the nodes linked by a dashed arrow.

We easily deduce that the sum of the elements of the `Slice_op` family equals the degree `k`.

```
Lemma Slice_op_sum: ∀(d k i : nat) (alpha:'nat^d), i ⩽ k → sum alpha = k−i →
  sum (Slice_op i alpha) = k.
```

The proof of this lemma is derived from the definition of the `Slice_op` function. It uses the property of sum functions as outlined in the `sum_concatF` lemma, detailed in Section 5.3.1. This particular lemma asserts that the sum of two concatenated sequences equals the sum of each individual sequence. Consequently, by adding the sum of `i` to the sum of `alpha`, which is given as `k − i`, the resultant sum is `k`.

In the definition of the family $\mathcal{C}^d_k$, the notation $\biguplus$ signifies a disjoint union, meaning that each family $\check{\mathcal{S}}^d_{k,k-i}$ constitutes a distinct and separate part of the total family $\mathcal{C}^d_k$, with no shared elements among these parts. Similarly to the case of the family $\mathcal{A}^d_k$, determining the cardinality of the family $\mathcal{C}^d_k$ requires first verifying that $\mathcal{C}^d_k$ is injective (see Section 8.1.3). For the moment, we assume that the cardinality of $\mathcal{C}^d_k$ is computed as follows:

$$\operatorname{card} \mathcal{C}^d_k = \binom{d+k-1}{d-1} \overset{\text{def.}}{=} \frac{(d+k-1)!}{(d-1)!\,k!}.$$

Building on this, we will construct the family $\mathcal{C}^d_k$ corresponding to this computed length.

In Coq, the family $\mathcal{C}^d_k$ when $d > 0$ is implemented using the concatenation function `concatnF` (see Section 5.3.1), in conjunction with the `Slice_fun` function:

```
Fixpoint CSdk (d k :nat) : 'I_((pbinom k d).+1) → 'nat^d.+1 :=
  match d with
    | 0 ⇒ fun _ _ ⇒ k
    | S dd ⇒ castF _ (concatnF (fun i:'I_k.+1 ⇒ Slice_fun (k−i) (CSdk dd i)))
    end.
```

In this definition, the `CSdk` function computes $\mathcal{C}_k^{d+1}$, as indicated by the size of this family. Specifically, when `d` $= 0$, `CSdk 0 k` corresponds to $\mathcal{C}_k^1$, and the only family with size 1 and degree `k` is represented by $\{$`k`$\}$. For `d` > 0, `CSdk` is exactly the Equation (8.3). Additionally, `'I_((pbinom d k).+1` represents the finite type of ordinals of bound `pbinom d k` (see Section 2.3).

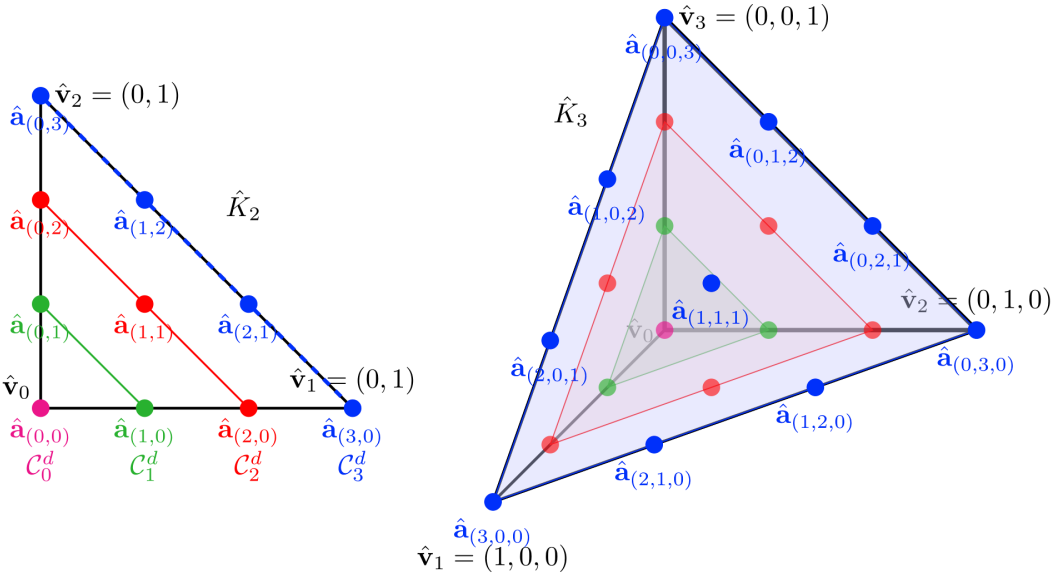

Figure 8.2: Lagrange nodes $\hat{\mathbf{a}}_{\boldsymbol{\alpha}}$ of the reference simplex $\hat{K}_d$ when $d \in \{2,3\}$ and $k = 3$ (see Section 9.1.1). Each node is depicted as a colored ball, and corresponds to a unique element of $\mathcal{A}_3^d$. The colors correspond to degrees $\ell \leqslant 3$ of polynomials, or equivalently to lengths of multi-indices (i.e., in $\mathcal{C}_\ell^d$ for $\ell \leqslant 3$). In pink, the node $\hat{\mathbf{a}}_{\mathbf{0}}$ corresponds to constant polynomials (with degree 0) in $\mathcal{P}_0^d$, and the multi-index 0 in the singleton $\mathcal{C}_0^d$. In green, the nodes correspond to non-constant affine polynomials (with degree 1), and multi-indices $\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_d$ in $\mathcal{C}_1^d$. In red, the nodes correspond to non-affine quadratic polynomials (with degree 2), and multi-indices in $\mathcal{C}_2^d$. In blue, the nodes correspond to non-quadratic cubic polynomials (with degree 3), and multi-indices in $\mathcal{C}_3^d$. We observe in this picture that $\mathcal{A}_3^d = \mathcal{C}_0^d \cup \mathcal{C}_1^d \cup \mathcal{C}_2^d \cup \mathcal{C}_3^d$.

To enhance our understanding of the concepts of layers $\mathcal{C}_k^d$ and slices $\check{\mathcal{S}}_{k,k-i}^d$, consider the following example with $d = 3$ and $k = 3$ (voir Figure 8.1).

$$
\begin{aligned}
\mathcal{C}_3^3 &= \biguplus_{i=0}^{k=3} \check{\mathcal{S}}_{k=3,3-i}^{d=3} = \check{\mathcal{S}}_{k=3,3}^{d=3} \uplus \check{\mathcal{S}}_{k=3,2}^{d=3} \uplus \check{\mathcal{S}}_{k=3,1}^{d=3} \uplus \check{\mathcal{S}}_{k=3,0}^{d=3} \\
&= \{(3,\boldsymbol{\alpha}) \in \mathbb{N}^3 \mid \boldsymbol{\alpha} \in \mathcal{C}_0^2\} \uplus \{(2,\boldsymbol{\alpha}) \in \mathbb{N}^3 \mid \boldsymbol{\alpha} \in \mathcal{C}_1^2\} \uplus \{(1,\boldsymbol{\alpha}) \in \mathbb{N}^3 \mid \boldsymbol{\alpha} \in \mathcal{C}_2^2\} \\
&\quad \uplus \{(0,\boldsymbol{\alpha}) \in \mathbb{N}^3 \mid \boldsymbol{\alpha} \in \mathcal{C}_3^2\} \\
&= \{(3,0,0)\} \uplus \{(2,1,0),(2,0,1)\} \uplus \{(1,2,0),(1,1,1),(1,0,2)\} \\
&\quad \uplus \{(0,3,0),(0,2,1),(0,1,2),(0,0,3)\} \\
&= \{(3,0,0),(2,1,0),(2,0,1),(1,2,0),(1,1,1),(1,0,2),(0,3,0),(0,2,1), \\
&\quad (0,1,2),(0,0,3)\}.
\end{aligned}
$$

The Coq definition of the family of multi-indices $\mathcal{A}_k^d$ is represented as:

```
Definition Adk (d k : nat) : 'I_((pbinom d k).+1) → 'nat^d :=
  match d with
  | 0 ⇒ zero
  | S d ⇒ castF _ (concatnF (fun i:'I_k.+1 ⇒ CSdk d i))
  end.
```

When `d` is zero, the family `Adk 0 k` returns the trivial 0-family. For `d` > 0, it calculates $\mathcal{A}_k^d$ as defined in Equation (8.2) using the `concatnF` function. For an in-depth explanation of the `concatnF` function, refer to Sections 5.3.1. Figure 8.2 provides a visual representation of the organization of these indices.

When $d = 1$, the family $\mathcal{A}_k^1$ includes the one-dimensional multi-indices α, consisting of all natural numbers from 0 to k, inclusive. This family is defined as:

$$\mathcal{A}_k^1 \overset{\text{def.}}{=} \{\alpha \in \mathbb{N} \mid \alpha \leqslant k\} = \{0, 1, 2, \ldots, k\}. \tag{8.4}$$

This can be verified easily using the Equation (8.2) that $\mathcal{A}_k^1$ can be expressed as a disjoint union of $\mathcal{C}_\ell^1$ for ℓ varying from 0 to k. Notably, since $\mathcal{C}_\ell^1 = \{\ell\}$ for each ℓ, the union of these families covers every integer from 0 to k. Therefore, the elements of this union are exactly $\{0, 1, 2, \ldots, k\}$, matching the right-hand side of the Equation 8.4. Furthermore, the total number of elements in $\mathcal{A}_k^1$ is given by:

$$\text{card } \mathcal{A}_k^1 = \binom{k+1}{k} = k + 1.$$

On the other hand, when $k = 1$, $\mathcal{A}_1^d$ includes all d-dimensional multi-indices $\boldsymbol{\alpha}$ where the total sum of all the components of $\boldsymbol{\alpha}$ does not exceed 1. This effectively limits the entries to be mostly zeros, with at most one component being 1, and the rest being 0. The family $\mathcal{A}_1^d$ is defined as:

$$\mathcal{A}_1^d \overset{\text{def.}}{=} \{\boldsymbol{\alpha} \in \mathbb{N}^d \mid \sum_{i=0}^{d-1} \alpha_i \leqslant 1\} = \{\mathbf{0}, \boldsymbol{\delta}_0, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_{d-1}\}. \tag{8.5}$$

The notation $\mathbf{0}$ denotes the zero multi-index $(0, \ldots, 0)$ in d dimensions, and $\boldsymbol{\delta}_i$ is a family of size d that has 1 at the i-th position and 0 elsewhere (see Section 5.3.5). The total number of elements in $\mathcal{A}_1^d$ is given by:

$$\text{card } \mathcal{A}_1^d = \binom{d+1}{1} = d + 1.$$

In Coq, a lemma related to this family is captured as follows:

```
Lemma Ad1_eq : ∀d, Adk d 1 = castF _ (concatF (singleF (constF d 0)) (itemF d 1)).
```

All functions referenced in this lemma, including `castF`, `concatF`, `singleF`, `constF`, are discussed in detail in Section 5.2.3, while `itemF` is elaborated in Section 5.3.1. The proof of this lemma uses the fact that $\mathcal{A}_1^d = \mathcal{C}_0^d \uplus \mathcal{C}_1^d$, where $\mathcal{C}_0^d$ is the family of all d-dimensional multi-indices where all components sum to zero. Practically, this family contains only the zero multi-index $(0, 0, ..., 0)$ in d dimensions. Meanwhile, $\mathcal{C}_1^d$ comprises all d-dimensional multi-indices with components summing to one, including all standard basis vectors $(1, 0, ..., 0), (0, 1, ..., 0), ..., (0, 0, ..., 1)$.

We observe that, regardless of the parameter `k`, the output of `Adk d k` when applied to the smallest index, `ord0`, always yields the zero multi-index of length `d` (refer to Section 2.3 for more detail about the ordinals).

```
Lemma Adk_0 : ∀d k, Adk d k ord0 = zero.
```

The proof of this lemma begins by unfolding the definition of $\mathcal{A}_k^d$ as a concatenated function made up of the layers $\mathcal{C}_k^d$, as detailed in Equation (8.2). This is followed by a case analysis based on the dimension d. In the case where $d = 0$, the family of multi-indices `Adk` is the vector zero of size d. Conversely, when $d \neq 0$, the lemma `concatnF_splitn_ord`, referenced in Section 5.3.1, is used. Employing this lemma, the function $\mathcal{A}_k^d$ simplifies to $\mathcal{A}_k^d[0] = \mathcal{C}_0^d[0] = (0, \dots, 0)$. Here, $\mathcal{C}_0^d[0]$ signifies the first item within the family $\mathcal{A}_k^d$.

Another significant result is that there exists a bijection between the family of multi-indices $\mathcal{A}_k^{d-1}$ and the layer $\mathcal{C}_k^d$, by the function $f_{k,0}^d$ (see Figure 8.3). This mapping is explicitly defined as:

$$\begin{array}{rccc} f_{k,0}^d & : & \mathcal{A}_k^{d-1} & \to & \mathcal{C}_k^d \\ & & \boldsymbol{\alpha} & \longmapsto & (k - |\boldsymbol{\alpha}|, \boldsymbol{\alpha}) \end{array} \tag{8.6}$$

Specifically, this bijection implies that the cardinality of the two families is equal:

$$\mathrm{card}(\mathcal{C}_k^d) = \mathrm{card}(\mathcal{A}_k^{d-1}).$$

We formalize the function $f_{k,0}^{d_1+1}$ in Coq using the concatenation function `Slice_op` defined above.

```
Definition T_node_face0 (d1 k : nat) (ipk : 'I_(pbinom d1 k).+1) :=
  Slice_op (k − sum (Adk d1 k ipk)) (Adk d1 k ipk).
```

We demonstrate that the sum of the sequence produced by `T_node_face0` equals the degree k.

```
Lemma T_node_face0_sum_eq : ∀ipk : 'I_(pbinom d1 k).+1, sum (T_node_face0 ipk) = k.
```

The proof follows easily from the previous lemma `Slice_op_sum`.

8.1.2 Ordering Multi-Indices

The family of multi-indices $\mathcal{A}_k^d$, which represents variable powers in multivariate polynomials, is organized in an increasing order, as depicted in Figure 8.3. To establish this ordering, the `grsymlex_lt` function is used in Coq. This order is mathematically defined for all $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{N}^d$ as follows:

$$\boldsymbol{\alpha} <^{\mathrm{grsymlex}} \boldsymbol{\beta} \overset{\mathrm{def.}}{=} \begin{cases} |\boldsymbol{\alpha}| < |\boldsymbol{\beta}|, & \text{or} \\ |\boldsymbol{\alpha}| = |\boldsymbol{\beta}| \ \wedge \ \boldsymbol{\beta} <^{\mathrm{lex}} \boldsymbol{\alpha}. \end{cases} \tag{8.7}$$

Here, $|\boldsymbol{\alpha}|$ and $|\boldsymbol{\beta}|$ denote the sum of the components of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, respectively. The ordering $<^{\mathrm{lex}}$ is the lexicographic order, similar to the dictionary order. In the expression $\boldsymbol{\beta} <^{\mathrm{lex}} \boldsymbol{\alpha}$, we intentionally switch the order of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ so that the ordering aligns with the well-ordering of the multi-indices $(k\boldsymbol{\delta}_i)_{i\in[1..d]}$ (which are associated to the reference vertices $\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \dots, \hat{\mathbf{v}}_d$) (see Figure 8.3 and Section 9.1.1).

The `grsymlex_lt` function is a *strict total monomial order*, which means that the relation defined by `grsymlex_lt` is total, irreflexive, and transitive. These properties are formalized in the following lemmas:

```
Lemma grsymlex_lt_total_strict : ∀{n} (x y:'nat^n),
  x ≠ y → grsymlex_lt x y ∨ grsymlex_lt y x.
Lemma grsymlex_lt_irrefl : ∀{n} (x:'nat^n), ¬ grsymlex_lt x x.
Lemma grsymlex_lt_trans : ∀{n} (x y z:'nat^n),
  grsymlex_lt x y → grsymlex_lt y z → grsymlex_lt x z.
```

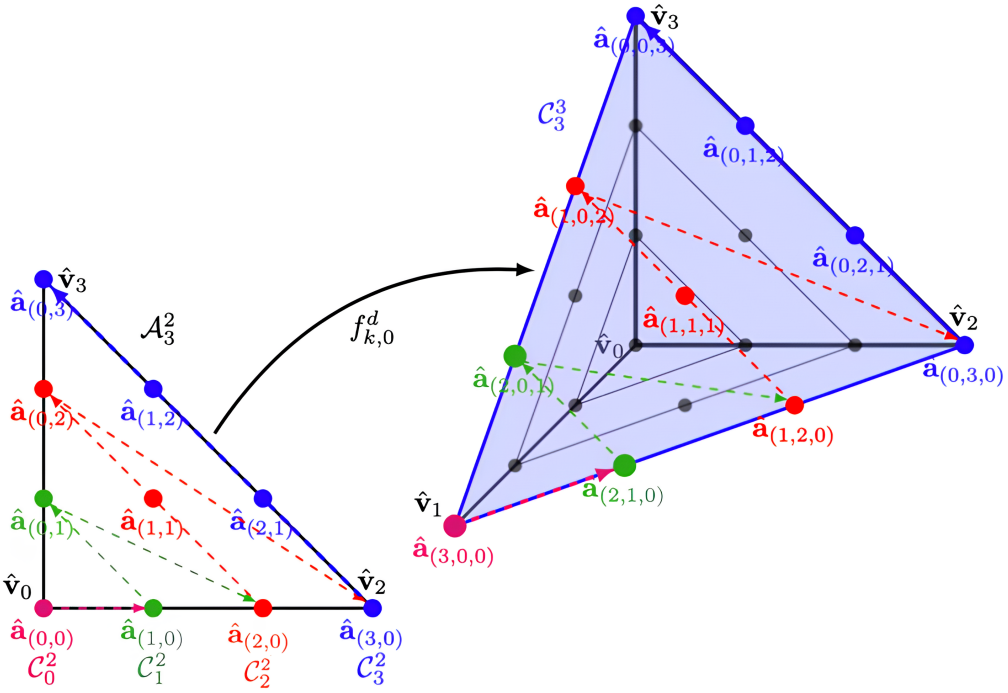

Figure 8.3: This figure illustrates the cases for $d \in \{2, 3\}$ and $k = 3$. The multi-indices in $\mathcal{A}_3^2$ are mapped to those of $\mathcal{C}_3^3$ via the mapping $f_{k,0}^d$ (see Equation (8.6) in Section 8.1.1). This mapping is depicted geometrically, showing how the reference triangle nodes correspond to the nodes on the blue face of the tetrahedron. Notably, this face, opposite vertex $\hat{\mathbf{v}}_0$, contains nodes indexed by $\mathcal{C}_k^d$. The node coloring helps in visualizing the mapping: for every $(\alpha_0, \alpha_1) \in \mathcal{A}_3^2$, the mapping is defined as $f_{3,0}^2(\alpha_0, \alpha_1) = (3 - (\alpha_0 + \alpha_1), \alpha_0, \alpha_1)$. For example, for $(\alpha_0, \alpha_1) = (0, 0)$ we have $f_{3,0}^2(0, 0) = (3 - 0, (0, 0)) = (3, 0, 0)$. The figure also indicates the order of multi-indices $\mathcal{A}_3^d$ under the monomial order `grsymlex_lt`, represented by dashed arrows. For example, in $\mathcal{A}_3^3$, we have $(3, 0, 0) < (2, 1, 0) < (2, 0, 1) < (1, 2, 0) < (1, 1, 1) < (1, 0, 2) < (0, 3, 0) < (0, 2, 1) < (0, 1, 2) < (0, 0, 3)$.

An equivalent and recursive definition of the function `grsymlex_lt` compares first the sum of the elements in the multi-index `x` and that in the multi-index `y`, each of length `d`. If the sums are equal, it compares the remaining elements after omitting the first. This comparison uses the `skipF` function to reduce the vector size (refer to Section 5.2.3).

```
Lemma grsymlex_lt_S : ∀{n : nat} (x y : 'nat^n.+1), grsymlex_lt x y ↔
  sum x < sum y ∨ sum x = sum y ∧ grsymlex_lt (skipF x ord0) (skipF y ord0).
```

We verify that both families of multi-indices, $\mathcal{A}_k^d$ and $\mathcal{C}_k^d$, are sorted according to the monomial order `grsymlex_lt`, using the `sortedF` function defined in Section 5.2.3. For the family $\mathcal{C}_k^d$, we establish the following lemma:

```
Lemma CSdk_grsymlex_lt : ∀d k : nat, sortedF grsymlex_lt (CSdk d k).
```

The proof is based on induction on d. In the base case where $d = 0$, $\mathcal{C}_k^{d+1}$ returns a singleton, making the family trivially sorted. For the inductive case ($d > 0$), the proof uses several properties, including the lemma `grsymlex_lt_trans`, which ensures the transitivity of the monomial order. Additionally, from Equation (8.3), the family $\mathcal{C}_k^d$ is expressed as a disjoint union of slices $\check{\mathcal{S}}_{k,k-i}^d$. Since $\boldsymbol{\alpha}_i \in \mathcal{C}_i^{d-1}$ is sorted according to the monomial order `grsymlex_lt` by the induction hypothesis, applying the `Slice_fun` function preserves this sorting, leading to the conclusion that the entire multi-index family $\mathcal{C}_k^d$ is fully sorted under the monomial order.

The family $\mathcal{A}_k^d$ is also sorted, and the proof follows a similar path, as specified by the following lemma:

```
Lemma Adk_grsymlex_lt : ∀d k : nat, sortedF grsymlex_lt (Adk d k).
```

The proof of this lemma employs pattern matching on the variable `d` to divide the argument in two distinct cases: When `d` $= 0$, the family $\mathcal{A}_k^0$ contains a single element, which is trivially sorted, due to the absence of other elements to compare against. As for the inductive step `d` > 0, according to the definition in Equation (8.2), the family $\mathcal{A}_k^d$ is a concatenation of the family of layers $\mathcal{C}_\ell^d$ for each $\ell \in [0..k]$. Given that the monomial order `grsymlex_lt` is transitive, as established by the lemma `grsymlex_lt_trans`, and considering that each family $\mathcal{C}_\ell^d$ is sorted according to this order, as demonstrated by the previous lemma `CSdk_grsymlex_lt`, coupled with the fact that the last item of $\mathcal{C}_\ell^d$ is smaller than the first item of $\mathcal{C}_{\ell+1}^d$, it follows that the entire concatenated family $\mathcal{A}_k^d$ retains its sorted order.

To illustrate this lemma and its proof with a simple example, consider the case where $d = 2$ and $k = 3$, as depicted in Figure 8.3. Here, $\mathcal{A}_3^2$ denotes the family of all multi-indices (α_0, α_1) for two variables, where the total degree does not exceed 3 (i.e., $|\boldsymbol{\alpha}| = \alpha_0 + \alpha_1 \leqslant 3$). We will examine how these multi-indices are ordered according to `grsymlex_lt` and verify the claim of the lemma `Adk_grsymlex_lt` that `Adk` is sorted. The multi-indices, grouped by their total degrees, include,

- $|\boldsymbol{\alpha}| = 0 :\ \mathcal{C}_0^2 = \{(0,0)\}$.
- $|\boldsymbol{\alpha}| = 1 :\ \mathcal{C}_1^2 = \{(1,0),(0,1)\}$.
- $|\boldsymbol{\alpha}| = 2 :\ \mathcal{C}_2^2 = \{(2,0),(1,1),(0,2)\}$.
- $|\boldsymbol{\alpha}| = 3 :\ \mathcal{C}_3^2 = \{(3,0),(2,1),(1,2),(0,3)\}$.

The `grsymlex_lt` order sorts multi-indices primarily by their sums, and secondarily by iterating after skipping the first component. Therefore, the $\mathcal{C}_\ell^2$ are sorted, and the last of $\mathcal{C}_\ell^2$ is smaller than the first of $\mathcal{C}_{\ell+1}^2$ for $\ell \in [0..2]$. The expected sorted sequence of these indices is as follows:

$$\mathcal{A}_3^2 = \{(0,0),(1,0),(0,1),(2,0),(1,1),(0,2),(3,0),(2,1),(1,2),(0,3)\}.$$

8.1.3 Bijectivity of $\mathcal{A}_k^d$

The function $\mathcal{A}_k^d$ maps indices from the finite set `'I_((pbinom d k).+1)` to vectors of natural numbers `'nat^d` with the condition $|\boldsymbol{\alpha}| \leqslant k$. The bijectivity of $\mathcal{A}_k^d$, derived from its injective and surjective properties, enables the definition of its inverse function.

We begin by establishing the surjectivity of the $\mathcal{A}_k^d$ function, which ensures that for every possible vector of natural number in dimension `d` whose elements sum to at most `k`, there exists some index `ipk` in `'I_((pbinom d k).+1)` that will produce it.

```
Lemma Adk_surj : ∀d k (b:'nat^d), sum b ⩽ k → { ipk | b = Adk d k ipk }.
```

The proof of this lemma follows directly from the surjectivity of $\mathcal{C}_k^d$. It requires the application of various properties related to the slices $\check{\mathcal{S}}_{k,k-i}^d$ and their concatenation (see Section 5.3.1).

Next, we establish the injectivity of the $\mathcal{A}_k^d$ function, ensuring that every different input index `ipk` in `'I_((pbinom d k).+1)` produces a unique vector in `'nat^d`, meaning no two different multi-indices will have the same vector output.

```
Lemma Adk_inj : ∀d k : nat, injective (Adk d k).
```

To prove this property, we apply the `sortedF_inj` lemma, which states that a function is injective if it is sorted according to an irreflexive binary relation. In this case, the binary relation in question is the irreflexive monomial order `grsymlex_lt` as established by the lemma `grsymlex_lt_irrefl`, discussed in Section 8.1.2. Given that the family $\mathcal{A}_k^d$ has been shown to be sorted according to `grsymlex_lt` by the lemma `Adk_grsymlex_lt`, we can deduce the injectivity of $\mathcal{A}_k^d$.

Similarly, the injectivity of the function $\mathcal{C}_k^d$ follows the same reasoning as $\mathcal{A}_k^d$, using the lemma `CSdk_grsymlex_lt` to prove that $\mathcal{C}_k^d$ is sorted under `grsymlex_lt`, thus ensuring its injectivity. This is formalized as:

```
Lemma CSdk_inj: ∀ d k : nat, injective (CSdk d k).
```

Given that $\mathcal{A}_k^d$ is both injective and surjective between the finite set `'I_((pbinom d k).+1)` and the family of natural numbers in `'nat^d` with the condition $|\boldsymbol{\alpha}| \leqslant k$, we define its inverse function as follows:

```
Definition Adk_inv (d k : nat) (b : 'nat^d) : 'I_((pbinom d k).+1) :=
  match (le_dec (sum b) k) with
    | left H ⇒ proj1_sig (Adk_surj d k b H)
    | right _ ⇒ ord0
  end.
```

This function outputs an inverse index within a finite set `'I_((pbinom d k).+1)`. It uses a conditional statement to check whether the sum of the elements in vector `b` is less than or equal to `k`, and if the sum exceeds `k`, the function defaults to `ord0`, indicating that no valid index corresponds to `b` under the $\mathcal{A}_k^d$ mapping.

8.2 $\mathcal{P}_k^d$ Polynomial Space

In this section, we explore the polynomial space $\mathcal{P}_k^d$ (see [36, p.78], [59, p.108]), which consists of a d-variate polynomials, with total degree not exceeding k. A basis of this space is constructed using the family of multi-indices $\mathcal{A}_k^d$ defined in Section 8.1. Within the framework of this thesis, we will recognize that the polynomial space $\mathcal{P}_k^d$ is characterized by multiple bases for special values of d and k. In this section, we introduce the monomial basis, and in subsequent chapters, we will explore the Lagrange polynomial basis. In the upcoming sections, we will focus on specific examples within this space, particularly for the cases where $k = 1$ and $d = 1$, detailed in Sections 8.3 and 8.4, respectively.

8.2.1 Definition of the Polynomial Space $\mathcal{P}_k^d$ and its Basis $\mathcal{B}^{d,k}$

The polynomial space $\mathcal{P}_k^d$ is defined as follows:

$$\mathcal{P}_k^d \overset{\text{def.}}{=} \text{span}\,(\boldsymbol{X}^{\boldsymbol{\alpha}})_{\boldsymbol{\alpha}\in\mathcal{A}_k^d} = \left\{ \left(\mathbf{x} \mapsto \sum_{\boldsymbol{\alpha}\in\mathcal{A}_k^d} c_{\boldsymbol{\alpha}} \prod_{i=0}^{d-1} x_i^{\alpha_i} \right) : \mathbb{R}^d \to \mathbb{R} \;\middle|\; (c_{\boldsymbol{\alpha}})_{\boldsymbol{\alpha}\in\mathcal{A}_k^d} \in \mathbb{R} \right\}. \tag{8.8}$$

where each monomial is represented as:

$$\boldsymbol{X}^{\boldsymbol{\alpha}} \overset{\text{def.}}{=} \left(\mathbf{x} \in \mathbb{R}^d \longmapsto \prod_{i=0}^{d-1} x_i^{\alpha_i} \in \mathbb{R} \right). \tag{8.9}$$

Each polynomial in the space $\mathcal{P}_k^d$ is expressed as a linear combination of terms. Each term is a product of the variables $x_i \in \mathbb{R}$ raised to some power α_i, where the multi-index $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_{d-1})$ varies across the family $\mathcal{A}_k^d$. This family includes all tuples of non-negative integers such that their sum does not exceed k (see Section 8.1). Additionally, the coefficients $c_{\boldsymbol{\alpha}} \in \mathbb{R}$ for each $\boldsymbol{\alpha} \in \mathcal{A}_k^d$ are the scalars that weight each monomial in this linear combination. For more detailed information about monomials, refer to Section 5.3.2. The monomials $\boldsymbol{X}^{\boldsymbol{\alpha}}$ form a family within the polynomial space $\mathcal{P}_k^d$, which we denote as $\mathcal{B}_{\boldsymbol{\alpha}}^{d,k}$, to make the dependence on d and k explicit. The family $\mathcal{B}^{d,k}$ includes all possible monomials formed by taking each combination of exponents in $\mathcal{A}_k^d$.

Let us consider `FRd d` to be the vector space of functions mapping from the real space $\mathbb{R}^d$ to $\mathbb{R}$. The Coq implementation of $\mathcal{B}^{d,k}$ is provided as follows:

```
Definition BasisPdk d k : '(FRd d)^((pbinom d k).+1) :=
  fun (ipk : 'I_((pbinom d k).+1)) (x : 'R^d) ⇒ powF_P (Adk d k ipk) x.
```

Here, `powF_P` represents the product of powers of variables, where each variable x_i from the vector $\mathbf{x} \in \mathbb{R}^d$ is raised to a corresponding power (see Section 5.3.2). This family serves as a collection of functions spanning the entire space $\mathcal{P}_k^d$, allowing any polynomial in this space to be represented as a linear combination of the monomials of this family. This also means that $\mathcal{B}^{d,k}$ is a generating family for $\mathcal{P}_k^d$.

In the following Coq snippet, the space $\mathcal{P}_k^d$ is defined as the linear span of the family provided by $\mathcal{B}^{d,k}$ (see Section 5.4.1).

```
Definition Pdk d k : FRd d → Prop := lin_span (BasisPdk d k).
```

Since every linear span in a module space over a ring (such as the real numbers $\mathbb{R}$) satisfies the properties of a module space, we conclude that the $\mathcal{P}_k^d$ is closed under the linear operations.

```
Lemma Pdk_cms : ∀d k, compatible_ms (Pdk d k).
```

Here, `compatible_ms` means that the space $\mathcal{P}_k^d$ qualifies as a subgroup of the space of functions `FRd` and is closed under scalar multiplication, as detailed in Section 5.3.3.

Moreover, the polynomial space $\mathcal{P}_k^d$ is closed under linear combinations. This closure is presented in the following lemma:

```
Lemma Pdk_lc : ∀ {d} {n} k (p : '( FRd d)^n), (∀ i : 'I_n, Pdk d k (p i)) →
  ∀ (L:' R^n), Pdk d k (lin_comb L p).
```

This lemma is directly proved by the lemma `lin_span_lc_closed` introduced in Section 5.4.1, which ensures that linear combinations of functions `p` within the polynomial space $\mathcal{P}_k^d$ remain within it, as $\mathcal{P}_k^d$ is a linear span of the family $\mathcal{B}^{d,k}$.

8.2.2 Linear Independence of the Family $\mathcal{B}^{d,k}$

The family $\mathcal{B}^{d,k}$ consists of monomials $\boldsymbol{X}^{\boldsymbol{\alpha}}$ that generate the polynomial space $\mathcal{P}_k^d$ for all $\boldsymbol{\alpha} \in \mathcal{A}_k^d$. To prove that $\mathcal{B}^{d,k}$ forms a basis of the space $\mathcal{P}_k^d$, we must demonstrate its linear independence. This is stated by the following lemma:

```
Lemma BasisPdk_lin_indep: lin_indep (BasisPdk d k).
```

To demonstrate the linear independence of $\mathcal{B}^{d,k}$, we begin by considering a zero linear combination of the monomials $\boldsymbol{X}^{\boldsymbol{\alpha}}$ with coefficents $c_{\boldsymbol{\alpha}}$, and prove that $c_{\boldsymbol{\alpha}}$ are zero. For further details on linear independence, see Section 5.4.2. Let us assume that we have $c_{\boldsymbol{\alpha}} \in \mathbb{R}$ such that,

$$\sum_{\boldsymbol{\alpha}\in\mathcal{A}_k^d} c_{\boldsymbol{\alpha}} \boldsymbol{X}^{\boldsymbol{\alpha}} = 0. \tag{8.10}$$

We want to prove that $\boldsymbol{c}$ is the zero family. Let us consider $\boldsymbol{\beta} \in \mathcal{A}_k^d$, and prove that $c_{\boldsymbol{\beta}} = 0$:

We begin by applying the partial derivative operator $\partial^{\boldsymbol{\beta}}$ to both sides of Equation (8.10), where each β_i represents the order of derivation with respect to the variable x_i. The partial derivative operator is defined as:

$$\partial^{\boldsymbol{\beta}} \stackrel{\text{def.}}{=} \frac{\partial^{|\boldsymbol{\beta}|}}{\partial x_0^{\beta_0} \partial x_1^{\beta_1} ... \partial x_{d-1}^{\beta_{d-1}}},$$

where $|\boldsymbol{\beta}| = \beta_0 + \beta_1 + \ldots + \beta_{d-1}$ is the total order of the derivative. In Coquelicot, the derivative operator is defined, but it is limited to polynomials for the sake of simplicity.

Now, using the linearity of partial derivatives, we obtain:

$$\partial^{\boldsymbol{\beta}} \left(\sum_{\boldsymbol{\alpha}\in\mathcal{A}_k^d} c_{\boldsymbol{\alpha}} \boldsymbol{X}^{\boldsymbol{\alpha}} \right) = \sum_{\boldsymbol{\alpha}\in\mathcal{A}_k^d} c_{\boldsymbol{\alpha}}\ \partial^{\boldsymbol{\beta}} \boldsymbol{X}^{\boldsymbol{\alpha}} = \partial^{\boldsymbol{\beta}} 0 = 0. \tag{8.11}$$

By evaluating the equality (8.11) between functions at 0 in $\mathbb{R}^d$, we obtain the following identity:

$$\sum_{\boldsymbol{\alpha}\in\mathcal{A}_k^d} c_{\boldsymbol{\alpha}}\ \partial^{\boldsymbol{\beta}} \boldsymbol{X}^{\boldsymbol{\alpha}}(0) = 0.$$

We then proceed by studying cases based on the respective values of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, leading to the decomposition:

$$c_{\boldsymbol{\beta}}\ \partial^{\boldsymbol{\beta}} \boldsymbol{X}^{\boldsymbol{\beta}}(0) + \sum_{\substack{\boldsymbol{\alpha}\in\mathcal{A}_k^d \\ \forall i\in[0..d-1], \boldsymbol{\alpha}_i < \boldsymbol{\beta}_i}} c_{\boldsymbol{\alpha}}\ \partial^{\boldsymbol{\beta}} \boldsymbol{X}^{\boldsymbol{\alpha}}(0) + \sum_{\substack{\boldsymbol{\alpha}\in\mathcal{A}_k^d \\ \exists i\in[0..d-1], \boldsymbol{\alpha}_i > \boldsymbol{\beta}_i}} c_{\boldsymbol{\alpha}}\ \partial^{\boldsymbol{\beta}} \boldsymbol{X}^{\boldsymbol{\alpha}}(0) = 0. \tag{8.12}$$

Each component of this sum will now be examined in detail.

(i) first term of the sum: case $\boldsymbol{\alpha} = \boldsymbol{\beta}$.
This case involves differentiating the monomial $\boldsymbol{X}^{\boldsymbol{\beta}}$ with respect to the same $\boldsymbol{\beta}$. According to the differentiation rules for monomials, we have:

$$\partial^{\boldsymbol{\beta}} \boldsymbol{X}^{\boldsymbol{\beta}} = \boldsymbol{\beta}! \neq 0, \tag{8.13}$$

where $\boldsymbol{\beta}! = \prod_{i=0}^{d-1} \beta_i$ refers to the factorial of the multi-index $\boldsymbol{\beta}$. The assertion (8.13) is supported by the following lemma:

```
Lemma BasisPdk_lin_indep_aux1 : ∀d k beta : 'nat^d) (i : 'I_(pbinom d k).+1),
  beta = Adk d k i → DeriveF beta (BasisPdk d k i) zero ≠ zero.
```

Given this lemma, we can simplify Equation (8.12) to:

$$c_{\boldsymbol{\beta}}\ \boldsymbol{\beta}! + \sum_{\substack{\boldsymbol{\alpha}\in\mathcal{A}_k^d \\ \forall i\in[0..d-1], \boldsymbol{\alpha}_i < \boldsymbol{\beta}_i}} c_{\boldsymbol{\alpha}}\ \partial^{\boldsymbol{\beta}} \boldsymbol{X}^{\boldsymbol{\alpha}}(0) + \sum_{\substack{\boldsymbol{\alpha}\in\mathcal{A}_k^d \\ \exists i\in[0..d-1], \boldsymbol{\alpha}_i > \boldsymbol{\beta}_i}} c_{\boldsymbol{\alpha}}\ \partial^{\boldsymbol{\beta}} \boldsymbol{X}^{\boldsymbol{\alpha}}(0) = 0. \tag{8.14}$$

(ii) second term of the sum: case $\forall i \in [0..d-1], \boldsymbol{\alpha}_i < \boldsymbol{\beta}_i$.
In this case, the derivative $\partial^{\boldsymbol{\beta}} \boldsymbol{X}^{\boldsymbol{\alpha}}$ involves differentiating the term $x_i^{\alpha_i}$ more times than its power α_i. This results in the derivative being zero. The following lemma establishes this result:

```
Lemma BasisPdk_lin_indep_aux2 : ∀d k beta : 'nat^d) (i : 'I_(pbinom d k).+1),
  beta ≠ Adk d k i → DeriveF alpha (BasisPdk d k i) zero = zero.
```

Therefore, the sum of these derivatives evaluated at zero also equals zero, due to the properties of differentiation:

$$\sum_{\substack{\boldsymbol{\alpha} \in \mathcal{A}_k^d \\ \forall i \in [0..d-1], \boldsymbol{\alpha}_i < \boldsymbol{\beta}_i}} c_{\boldsymbol{\alpha}} \partial^{\boldsymbol{\beta}} \boldsymbol{X}^{\boldsymbol{\alpha}}(0) = 0.$$

Consequently, Equation (8.14) can be further simplified to:

$$c_{\boldsymbol{\beta}}\ \boldsymbol{\beta}! + 0 + \sum_{\substack{\boldsymbol{\alpha} \in \mathcal{A}_k^d \\ \exists i \in [0..d-1], \boldsymbol{\alpha}_i > \boldsymbol{\beta}_i}} c_{\boldsymbol{\alpha}}\ \partial^{\boldsymbol{\beta}} \boldsymbol{X}^{\boldsymbol{\alpha}}(0) = 0. \tag{8.15}$$

(iii) third term of the sum: case $\exists i \in [0..d-1], \boldsymbol{\alpha}_i > \boldsymbol{\beta}_i$.
In this case, the derivative $\partial^{\boldsymbol{\beta}} \boldsymbol{X}^{\boldsymbol{\alpha}}$ involves differentiating the term $x_i^{\alpha_i}$ fewer times than its power α_i. As a result, evaluating these derivatives at zero yields zero for any exponent $\alpha_i > 0$. This is supported by the lemma `BasisPdk_lin_indep_aux2`. Consequently, Equation (8.15) becomes:

$$c_{\boldsymbol{\beta}}\ \boldsymbol{\beta}! + 0 + 0 = 0. \tag{8.16}$$

This leads to the conclusion that $c_{\boldsymbol{\beta}} = 0$ for all $\boldsymbol{\beta} \in \mathcal{A}_k^d$. Therefore, we have established the linear independence of the family $\mathcal{B}^{d,k}$.
□

Consequently, the family $\mathcal{B}^{d,k}$ constitutes a basis for the vector space $\mathcal{P}_k^d$, as demonstrated by the following lemma:

```
Lemma BasisPdk_basis : basis (Pdk d k) (BasisPdk d k).
```

The proof is a direct application of the lemma `basis_lin_span_equiv`, established in Section 5.4.2. This lemma states that since the family $\mathcal{B}^{d,k}$ is linearly independent by the preceding lemma `BasisPdk_lin_indep`, it necessarily forms a basis for the linear span of $\mathcal{B}^{d,k}$.

Therefore, the dimension of $\mathcal{P}_k^d$ is equal to the cardinality of $\mathcal{B}^{d,k}$, which aligns with the cardinality of $\mathcal{A}_k^d$. The specific dimension of this space is established by the following equation:

$$\dim \mathcal{P}_k^d = \binom{d+k}{k} = \frac{(d+k)!}{d!\ k!}. \tag{8.17}$$

This calculated dimension is supported through this lemma,

```
Lemma Pdk_has_dim : has_dim (Pdk d k) ((pbinom d k).+1).
```

8.2.3 Overview of Polynomial Space $\mathcal{P}_k^d$ Properties

The following section highlights a series of fundamental properties of the polynomial spaces $\mathcal{P}_k^d$, which are essential for developing results related to Lagrange polynomials through linear algebraic operations in the upcoming chapters.

Decomposition of Polynomial Spaces

We introduce a structured method for decomposing a polynomial in higher-dimensional spaces into components from polynomial spaces in lower dimensions or of lower degree. Specifically, for any polynomial $p \in \mathcal{P}_{k+1}^{d+1}$, there exist two unique polynomials, $p_0 \in \mathcal{P}_{k+1}^d$ and $p_1 \in \mathcal{P}_k^{d+1}$ that we build in Coq, such that:

$$\forall (x_0, \ldots, x_d) \in \mathbb{R}^{d+1}, \quad p(x_0, \ldots, x_d) = p_0(x_0, \ldots, x_{d-1}) + x_d \; p_1(x_0, \ldots, x_d). \tag{8.18}$$

Polynomial Multiplication in Polynomial Spaces and Monotonicity

Furthermore, for any given polynomial degrees $k, \ell \in \mathbb{N}$, and dimension d, we consider two polynomials $p \in \mathcal{P}_k^d$ and $q \in \mathcal{P}_\ell^d$. The product pq then resides in $\mathcal{P}_{k+\ell}^d$. This property extends to finite products of polynomials. Specifically, assuming each polynomial function p_i belongs to $\mathcal{P}_{\alpha_i}^d$ for all $i \in \{0, \ldots, n-1\}$, i.e., each p_i is a polynomial in d variables with a total degree at most α_i, such that $\sum_{i=0}^{n-1} \alpha_i \leqslant k$, then the product of these polynomials is

$$\prod_{i=0}^{n-1} p_i \in \mathcal{P}_k^d. \tag{8.19}$$

We further address the principle of monotonicity within the polynomial spaces $\mathcal{P}_k^d$. This concept highlights that polynomial spaces of higher degrees contain all polynomials from their lower-degree counterparts. Mathematically, this is articulated by stating that if $k_1 \leqslant k_2$, and a polynomial $p \in \mathcal{P}_{k_1}^d$, then $p \in \mathcal{P}_{k_2}^d$.

Composing Polynomials with Affine Maps

Let us now establish and verify the compatibility of the polynomial spaces $\mathcal{P}_k^d$ with the operation of composing polynomials with affine maps (see Section 5.3.6). This is formalized in the following lemma:

```
Lemma Pdk_compose_am : ∀{n d} k (p : FRd d) (f : 'R^n → 'R^d),
  aff_map f → (Pdk d k) p → (Pdk n k) (compose p f).
```

Consider an affine map $f : \mathbb{R}^n \to \mathbb{R}^d$, defined as $f(\mathbf{x}) = (f_i(\mathbf{x}))_{i \in [0..d-1]}$. For further details on affine mappings, refer to Section 5.3.6. Given that p is a polynomial in $\mathcal{P}_k^d$, it can be expressed as a linear combination of basis polynomials $\mathcal{B}^{d,k}$, such that $p = \sum_{\boldsymbol{\alpha} \in \mathcal{A}_k^d} L_{\boldsymbol{\alpha}} \mathcal{B}_{\boldsymbol{\alpha}}^{d,k}$ with $\boldsymbol{L} \in \mathbb{R}^{\mathrm{card}\, \mathcal{A}_k^d}$. To verify this lemma, we must demonstrate that:

$$p \circ f = \left(\sum_{\boldsymbol{\alpha} \in \mathcal{A}_k^d} L_{\boldsymbol{\alpha}} \mathcal{B}_{\boldsymbol{\alpha}}^{d,k} \right) \circ f = \sum_{\boldsymbol{\alpha} \in \mathcal{A}_k^d} L_{\boldsymbol{\alpha}} \left(\mathcal{B}_{\boldsymbol{\alpha}}^{d,k} \circ f \right) = \sum_{\boldsymbol{\alpha} \in \mathcal{A}_k^d} L_{\boldsymbol{\alpha}} \left(\mathbf{x} \longmapsto \prod_{i=0}^{d-1} f_i^{\alpha_i}(\mathbf{x}) \right) \in \mathcal{P}_k^n.$$

Given that f is affine, each power function $f_i^{\alpha_i}$ resides within $\mathcal{P}_{\alpha_i}^n$. Therefore, the product of these mappings, $\prod_{i=0}^{d-1} f_i^{\alpha_i}$, stays within $\mathcal{P}_k^n$, as addressed by Equation (8.19). Applying Lemma

`Pdk_lc`, which is discussed in Section 8.2.1, we confirm that the composition is compatible with the structure of the polynomial space.
□

The following lemma asserts the reverse case, demonstrating that if composing a polynomial p with a bijective affine map f yields a polynomial in the n-dimensional space $\mathcal{P}_k^n$, then p must necessarily be an element of the d-dimensional space $\mathcal{P}_k^d$.

```
Lemma Pdk_am_rev : ∀{d} k (p : FRd d) (f : 'R^d → 'R^d),
  aff_map f → bijective f → (Pdk d k) (compose p f) → (Pdk d k) p.
```

To validate this lemma, we approach the proof by initially considering the function $(p \circ f) \circ f^{-1}$ instead of directly examining p, using the identity property of bijections. This method is supported by the lemma `f_inv_can_r`, which states that the composition of f with its inverse yields the identity function when f is bijective. According to the lemma `am_bij_compat` discussed in Section 5.3.6, f^{-1} is also an affine map. Thus, since $p \circ f \in \mathcal{P}_k^d$, we conclude, through the previously verified lemma `Pdk_compose_am` that $(p \circ f) \circ f^{-1} \in \mathcal{P}_k^d$. Consequently, $p \in \mathcal{P}_k^d$.
□

Finally, we prove that given a generating family $(B_{\boldsymbol{\alpha}})_{\boldsymbol{\alpha} \in \mathcal{A}_k^d}$ for the polynomial space $\mathcal{P}_k^d$, and an affine bijective function f, then the polynomial space $\mathcal{P}_k^d$ is exactly the linear span of their composition.

```
Lemma Pdk_am_compose_basis : ∀{d} k (B : '(FRd d)^(pbinom d k).+1)
  (f : 'R^d → 'R^d), lin_gen (Pdk d k) B → aff_map f → bijective f →
    Pdk d k = lin_span (fun i ⇒ compose (B i) f).
```

Let us dissect the proof step by step. To demonstrate the equality of these two spaces, we need to prove that each space is a subset of the other. Consider an affine bijective function $f : \mathbb{R}^d \to \mathbb{R}^d$.

(i) Prove $\mathcal{P}_k^d \subset \text{span}\ (B \circ f)$:

Considering a polynomial p in $\mathcal{P}_k^d$, from the lemma `am_bij_compat` as discussed in Section 5.3.6, we deduce that f^{-1} is affine. Applying the lemma `Pdk_compose_am`, we get $p \circ f^{-1}$ also belongs to $\mathcal{P}_k^d$. Assuming B forms a generating family for the polynomial space $\mathcal{P}_k^d$, it follows that $p \circ f^{-1} \in \text{span}\ B$, thereby it can be expressed as a linear combination of basis elements in B:

$$\forall \mathbf{x} \in \mathbb{R}^d, \quad p \circ f^{-1}(\mathbf{x}) = \sum_{\boldsymbol{\alpha} \in \mathcal{A}_k^d} L_{\boldsymbol{\alpha}} B_{\boldsymbol{\alpha}}(\mathbf{x}).$$

Composing both sides with f and applying the lemma `f_inv_can_l` which states the canonical left inverse property of f, we establish that:

$$\forall \mathbf{x} \in \mathbb{R}^d, \quad p(\mathbf{x}) = (p \circ f^{-1} \circ f)(\mathbf{x}) = \sum_{\boldsymbol{\alpha} \in \mathcal{A}_k^d} L_{\boldsymbol{\alpha}} (B_{\boldsymbol{\alpha}} \circ f)(\mathbf{x}).$$

Thus, p is in the span of $B \circ f$, completing the proof that $\mathcal{P}_k^d \subset \text{span}(B \circ f)$.

(ii) Prove $\text{span}\ (B \circ f) \subset \mathcal{P}_k^d$:

Consider any polynomial p within $\text{span}(B \circ f)$. This implies that p can be expressed as follows:

$$\forall \mathbf{x} \in \mathbb{R}^d, \quad p(\mathbf{x}) = \sum_{\boldsymbol{\alpha} \in \mathcal{A}_k^d} L_{\boldsymbol{\alpha}} (B_{\boldsymbol{\alpha}} \circ f)(\mathbf{x}).$$

Since the family B forms a generating family for the space $\mathcal{P}_k^d$, we know from the lemma `lin_gen_inclF` that each term in the family B belongs to $\mathcal{P}_k^d$ (refer to Section 5.4.2). Applying the lemma `Pdk_compose_am`, it follows that the composition $B \circ f$ also lies within $\mathcal{P}_k^d$. Consequently, their linear combination $\sum_{\boldsymbol{\alpha} \in \mathcal{A}_k^d} L_{\boldsymbol{\alpha}}(B_{\boldsymbol{\alpha}} \circ f) \in \mathcal{P}_k^d$, as stated by the lemma `Pdk_lc`. Thus, p belongs to $\mathcal{P}_k^d$, completing the proof that $\text{span}(B \circ f) \subset \mathcal{P}_k^d$.

This lemma will be used for later discussions, particularly for establishing that the $\mathcal{P}_k^d$ spaces for $d = 1$ can be expressed as the linear span of the Lagrange polynomial basis $\mathcal{L}^{1,k}$ as well, as outlined in Section 9.6.

8.3 $\mathcal{P}_1^d$ Polynomial Space

This section focuses on the case $k = 1$ of the space $\mathcal{P}_1^d$, which provides a framework for linear approximations in multiple dimensions. This space consists of all polynomials in d variables with degree at most 1. It can be defined as:

$$\mathcal{P}_1^d = \text{span}(1, x_0, x_1, \ldots, x_{d-1}). \tag{8.20}$$

This space includes affine polynomials, which are the sum of a constant term and linear terms in each of the d variables.

The dimension of $\mathcal{P}_1^d$ is:

$$\dim \mathcal{P}_1^d = \binom{d+1}{1} = d + 1.$$

A basis for $\mathcal{P}_1^d$ consists of all monomials of degree at most 1. Explicitly, it can be written as:

$$\mathcal{B}^{d,1} = (1, x_0, x_1, \ldots, x_{d-1})\,.$$

In this notation, 1 represents the constant polynomial, and x_i denotes the linear terms associated with the variable x_i for $i \in [0..d-1]$. To elaborate further on the construction of the $\mathcal{B}^{d,1}$ family, we reference the following two Coq lemmas.

The first lemma specifies that for any dimension d, the first basis element, indexed by `ord0` (representing the zero index as detailed in Section 2.3), is the constant function 1. This corresponds directly to the constant term within the polynomial basis.

```
Lemma BasisPd1_0 : ∀{d:nat}, BasisPdk d 1 ord0 = fun _ ⇒ 1.
```

The proof of this lemma is derived directly from lemma `Ad1_eq`, detailed in Section 8.1.1. This lemma states that in the polynomial space $\mathcal{P}_1^d$ when the degree $k = 1$, the exponent vector for the constant term within the family $\mathcal{A}_1^d$ consists solely of zero exponents. Thus, the polynomial function represented by this basis element will consistently yield 1 for any input.

The second lemma focuses on the remaining basis elements, specifically the linear terms in each variable x_i. For any index `ipk` other than `ord0` (the zero index), we define $\mathcal{B}^{d,1}$ as $x \mapsto x_{\texttt{ipk}-1}$. This is stated by the following lemma:

```
Lemma BasisPd1_neq0 : ∀{d:nat} (ipk : 'I_(pbinom d 1).+1) (H : (ipk ≠ ord0)),
  BasisPdk d 1 ipk = fun x ⇒ x (lower_S (cast _ _ H)).
```

Here, the function `lower_S` used to adjust the index `ipk` by lowering it within a set of indices of size `d`+1, given that `ipk` is not the zero index (see Section 2.3). The proof of this lemma utilizes

the `Ad1_neq0` lemma, which focuses on cases where the index `ipk` is different from `ord0` (refer to Equation (8.5)). This lemma specifies that for such indices, the exponent family sets one entry to 1 and all others to 0, indicating each term is a single variable raised to the power of 1.

8.4 $\mathcal{P}_k^1$ Polynomial Space

In this section, we address the case $d = 1$ of the polynomial space $\mathcal{P}_k^1$, which consists of all polynomials in one variable up to a specified degree k. Mathematically, $\mathcal{P}_k^1$ is defined as:

$$\mathcal{P}_k^1 = \text{span } (1, x, x^2, \ldots, x^k).$$

The dimension of $\mathcal{P}_k^1$ is:

$$\dim \mathcal{P}_k^1 = \binom{k+1}{k} = k + 1.$$

The basis for $\mathcal{P}_k^1$ includes all monomials from degree 0 up to degree k, represented as the powers of the variable x. Specifically, the basis is defined as:

$$\mathcal{B}^{1,k} = \left\{1, x, x^2, \ldots, x^k\right\}.$$

where 1 represents the constant polynomial, and x^i represents the polynomial term of degree i for $i \in [1..k]$. These monomials collectively form a basis that spans the space $\mathcal{P}_k^1$, enabling the representation of any polynomial up to degree k.

Chapter 9

Reference and Current Finite Elements: Geometry and $\mathcal{P}_k^d$ Lagrange Polynomial Basis

In this chapter, we build current finite elements from the reference element and formalize the Lagrange polynomials and affine transformations. We particularly emphasize their roles in finite element analysis, as elaborated in [36, Section 5.1 p.49] and [55, Section 2.4.3 p.84]. A detailed examination of how these polynomials are formulated and implemented in both reference and current geometrical elements is provided. Additionally, we explore specific cases when the spatial dimension $d = 1$ and the polynomial degree $k = 1$.

We begin with Section 9.1, where we discuss vertices, sub-vertices, nodes, and sub-nodes in both reference and current geometric elements (simplices). We highlight the affine independence property and provide detailed examples to enhance comprehension of these concepts. This section is essential as it lays the groundwork for subsequent sections, where we will demonstrate that the Lagrange polynomials are evaluated at Lagrange nodes to approximate the solution of the system. Additionally, the significance of these nodes and vertices extends further, as they are necessary for illustrating their transformation through an affine map between reference and current geometric elements. Furthermore, sub-vertices and sub-nodes will be required in the next chapter (see Chapter 10) to prove the unisolvence property.

Subsequently, in Section 9.2, we concentrate on the $k = 1$ case of reference Lagrange interpolation polynomials within the reference geometric element, as detailed in [16, Section 3.2 p.72]. We specifically explore the approximation space $\mathcal{P}_1^d$, which includes polynomials of degree at most one within a d-dimensional space. This section also outlines several key properties of these polynomials.

Next, in Section 9.3, we address a specific affine transformation [55, Section 2.4.6 p.87], denoted by T_{geo}^K, which is essential for mapping the reference element onto the current element in physical space. This section further explores how these transformations preserve geometric features such as vertices, nodes, and barycentric coordinates. Additionally, we construct the inverse of T_{geo}^K, thereby enabling the reverse mapping from the current to the reference element.

Building upon this section, we extend our discussion in Section 9.4 to the construction of current simplices finite elements on each simplex in the mesh $\mathcal{T}_h$, from the reference finite element, and the current vertices. The approach involves mapping polynomial functions

and degrees of freedom from the reference to the current elements through the geometric transformations ψ^K and T^K_{geo}.

Following this, in Section 9.5 we address the $k = 1$ case of current Lagrange polynomials in the space $\mathcal{P}^d_1$, for current geometric elements. These polynomials are defined by a composition of the reference polynomials with the inverse affine transformation, denoted $(T^K_{\text{geo}})^{-1}$.

The final Section 9.6, delves into the case when $d = 1$ of Lagrange polynomials, focusing on polynomials of degrees up to k. Both the reference and current polynomials are thoroughly explored, along with their properties.

9.1 Simplicial Geometry

This section presents various geometric features and their interactions within the context of the finite element method (FEM). As elaborated in Chapter 6, FEM simplifies complex geometrical shapes by decomposing them into smaller, more manageable components, known as *geometric elements*. Within the mesh $\mathcal{T}_h$, each element is characterized by *vertices*, which define the shape of the element, and *nodes*, which represent the degrees of freedom where the system of equations that FEM addresses is formulated.

Beware of the notation used for vertices and nodes. Those denoted with a hat (such as $\hat{v}$ and $\hat{a}$) are reference vertices and nodes, while those marked with a check (such as $\check{v}$ and $\check{a}$) represent sub-nodes and sub-vertices of the current elements.

9.1.1 Definition of Reference Vertices and Lagrange Nodes

Defining and manipulating vertices and nodes is fundamental for establishing the geometry and constructing linear forms of the Lagrange finite elements. *Vertices* represent the corner points of a geometric element, defining the physical shape and boundaries. *Nodes*, on the other hand, are specific locations within or on the boundaries of the element where the solution to a system of equations is computed, as illustrated in Figure 9.1. While nodes may coincide with vertices, they can also include additional points along the faces or within the interior of the element. The number of these nodes is dictated by the degree of the polynomial used in the approximation space, which influences the level of precision in the numerical solution of the equations.

Reference Vertices

We introduce the family of reference vertices for the reference simplex $\hat{K}$ in the affine space $\mathbb{R}^d$ (see Section 6.2.2). It is mathematically expressed for all $i \in [0..d]$:

$$\hat{\mathbf{v}}_i \overset{\text{def.}}{=} \begin{cases} \mathbf{0} & \text{if } i = 0, \\ \boldsymbol{\delta}_{i-1} & \text{else.} \end{cases} \tag{9.1}$$

This family consists of $d+1$ reference points in $\mathbb{R}^d$, which define the boundaries of the simplex, and $\mathbf{0}$ is the zero family of length d. This is represented within Coq as follows:

```
Definition vtx_simplex_ref : 'R^d)^(S d) :=
  fun i : 'I_d.+1 ⇒ match (ord_eq_dec i ord0) with
    | left _ ⇒ zero
    | right _ ⇒ kronecker (i − 1)
    end.
```

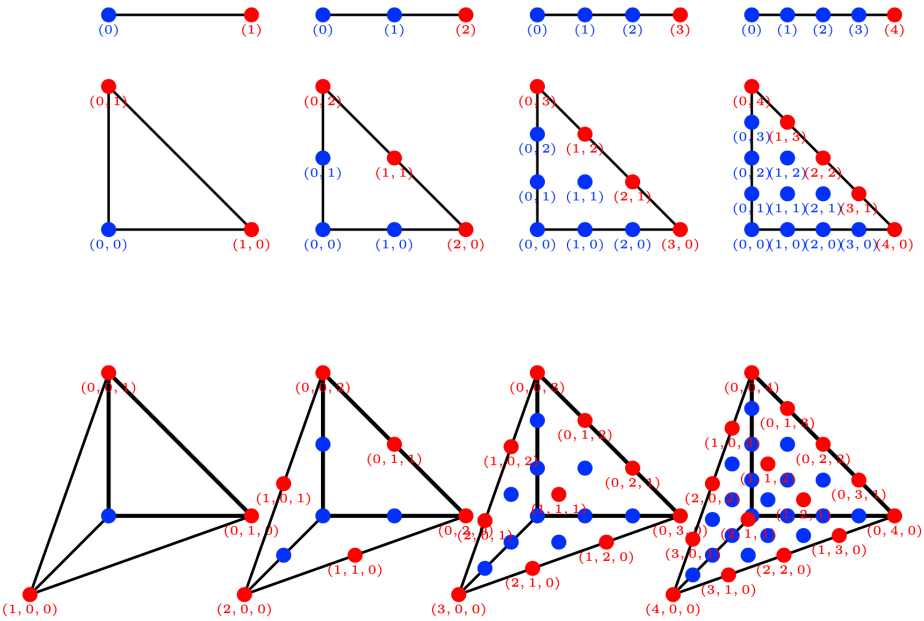

Figure 9.1: Lagrange nodes of the reference simplex for $d = 1, 2, 3$ (rows) and $k = 1, 2, 3, 4$ (columns) are detailed in Section 9.1.1. Nodes represent all the points within these geometric shapes, whereas vertices specifically define the corners. Nodes corresponding to the highest degree are depicted in red, and the others in blue. The multi-indices of the nodes are indicated for all cases when $d = 1$ and $d = 2$, but only for the highest degree when $d = 3$.

This definition specifies a function that assigns a d-dimensional point to each index $i \in [0..d]$ within $\mathbb{R}^d$. Specifically, for the zero index $i = 0$, the function returns the first vertex consisting entirely of zeros. For any non-zero index, it returns a unit vector defined using the Kronecker delta function (see Section 5.3.5), where all components are zero except for the $(i-1)$-th component, which is set to 1. The reference vertices are depicted in Figure 9.2.

For instance, let us consider the case where $d = 3$ and $i \in [0..3]$, a simplex forms a tetrahedron with 4 vertices:

- For $i = 0$, the output is $\hat{\mathbf{v}}_0 = (0, 0, 0)$, denoting the origin point.
- For $i = 1$, the output is $\hat{\mathbf{v}}_1 = (1, 0, 0) = \boldsymbol{\delta}_0$, indicating the unit point along the x-axis.
- For $i = 2$, the output is $\hat{\mathbf{v}}_2 = (0, 1, 0) = \boldsymbol{\delta}_1$, indicating the unit point along the y-axis.
- For $i = 3$, the output is $\hat{\mathbf{v}}_3 = (0, 0, 1) = \boldsymbol{\delta}_2$, indicating the unit point along the z-axis.

Affine Independence of the Reference Vertices

An essential property of these vertices is their *affine independence*, which states that a family of points $(A_i)_{i\in[0..d]}$ is affinely independent if and only if the family of vectors $(A_{i+1} - A_0)_{i\in[0..d-1]}$ is linearly independent (see Sections 5.4.3 and 5.4.2 for the detailed definitions). This property ensures that no vertex can be expressed as an affine combination of the others (see Section 5.3.6). Affine independence offers several advantages: Firstly, it allows each element to retain a unique shape, which prevents the occurrence of degenerate elements where the shape might collapse into a lower-dimensional space, such as a triangle collapsing into a line segment. Another advantage is that it allows the transformation mapping T^K_{geo} to preserve the original shape of

the geometric element. This transformation mapping is briefly introduced in Section 6.2.2 and defined by the Equation (9.12), and will be discussed in more detail in Section 9.3 of this chapter.

We establish the affine independence of the reference vertices $\hat{\mathbf{v}}_0, \ldots, \hat{\mathbf{v}}_d$.

```
Lemma vtx_simplex_ref_affine_independent: affine_independent vtx_simplex_ref.
```

The proof of this lemma requires demonstrating that the only solution to the linear equation $\sum_{i=0}^{d-1} c_i \, (\hat{\mathbf{v}}_{i+1} - \hat{\mathbf{v}}_0) = 0$ is the trivial one, where all coefficients c_i are zero. Unfolding the definition of `vtx_simplex_ref` reveals that the difference between each $\hat{\mathbf{v}}_{i+1}$ and $\hat{\mathbf{v}}_0$ is the Kronecker delta function minus the zero function, simplifying to $\sum_{i=0}^{d-1} c_i(\delta_{i,j} - \mathbf{0}) = 0$ for $j \in [0..d-1]$. Using the properties of the Kronecker delta function (refer to Section 5.3.5), we conclude that $c_j = 0$ for all $j \in [0..d-1]$, establishing the affine independence of the family $(\hat{\mathbf{v}}_i)_{i \in [0..d]}$.

Reference Lagrange Nodes

In the Finite Element Method, reference Lagrange nodes within a simplex are predefined points that consist in constructing Lagrange polynomials (see Section 9.6), that approximate the solutions to equations defined over a geometric domain. These nodes are typically located at specific geometric locations such as vertices, edge centers, and face centers, among others. They are evenly distributed on the reference simplex $\hat{K}$. Mathematically, these nodes can be defined as follows:

$$\forall i \in [0..d-1], \quad (\hat{\mathbf{a}}_{\boldsymbol{\alpha}})_i \overset{\text{def.}}{=} \frac{\alpha_i}{k}, \quad \forall \boldsymbol{\alpha} \in \mathcal{A}_k^d. \tag{9.2}$$

Here, k denotes the degree of polynomial approximation for the space $\mathcal{P}_k^d$, and $\boldsymbol{\alpha}$ specifies the coordinates of each node (see Section 8.1.1). In Coq, we define the explicit expression of the reference Lagrange node as $(\hat{\mathbf{a}}_{\mathcal{A}_k^d[\mathtt{ipk}]})_i$, where $\mathtt{ipk} \in [0..\mathrm{card}\mathcal{A}_k^d - 1]$ and $i \in [0..d-1]$. In this context, `ipk` corresponds to the numbering of the multi-index $\boldsymbol{\alpha}$ within the family $\mathcal{A}_k^d$, and i represents the index of each item of $\mathcal{A}_k^d[\mathtt{ipk}]$ ranging from 0 to $d-1$. For the sake of brevity, we often work with nodes of the form $\hat{\mathbf{a}}_{\boldsymbol{\alpha}}$, which take vectors as inputs.

In Coq, the reference Lagrange nodes are implemented as:

```
Definition node_ref := fun (ipk : 'I_((pbinom d k).+1)) (i : 'I_d) ⇒
  INR (Adk d k ipk i) / INR k.
```

Here, the binomial expression `(pbinom d k).+1` represents the cardinal of $\mathcal{A}_k^d$ (refer to Sections 5.5 and 8.1). This definition is done using the `INR : nat → R` function that converts a natural number into a real number. This conversion is necessary because the function `node_ref` performs real arithmetic rather than integer arithmetic.

For a practical example, consider a 3D case with $k = 3$, as depicted in Figure 9.2. The family $\mathcal{A}_3^3$ includes $\boldsymbol{\alpha}$ values in $\mathbb{N}^3$ such that $|\boldsymbol{\alpha}| \leqslant 3$. Reference nodes would then be points in the unit simplex. For instance, for $\boldsymbol{\alpha} = (1,1,1)$ we obtain:

$$\hat{\mathbf{a}}_{(1,1,1)} = \frac{(1,1,1)}{3} = (1/3, 1/3, 1/3).$$

9.1.2 Definition of Current Lagrange Nodes

In the FEM, current Lagrange nodes [36] are not arbitrarily placed but are calculated through a process involving the vertices of the simplex. Mathematically, the position of each node within

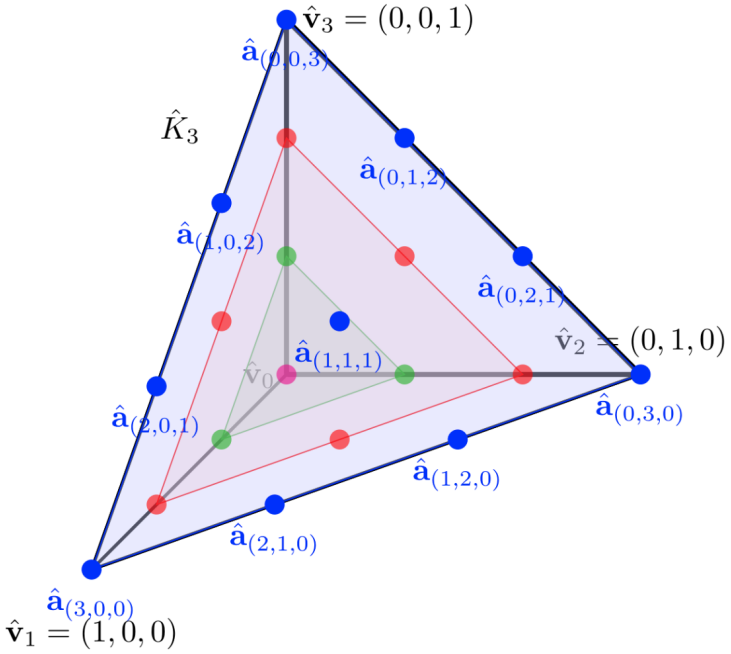

Figure 9.2: Lagrange nodes $\hat{\mathbf{a}}_{\boldsymbol{\alpha}}$ of the reference simplex $\hat{K}_3$ when $d = 3$ and $k = 3$. Each node is depicted as a colored ball, and corresponds to a unique element of $\mathcal{A}^3_3$. The colors correspond to degrees $\ell \leqslant 3$ of polynomials, or equivalently to lengths of multi-indices (i.e., in $\mathcal{C}^d_\ell$ for $\ell \leqslant 3$). In pink, the node $\hat{\mathbf{a}}_{\mathbf{0}}$ corresponds to constant polynomials (with degree 0) in $\mathcal{P}^3_0$, and the multi-index $\mathbf{0}$ in the singleton $\mathcal{C}^3_0$. In green, the nodes correspond to non-constant affine polynomials (with degree 1), and multi-indices $\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_d$ in $\mathcal{C}^3_1$. In red, the nodes correspond to non-affine quadratic polynomials (with degree 2), and multi-indices in $\mathcal{C}^3_2$. In blue, the nodes correspond to non-quadratic cubic polynomials (with degree 3), and multi-indices in $\mathcal{C}^3_3$. We observe in this picture that $\mathcal{A}^3_3 = \mathcal{C}^3_0 \cup \mathcal{C}^3_1 \cup \mathcal{C}^3_2 \cup \mathcal{C}^3_3$.

a simplex is derived using a linear combination of the vertex coordinates of the simplex. The weights of these combinations are determined by a family of multi-indices and the degree of the polynomial space, denoted as k. Let $(\mathbf{v}_i)_{i\in[0..d]}$ be a family of affinely independent points, the general formula for node positioning is represented in Coq as `node_cur` and expressed mathematically as:

$$\mathbf{a}_{\boldsymbol{\alpha}} \overset{\text{def.}}{=} \left(1 - \frac{|\boldsymbol{\alpha}|}{k}\right) \mathbf{v}_0 + \sum_{i=0}^{d-1} \frac{\alpha_i}{k}\, \mathbf{v}_{i+1}, \quad \forall \boldsymbol{\alpha} \in \mathcal{A}^d_k. \tag{9.3}$$

In the equation provided (9.3), $\left(1 - \frac{|\boldsymbol{\alpha}|}{k}\right)$ and $\frac{\alpha_i}{k}$ represent the barycentric coordinates of the current node $\mathbf{a}_{\boldsymbol{\alpha}}$ with respect to the vertices $(\mathbf{v}_i)_{i\in[0..d]}$ (refer to Section 5.3.6). These barycentric coordinates ensure that the current node $\mathbf{a}_{\boldsymbol{\alpha}}$ is a convex combination of the vertices $(\mathbf{v}_i)_{i\in[0..d]}$, positioned either inside or on the boundary of the simplex K. This distribution guarantees that the sum of the barycentric coordinates equals 1:

$$\left(1 - \frac{|\boldsymbol{\alpha}|}{k}\right) + \sum_{i=0}^{d-1} \frac{\alpha_i}{k} = 1.$$

Thus, $\mathbf{a}_{\boldsymbol{\alpha}}$ geometrically represents the barycenter of the vertices given by these coordinates.

The following Coq code snippet represents $\mathbf{a}_{\boldsymbol{\alpha}}$ as the function that computes the node positions based on the current vertices `vtx_cur` of the simplex in $\mathbb{R}^d$ and the maximum degree of polynomials k. This function is defined as follows:

```
Definition node_cur (vtx_cur : '('R^d)^(S d)) : 'R^{(pbinom d k).+1,d} :=
  fun (ipk : 'I_((pbinom d k).+1)) ⇒
    scal (1 − scal (/ INR k) (sum (mapF INR (Adk d k ipk)))) (vtx_cur ord0) +
    lin_comb (scal (/ INR k) (mapF INR (Adk d k ipk))) (liftF_S vtx_cur).
```

This function takes `vtx_cur` as argument, where `vtx_cur` is a family of $d+1$ vertices of the simplex in d-dimensional real space (`'R^d`). The operation `liftF_S` consists in skipping the first item of the family of vertices, to only keep $(\mathbf{v}_{i+1})_{i\in[0..d-1]}$. Moreover, the `mapF` function serves in applying the `INR` function to each element of the vector returned by `Adk` (see Section 5.2.3 for more detail about finite families).

To provide an example of how the node positions are calculated and distributed within a simplex, let us consider the same previous example of a three-dimensional case with a polynomial degree $k = 3$, and a family of vertices $(\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3) = ((1,1,1),(2,3,5),(0,1,0),(3,1,4))$ that are affinely independent. According to the Equation (9.3), the current node for each $\boldsymbol{\alpha} \in \mathcal{A}_3^3$ is defined by:

$$\mathbf{a}_{\boldsymbol{\alpha}} = \left(1 - \frac{|\boldsymbol{\alpha}|}{3}\right)\mathbf{v}_0 + \frac{\alpha_0}{3}\mathbf{v}_1 + \frac{\alpha_1}{3}\mathbf{v}_2 + \frac{\alpha_2}{3}\mathbf{v}_3.$$

For instance, for $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \alpha_2) = (1,0,0)$ we obtain

$$\mathbf{a}_{(1,0,0)} = \left(1 - \frac{1}{3}\right)(1,1,1) + \frac{1}{3}(2,3,5) = \left(\frac{4}{3}, \frac{5}{3}, \frac{7}{3}\right).$$

9.1.3 Connection Between Vertices and Nodes

The relationship between vertices and nodes is embedded, except when the degree $k = 0$: specifically, all vertices are considered nodes, but not all nodes are vertices. Depending on the polynomial degree, both reference and current nodes include the vertices and possibly other points along the face edges or even within the interior of the elements.

Let us demonstrate the equality and inclusion relationship between nodes $(\mathbf{a}_{\boldsymbol{\alpha}})_{\boldsymbol{\alpha}\in\mathcal{A}_k^d}$ and vertices $(\mathbf{v}_i)_{i\in[0..d]}$ for cases where $k = 1$ and $k > 1$ for all $d \geq 1$.

(i) For first-order polynomials where $k = 1$ (affine polynomials): Let us prove that the nodes coincide with the vertices, i.e.,

$$\forall \texttt{ipk} \in [0..d], \quad \mathbf{a}_{\mathcal{A}_k^d[\texttt{ipk}]} = \mathbf{v}_{\texttt{ipk}}. \tag{9.4}$$

This is captured in Coq as follows for all current vertices `vtx_cur`:

```
Lemma vtx_node_Pd1_cur : ∀(vtx_cur : 'R^{d.+1,d}) (ipk :'I_d.+1),
  vtx_cur ipk = node_cur d 1 vtx_cur (cast_ord _ ipk).
```

where, the `cast_ord` function is defined in Section 5.2.2. The proof of this lemma proceeds by first expanding the Equation (9.3) of `node_cur` when $k = 1$, leading to the equation:

$$\mathbf{a}_{\boldsymbol{\alpha}} = \left(1 - \sum_{i=0}^{d-1}\alpha_i\right)\mathbf{v}_0 + \sum_{i=0}^{d-1}\alpha_i\ \mathbf{v}_{i+1}, \quad \forall \boldsymbol{\alpha} \in \mathcal{A}_1^d. \tag{9.5}$$

From here, we address the case of the index `ipk` $\in [0..d]$ to be the first index `ord0` or not, leading to two specific cases, where `ipk` represents the numerotation of the multi-index $\boldsymbol{\alpha}$ in the family $\mathcal{A}_1^d$ (see Equation (8.5) in Section 8.1):

- Case 1: `ipk` $= 0$ (first vertex $\mathcal{A}_1^d[0] = (0, \ldots, 0)$):
  Using the `Adk_0` lemma, as detailed in Section 8.1.1, which specifies that the first multi-index for any dimension `d` and polynomial degree `k` results in 0, simplifies the equation as follows:

$$\mathbf{a}_{\mathcal{A}_1^d[0]} = (1-0)\,\mathbf{v}_0 + 0\;\mathbf{v}_1 + 0\;\mathbf{v}_2 + \ldots + 0\;\mathbf{v}_d = \mathbf{v}_0.$$

- Case 2: `ipk` $\neq 0$ (remaining vertices $\mathcal{A}_1^d[\texttt{ipk}] = \boldsymbol{\delta}_{\texttt{ipk}-1}$ for `ipk` $\in [1..d]$):
  The proof in this branch uses the `Ad1_eq` lemma, as outlined in Section 8.1.1, which establishes that for any dimension $d \geq 1$, the family $\mathcal{A}_1^d$, as described by the Equation (8.5), is defined as a concatenation of a zero function and a unit index function, denoted as `itemF d 1`. Given that `ipk` $\neq 0$, we affirm that the multi-index $\boldsymbol{\alpha}$ is equal to `itemF d 1`. This item function form is represented by the Kronecker delta function, which generates a vector where all elements are initially zero except for one element that equals 1 (refer to Section 5.2.3). Equation (9.5) becomes,

$$\mathbf{a}_{\mathcal{A}_1^d[\texttt{ipk}]} = \mathbf{a}_{\boldsymbol{\delta}_{\texttt{ipk}-1}} = \left(1 - \sum_{i=0}^{d-1} \delta_{\texttt{ipk}-1,i}\right)\mathbf{v}_0 + \sum_{i=0}^{d-1} \delta_{\texttt{ipk}-1,i}\;\mathbf{v}_{i+1} = \mathbf{v}_{\texttt{ipk}}, \tag{9.6}$$

  using properties of the Kronecker delta function (refer to Section 5.3.5), such as $\sum_{i=0}^{d-1} \delta_{\texttt{ipk}-1,i} = 1$ and $\sum_{i=0}^{d-1} \delta_{\texttt{ipk}-1,i}\;\mathbf{v}_{i+1} = \mathbf{v}_{\texttt{ipk}}$.

  The proof then concludes that for all cases, the node positions, as computed by `node_cur`, are indeed identical to the positions of the vertices. □

Similar to the previous lemma, the following lemma asserts that each reference vertex `vtx_simplex_ref` of the simplex in d dimensions corresponds exactly to a reference node calculated with a polynomial degree of 1.

```
Lemma vtx_node_Pd1_ref : ∀(ipk :'I_d.+1),
  vtx_simplex_ref d ipk = node_ref d 1 (cast_ord _ ipk).
```

The proof of this lemma is succinct, using the previously established lemma `vtx_node_Pd1_cur`.

(ii) For higher-order elements where $k > 1$, additional nodes are introduced. These nodes may be located on the faces or within the interior of the elements. In this case, we prove that every current vertex `vtx_cur` of a simplex in d dimensions is included within the family of current nodes `node_cur`, i.e.,

$$\forall \texttt{ipk} \in [0..d],\ \exists \texttt{jpk} \in [0..\text{card}\mathcal{A}_k^d - 1], \quad \mathbf{a}_{\mathcal{A}_k^d[\texttt{jpk}]} = \mathbf{v}_{\texttt{ipk}}. \tag{9.7}$$

This is translated in Coq as:

```
Lemma vtx_cur_invalF : ∀vtx_cur, invalF vtx_cur (node_cur vtx_cur).
```

The `invalF` function refers to the inclusion relation of finite families (see Section 5.2.3). The proof for this lemma is structured similarly to the previous one, using pattern matching to divide the proof into cases depending on whether the index `ipk` is the first index (`ord0`) or another index.

- Case 1: $\texttt{ipk} = 0$:
  This case is relatively straightforward. By setting $\texttt{jpk} = 0$, and using the `Adk_0` lemma established in Section 8.1.1, the equation of the current node (9.3) simplifies to merely the vertex at `ord0`, confirming direct inclusion.
- Case 2: $\texttt{ipk} \neq 0$:
  In this case, we use the inverse function `Adk_inv`, defined in Section 8.1.3, to determine the index `jpk` that should satisfy the equality between the current vertices and the current nodes $\mathbf{v}_{\texttt{ipk}} = \mathbf{a}_{\mathcal{A}_k^d[\texttt{jpk}]}$. The inverse multi-index is then chosen to be $\texttt{jpk} = (\mathcal{A}_k^d)^{-1}(k\ \boldsymbol{\delta}_{\texttt{ipk}-1})$ ensuring that $\mathcal{A}_k^d[\texttt{jpk}] \in \mathcal{A}_k^d$, since it satisfies the conditions that $k\ \boldsymbol{\delta}_{\texttt{ipk}-1} \in \mathbb{N}^d$ and $|k\ \boldsymbol{\delta}_{\texttt{ipk}-1}| = k\ \sum_{i=0}^{d-1} \delta_{\texttt{ipk}-1,i} = k \leqslant k$. Consequently, the current node defined in the Equation (9.3) transforms as follows:

$$\mathbf{a}_{\mathcal{A}_k^d[\texttt{jpk}]} = \mathbf{a}_{k\boldsymbol{\delta}_{\texttt{ipk}-1}} = \left(1 - \sum_{i=0}^{d-1} \frac{k\ \delta_{\texttt{ipk}-1,i}}{k}\right) \mathbf{v}_0 + \sum_{i=0}^{d-1} \frac{k\ \delta_{\texttt{ipk}-1,i}}{k}\ \mathbf{v}_{i+1} = \mathbf{v}_{\texttt{ipk}},$$

  using the same properties of the Kronecker delta, as outlined in Section 5.3.5. Thus, this validates the lemma. □

Similarly, every element of `vtx_simplex_ref` is an element of `node_ref`.

```
Lemma vtx_simplex_ref_inclF : invalF vtx_simplex_ref node_ref.
```

The proof of this lemma derives directly from the previously established lemma `vtx_cur_invalF`.

9.1.4 Lagrange Sub-vertices and Sub-nodes

In this section, we formalize sub-vertices and sub-nodes. These concepts are essential for the upcoming chapter (see Section 10.3.7), where we establish the unisolvence property of the Lagrange finite element. The proof depends on applying the inductive principle to k, which requires to relate concepts and properties between k and $k-1$. Here, we assume that $k \geq 1$.

Sub-Vertices of Lagrange Nodes

Let $(\mathbf{v}_i)_{i\in[0..d]}$ be an affine independent family of vertices in $\mathbb{R}^d$. We highlight the equivalence between the current sub-vertices and the Lagrange nodes. Specifically, the current sub-vertices $(\check{\mathbf{v}}_i)_{i\in[0..d]}$ of a simplex in $\mathbb{R}^d$ for all $d \geq 1$, correspond exactly to specific Lagrange nodes when decrementing the polynomial degree k by one (refer to Figure 9.3). More precisely, for all $k \geq 1$ we have:

$$\forall i \in [1..d], \quad \check{\mathbf{v}}_i \overset{\text{def.}}{=} \mathbf{a}_{(k-1)\boldsymbol{\delta}_{i-1}} \quad \text{and} \quad \check{\mathbf{v}}_0 \overset{\text{def.}}{=} \mathbf{v}_0. \tag{9.8}$$

where $\boldsymbol{\delta}_i$ is the Kronecker delta function (refer to Section 5.3.5).

```
Definition sub_vertex := fun (i : 'I_d.+1) ⇒
  match (ord_eq_dec i ord0) with
    | left _ ⇒ vtx_cur ord0
    | right Hi ⇒ node_cur_alt d k vtx_cur (Adk_inv d k (itemF d (k.−1) (lower_S Hi)))
   end.
```

In this context, `node_cur_alt` refers to the current Lagrange nodes defined by the Equation (9.3), and `itemF d (k.−1) (lower_S Hi)` is the finite family where elements are the kronecker delta function $\boldsymbol{\delta}_{i-1}$ multiplied by $k-1$. Moreover, the function `lower_S` is used to adjust the index `i` by lowering it by 1 within the set of indices `'I_d.+1` (see Section 5.2.2).

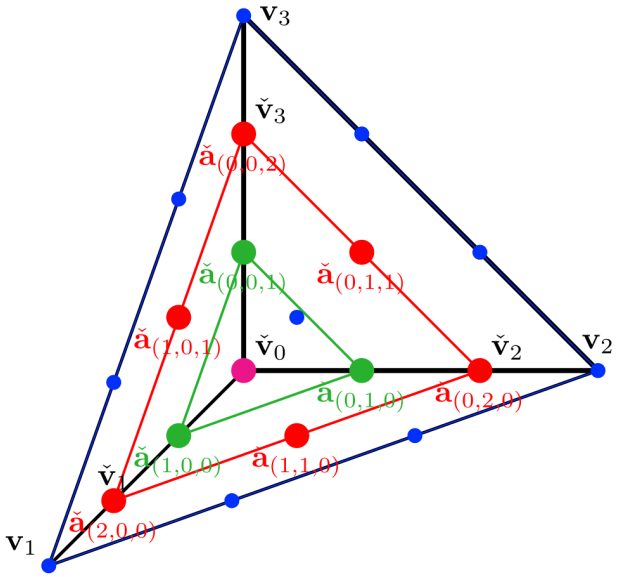

Figure 9.3: Illustration of the passage from the nodes of $\mathcal{A}_3^3$ (degree $\leqslant 3$) to the nodes of $\mathcal{A}_2^3$ (degree $\leqslant 2$) in the case $d=3$. It corresponds geometrically to passing from the tetrahedron defined by $(\mathbf{v}_0,\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3)$ to the tetrahedron defined by the sub-vertices $(\check{\mathbf{v}}_0,\check{\mathbf{v}}_1,\check{\mathbf{v}}_2,\check{\mathbf{v}}_3)$, where $\check{\mathbf{v}}_0=\mathbf{v}_0$ and, $\forall i\in[1..3], \check{\mathbf{v}}_i=\mathbf{a}_{2\boldsymbol{\delta}_{i-1}}$ (these are the *sub-vertices*, i.e., the last nodes along the axes that are not the vertices for $i\in[1..3]$). The sub-nodes $\check{\mathbf{a}}_{\boldsymbol{\alpha}}$ with $\boldsymbol{\alpha}\in\mathcal{A}_2^3$ in the tetrahedron $(\check{\mathbf{v}}_0,\check{\mathbf{v}}_1,\check{\mathbf{v}}_2,\check{\mathbf{v}}_3)$ are the same as the nodes $\mathbf{a}_{\boldsymbol{\alpha}}$ of $(\mathbf{v}_0,\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3)$ with $\boldsymbol{\alpha}\in\mathcal{A}_2^3$, except the (blue) nodes that correspond to $k=3$ ($\mathcal{C}_3^3$).

We express the sub-vertices by this expression for all $k\geq 1$:

$$\forall i\in[0..d],\quad \check{\mathbf{v}}_i=\frac{1}{k}\mathbf{v}_0+\frac{k-1}{k}\mathbf{v}_i. \tag{9.9}$$

Let us prove this Equation (9.9). We proceed by cases on the index i. When $i=0$, we have $\check{\mathbf{v}}_0=\mathbf{v}_0$. For $i\neq 0$, the proof uses the `sum_itemF` lemma (refer to Section 5.2) to demonstrate that the length $|(k-1)\,\boldsymbol{\delta}_{i-1}|=(k-1)\sum_{j=0}^{d-1}\delta_{i-1,j}=k-1\leqslant k$. Consequently, $(k-1)\,\boldsymbol{\delta}_{i-1}\in\mathcal{A}_k^d$. The computation then simplifies further as follows using Equations (9.3) and (9.8).

$$\begin{aligned}\check{\mathbf{v}}_i=\mathbf{a}_{(k-1)\boldsymbol{\delta}_{i-1}}&=\left(1-\sum_{j=0}^{d-1}\frac{(k-1)\;\delta_{i-1,j}}{k}\right)\mathbf{v}_0+\sum_{j=0}^{d-1}\frac{(k-1)\;\delta_{i-1,j}}{k}\;\mathbf{v}_{j+1},\\&=\frac{1}{k}\mathbf{v}_0+\frac{k-1}{k}\mathbf{v}_i.\end{aligned}$$

□

When the degree k is set to 1, we can deduce from Equation (9.9) that all sub-vertices of a simplex are identical to the first vertex $\mathbf{v}_0$, denoted in Coq as `vtx_cur ord0`. This assertion is formalized as follows:

```
Lemma sub_vertex_k_1 : ∀(i:'I_d.+1), k = 1 → sub_vertex i = vtx_cur ord0.
```

The function of sub-vertices referenced in the Equation (9.9) preserves affine independence when applied to a family of affinely independent vertices $(\mathbf{v}_i)_{i\in[0..d]}$ in $\mathbb{R}^d$ when $k > 1$.

```
Hypothesis k_gt_1 : 1 < k.
Hypothesis Hvtx : affine_independent vtx_cur.
Lemma sub_vertex_affine_independent : affine_independent (sub_vertex d k vtx_cur).
```

This is established by assuming the linear combination $\sum_{i=0}^{d-1} c_i(\check{\mathbf{v}}_{i+1} - \check{\mathbf{v}}_0) = 0$, we demonstrate that all coefficients c_i must be zero. By unfolding the Equation (9.9) of sub-vertices and using properties of affine combinations, scalar multiplication (see Section 5.3.6), we transform this equality into:

$$\sum_{i=0}^{d-1} c_i \left(\frac{k-1}{k} \mathbf{v}_{i+1} - \left(1 - \frac{1}{k}\right) \mathbf{v}_0 \right) = \frac{k-1}{k} \sum_{i=0}^{d-1} c_i (\mathbf{v}_{i+1} - \mathbf{v}_0) = 0.$$

Given that $\frac{k-1}{k} \neq 0$ and the vertices $\mathbf{v}_i$ are affinely independent, we affirm that all coefficients c_i are necessarily zero for $i \in [0..d-1]$. □

Sub-Nodes of Lagrange Nodes

Another interesting concept within the framework of Lagrange nodes is that of *sub-nodes*, as illustrated in Figure 9.3. These sub-nodes, which are points in $\mathbb{R}^d$, correspond to the current nodes at a lower polynomial degree $k-1$ associated with the sub-vertices $(\check{\mathbf{v}}_i)_{i\in[0..d]}$. The definition of sub-nodes, $\check{\mathbf{a}}_{\boldsymbol{\alpha}}$ for all $\boldsymbol{\alpha} \in \mathcal{A}^d_{k-1}$, is provided in Coq as follows:

```
Definition sub_node : 'R^d)^((pbinom d k.-1).+1) :=
  node_cur d (k.-1) (sub_vertex d k vtx_cur).
```

Mathematically, each sub-node is defined as an affine combination of sub-vertices (see Section 5.3.6), where the coefficients are derived from a family $\boldsymbol{\alpha}$ of multi-indices in $\mathcal{A}^d_{k-1}$, ensuring that the total weight of the coefficients sums up to 1. For all $d \geq 1$ and $k > 1$ we have:

$$\forall \boldsymbol{\alpha} \in \mathcal{A}^d_{k-1}, \quad \check{\mathbf{a}}_{\boldsymbol{\alpha}} \overset{\text{def.}}{=} \mathbf{a}^{\check{\mathbf{v}}}_{\boldsymbol{\alpha}}, \tag{9.10}$$

Here, $\mathbf{a}^{\check{\mathbf{v}}}_{\boldsymbol{\alpha}}$ represents the current node, as defined by Equation (9.3) in Section 9.1.2, which takes $\check{\mathbf{v}}$ as an argument. This means $\mathbf{a}_{\boldsymbol{\alpha}}$ is expressed in terms of the sub-vertices. For simplicity in mathematical expressions, we often omit the argument and refer to it simply as $\mathbf{a}_{\boldsymbol{\alpha}}$.

Now, by unfolding the Equation (9.9) of sub-vertices, we deduce that each sub-node defined on the simplex $(\check{\mathbf{v}}_0, \ldots, \check{\mathbf{v}}_d)$, when $\boldsymbol{\alpha} \in \mathcal{A}^d_{k-1}$, aligns exactly with the original current nodes $\mathbf{a}^{\mathbf{v}}_{\boldsymbol{\alpha}}$ generated at the degree k applied to current vertices $(\mathbf{v}_i)_{i\in[0..d]}$. This correspondence between nodes and sub-nodes is established through the following lemma:

```
Lemma sub_node_cur_eq : ∀(ipk : 'I_((pbinom d k.-1).+1)),
  sub_node ipk = node_cur d k vtx_cur (widen_ord _ ipk).
```

where, `widen_ord` is the injection from the set with a smaller number of elements corresponding here to $\mathcal{A}^d_{k-1}$, to a larger one $\mathcal{A}^d_k$ used to adjust the type of indices (see Section 5.2.2).

The proof of this lemma is verified for each $\boldsymbol{\alpha} \in \mathcal{A}^d_{k-1}$ as:

$$\begin{aligned}
\check{\mathbf{a}}_{\boldsymbol{\alpha}} &\overset{\text{def.}}{=} \mathbf{a}_{\boldsymbol{\alpha}}^{\check{\mathbf{v}}} = \left(1 - \frac{|\boldsymbol{\alpha}|}{k-1}\right) \check{\mathbf{v}}_0 + \sum_{i=0}^{d-1} \frac{\alpha_i}{k-1}\, \check{\mathbf{v}}_{i+1}, \quad \text{from Equation (9.10)} \\
&= \left(1 - \frac{|\boldsymbol{\alpha}|}{k-1}\right) \mathbf{v}_0 + \sum_{i=0}^{d-1} \frac{\alpha_i}{k-1} \left(\frac{1}{k}\mathbf{v}_0 + \frac{k-1}{k}\mathbf{v}_{i+1}\right), \quad \text{from Equation (9.9)} \\
&= \left(1 - \frac{|\boldsymbol{\alpha}|}{k}\right) \mathbf{v}_0 + \sum_{i=0}^{d-1} \frac{\alpha_i}{k}\mathbf{v}_{i+1}, \quad \text{(real arithmetic)} \\
&= \mathbf{a}_{\boldsymbol{\alpha}}^{\mathbf{v}}.
\end{aligned}$$

9.2 $\mathcal{P}_1^d$ Lagrange Polynomial Bases on the Reference Element

In this section, we formalize and explore the characteristics of Lagrange polynomials used to approximate solutions across the domain of the problem, particularly when the dimension $d \geq 1$ and the degree of approximation is $k = 1$. The so called reference *Lagrange polynomial bases*, denoted as $(\hat{\mathcal{L}}_j^{d,1})_{j\in[0..d]}$, for the polynomial space $\mathcal{P}_1^d$ (see Section 8.3), are mathematically represented as:

$$\forall j \in [0..d], \quad \forall \hat{\mathbf{x}} \in \mathbb{R}^d, \quad \hat{\mathcal{L}}_j^{d,1}(\hat{\mathbf{x}}) \overset{\text{def.}}{=} \begin{cases} 1 - \sum_{i=0}^{d-1} \hat{x}_i, & \text{if } j = 0 \\ \hat{x}_{j-1}, & \text{else.} \end{cases} \tag{9.11}$$

This is translated into Coq as a match-case approach:

```
Definition LagPd1_ref : 'I_(FRd d).+1 -> 'R^d := fun (j : 'I_d.+1) (x : 'R^d) =>
  match (ord_eq_dec j ord0) with
    | left _ => 1 - sum x
    | right H => x (lower_S H)
    end.
```

The function `lower_S` consists in decrementing the index j (see Section 5.2.2). These reference Lagrange polynomials $(\hat{\mathcal{L}}_j^{d,1})_{j\in[0..d]}$ are specifically designed to be 1 at one vertex and 0 at all others within the reference geometric element. Their primary role is to provide a basis for interpolating values within each element. To express this property, we refer to the Coq lemma verifying the Kronecker delta property:

```
Lemma LagPd1_ref_kron_vtx : ∀i j : 'I_d.+1,
  LagPd1_ref j (vtx_simplex_ref d i) = kronecker i j.
```

To demonstrate this, we first expand the definitions of `LagPd1_ref` and `vtx_simplex_ref`, followed by a detailed case study based on whether indices `i` or `j` are zero, dividing the proof into four distinct cases:

- Case 1: `i` $= 0$, `j` $= 0$:
  The equation simplifies to $1 - \sum_{\ell=0}^{d-1} 0 = 1 = \delta_{0,0}$, aligning with the Kronecker delta property as specified by the lemma `kronecker_is_1`. Since the summation of zeros simplifies to zero, the polynomial evaluates to one.

- Case 2: `i` $\neq 0$, `j` $= 0$:
  The polynomial equation becomes $1 - \sum_{\ell=0}^{d-1} \delta_{i-1,\ell} = 0 = \delta_{i,0}$, consistent with the lemma `kronecker_is_0` due to the non-matching indices. According to the lemma `sum_kronecker_r`, discussed in Section 5.3.5, the summation equals one, resulting in the polynomial evaluating to zero.

- Case 3: `i` $= 0$, `j` $\neq 0$:
  In this case, the equation evaluates again to zero, as expected when indices differ, supported by the lemma `kronecker_is_0`.

- Case 4: `i` $\neq 0$, `j` $\neq 0$:
  The equation simplifies to $\delta_{i-1,j-1} = \delta_{i,j}$. This equality holds regardless of whether the indices i and j are distinct or not, using lemma `kronecker_pred_eq`, the equality holds.

□

Furthermore, when $k = 1$, the Lagrange polynomials satisfy the Kronecker delta condition when evaluated at node points.

```
Lemma LagPd1_ref_kron_node : ∀i j : 'I_d.+1, LagPd1_ref j (node_ref d 1 i) = kronecker i j.
```

This property holds because when $k = 1$, we deduce from lemma `vtx_node_Pd1_ref`, that the reference vertices coincide with the reference nodes (refer to Section 9.1.3).

Another important property of these polynomials is that they are affine mappings in the function space `FRd d`. An affine map, in mathematical terms, can be described as a linear transformation followed by a translation, as defined in Section 5.3.6. This property is encapsulated in the lemma:

```
Lemma LagPd1_ref_am: ∀i : 'I_d.+1, aff_map (LagPd1_ref i).
```

Let $i \in [0..d]$. The proof is demonstrated by first defining a function g as $g(\hat{\mathbf{x}}) = \hat{\mathcal{L}}_i^{d,1}(\hat{\mathbf{x}}) - \hat{\mathcal{L}}_i^{d,1}(0)$. Using the lemma `am_lm_ms` detailed in Section 5.3.6, it suffices to demonstrate that the function g is linear. This involves examining different cases depending on the index i in $[0..d]$. When $i = 0$, we obtain $g(\hat{\mathbf{x}}) = 1 - \sum_{i=0}^{d-1} \hat{x}_i - 1 = -\sum_{i=0}^{d-1} \hat{x}_i$, which straightforwardly is a linear function after simplification. For $i \neq 0$, the function g directly accesses a specific component of $\hat{\mathbf{x}}$, i.e., $g(\hat{\mathbf{x}}) = \hat{x}_{i-1}$, which is inherently linear thanks to `lm_component` lemma (see Section 5.3.4). This confirms that g maintains linearity in all cases, thus establishing the affine nature of each $\hat{\mathcal{L}}_i^{d,1}$ function.
□

Additionally, the reference Lagrange polynomials $(\hat{\mathcal{L}}_j^{d,1})_{j\in[0..d]}$ simplify the linear combination of vertices of a reference simplex to a single coefficient under the condition that the sum of weights equals 1.

```
Lemma LagPd1_ref_lc : ∀(L : 'R^d.+1) (j : 'I_d.+1),
  sum L = 1 → LagPd1_ref j (lin_comb L (vtx_simplex_ref d)) = L j.
```

This lemma uses the property that an affine mapping preserves the barycenter of reference vertices $(\hat{\mathbf{v}}_i)_{i\in[0..d]}$ under an affine combination, as detailed in Section 5.3.6 by the `aff_map` definition. The result can be articulated through the equation:

$$\hat{\mathcal{L}}_j^{d,1}\left(\sum_{i=0}^{d} L_i \hat{\mathbf{v}}_i\right) = \sum_{i=0}^{d} L_i \hat{\mathcal{L}}_j^{d,1}(\hat{\mathbf{v}}_i) = \sum_{i=0}^{d} L_i \delta_{i,j} = L_j.$$

This follows directly from the previous Lemma `LagPd1_ref_kron_vtx`.
□

Lagrange polynomials are designed such that when evaluated at any point in the simplex in $\mathbb{R}^d$, the sum of all polynomials equals one. This property ensures that the polynomials form a partition of unity, which implies that these polynomials can be used to construct local approximations of functions. The summation property is formally expressed in the lemma:

```
Lemma LagPd1_ref_sum_1 : ∀x : 'R^d, sum (fun i : 'I_d.+1 ⇒ LagPd1_ref i x ) = 1.
```

The proof is straightforward and begins by applying the `sum_ind_1` lemma (refer to Section 5.3.1), which decomposes the summation of the polynomials $(\hat{\mathcal{L}}_i^{d,1})_{i\in[0..d]}$. By unfolding the definition given in the Equation (9.11) of these Lagrange polynomials, and using properties such as associativity and commutativity to rearrange and simplify the terms, the equation for any $\hat{\mathbf{x}} \in \mathbb{R}^d$ becomes:

$$\sum_{i=0}^{d} \hat{\mathcal{L}}_i^{d,1}(\hat{\mathbf{x}}) = \hat{\mathcal{L}}_0^{d,1}(\hat{\mathbf{x}}) + \sum_{i=0}^{d-1} \hat{\mathcal{L}}_{i+1}^{d,1}(\hat{\mathbf{x}}) = \left(1 - \sum_{i=0}^{d-1} \hat{\mathbf{x}}_i\right) + \sum_{i=0}^{d-1} \hat{\mathbf{x}}_i = 1.$$

□

From these results, we ensure that the Lagrange polynomials are linearly independent.

Lemma `LagPd1_ref_lin_indep` : `lin_indep LagPd1_ref`.

Indeed, let $(c_j)_{j\in[0,d]} \in \mathbb{R}$ such that for all $\hat{\mathbf{x}} \in \mathbb{R}^d$, $\sum_{j=0}^{d} c_j \hat{\mathcal{L}}_j^{d,1}(\hat{\mathbf{x}}) = 0$. From Lemma `LagPd1_ref_kron_vtx` and the ring properties of $\mathbb{R}$, setting $\hat{\mathbf{x}} = \hat{\mathbf{v}}_i$ for all $i \in [0..d]$, results in $\sum_{j=0}^{d} c_j \delta_{i,j} = c_i = 0$. Consequently, the Lagrange polynomial family is free.

Given the linear independence of the Lagrange polynomials, they form a basis for $\mathcal{P}_1^d$, since they span the entire vector space $\mathcal{P}_1^d$. This is true as stated in the lemma below,

Lemma `Pd1_lin_span_LagPd1_ref` : `Pdk d 1 = lin_span (LagPd1_ref d)`.

The proof for this lemma demonstrates that the space $\mathcal{P}_1^d$ spanned by the monomials functions $(\mathcal{B}_i^{d,1})_{i\in[0..d]}$, as detailed in Section 8.3, is equal to the linear span of the reference Lagrange polynomials $(\hat{\mathcal{L}}_j^{d,1})_{j\in[0..d]}$. Using the lemma `lin_span_ext` from Section 5.4.1, involves showing that every polynomial in $\mathcal{P}_1^d$ can be expressed as a linear combination of the Lagrange polynomials and vice versa, thereby proving that these two families span the same vector space. We begin with a case analysis on the index $i \in [0..d]$ for both inclusions.

(i) Prove that $\mathcal{B}_i^{d,1} \in \text{span } (\hat{\mathcal{L}}_j^{d,1})_{j\in[0..d]}$ (i.e., $\exists \boldsymbol{L} \in \mathbb{R}^{d+1}$, $\mathcal{B}_i^{d,1} = \sum_{j=0}^{d} L_j \hat{\mathcal{L}}_j^{d,1}$):

- Case 1: `i = 0`:
  We construct a vector $\boldsymbol{L}$:= (`concatF (singleF 1) ones`) in $\mathbb{R}^{d+1}$ (i.e., a collection of ones with the appropriate type $\boldsymbol{L} = (1, \ldots, 1) \in \mathbb{R}^{d+1}$), where `concatF` and `singleF` are defined in Section 5.2.3. This vector $\boldsymbol{L}$ is specifically designed to express the basis polynomial $\mathcal{B}_0^{d,1}$ in terms of the Lagrange polynomials as follows:

$$\begin{aligned}\sum_{j=0}^{d} L_j \hat{\mathcal{L}}_j^{d,1}(\hat{\mathbf{x}}) &= L_0 \hat{\mathcal{L}}_0^{d,1}(\hat{\mathbf{x}}) + \sum_{j=0}^{d-1} L_{j+1} \hat{\mathcal{L}}_{j+1}^{d,1}(\hat{\mathbf{x}}) \\ &= 1 \times (1 - \sum_{j=0}^{d-1} \hat{\mathbf{x}}_j) + (1, \ldots, 1) \times \sum_{j=0}^{d-1} \hat{\mathbf{x}}_j \\ &= 1.\end{aligned}$$

  This is verified using the `lc_ind_l` lemma, which consists in decomposing a linear combination into a sum of the first computed result and the linear combination of the rest (see Section 5.3.4). Hence, since $\mathcal{B}_0^{d,1} = 1$ from `BasisPd1_0` lemma as addressed in Section 8.3, the equation simplifies to:

$$\mathcal{B}_0^{d,1}(\hat{\mathbf{x}}) = \sum_{j=0}^{d} L_j \hat{\mathcal{L}}_j^{d,1}(\hat{\mathbf{x}}).$$

- Case 2: `i` $\neq 0$:
  In this segment of the proof, using the definition given in Equation (9.11), and referencing the `BasisPd1_neq0` lemma detailed in Section 8.3, we establish that $\hat{\mathcal{L}}_i^{d,1} = \mathcal{B}_i^{d,1} = \hat{\mathbf{x}}_{i-1}$ for $i \in [1..d]$. It then suffices to show that $\hat{\mathcal{L}}_i^{d,1} \in \text{span } \hat{\mathcal{L}}_i^{d,1}$, a conclusion that follows directly from the lemma `lin_span_inclF_diag`, as elaborated in Section 5.4.1.

(ii) Prove that $\hat{\mathcal{L}}_j^{d,1} \in \text{span } (\mathcal{B}_i^{d,1})_{i\in[0..d]}$:
This case proceeds similarly to the first addressing each case depending on whether the index `j` is zero or not. There are a few changes, particularly in the choice of vector $\boldsymbol{L}$. For `j` $\neq 0$, we construct the vector as $\boldsymbol{L} :=$ (`concatF` (`singleF` 1) (`opp ones`)) in $\mathbb{R}^{d+1}$ (i.e., a collection of ones with the appropriate type $\boldsymbol{L} = (1, -1, \ldots, -1) \in \mathbb{R}^{d+1}$), and then replicate the proof process.

By expanding the properties of Lagrange polynomials in multidimensional spaces, particularly at the first-degree approximation, we establish their bijectivity through the proofs of surjectivity and injectivity. This bijectivity is fundamental for proving the bijectivity of the geometric transformation T_{geo}^K, which will be defined in the next section to map the reference geometric element to the current one.

The demonstration of surjectivity is encapsulated in the following lemma:

```
Lemma LagPd1_ref_surj : ∀L : 'R^d.+1, sum L = 1 →
  ∃ x : 'R^d, (fun j : 'I_d.+1 ⇒ LagPd1_ref j x) = L.
```

We begin the proof by choosing a vector $\mathbf{x}$ in $\mathbb{R}^d$, which we specifically set as $L_{i+1} \in \mathbb{R}^d$ for all $i \in [0..d-1]$. To proceed, we verify that $\hat{\mathcal{L}}_j^{d,1}(L_{i+1}) = L_j$ for each $j \in [0..d]$ and $i \in [0..d-1]$. We analyze the scenarios for the index j. In the base case where $j = 0$, the definition of the Lagrange polynomial unfolds through Equation (9.11), and applying the lemma `sum_ind_l` (see Section 5.3.1) leads us to:

$$\hat{\mathcal{L}}_0^{d,1}(L_{i+1}) = 1 - \sum_{\ell=0}^{d-1} L_{\ell+1} = L_0.$$

For cases where $j \neq 0$, we confirm that $\hat{\mathcal{L}}_{j+1}^{d,1}(L_{i+1}) = L_j$ using the same Equation (9.11).

Now, we address the injectivity property of the Lagrange polynomials, which is established through the following lemma:

```
Lemma LagPd1_ref_inj : ∀(x y : 'R^d),
  (∀ j : 'I_d.+1, LagPd1_ref j x = LagPd1_ref j y) → x = y.
```

The proof of this lemma is straightforward. Since for $i \in [0..d-1]$ $\hat{\mathbf{x}}_i = \hat{\mathcal{L}}_{i+1}^{d,1}(\hat{\mathbf{x}})$ and $\hat{\mathbf{y}}_i = \hat{\mathcal{L}}_{i+1}^{d,1}(\hat{\mathbf{y}})$ hold for all $\hat{\mathbf{x}}, \hat{\mathbf{y}} \in \mathbb{R}^d$, the desired result follows directly from our initial assumption.

9.3 Affine Geometrical Transformation of Finite Element

The generation of a current geometric element K from the reference geometric element, denoted as $\hat{K}$, involves a C^1-diffeomorphism geometric transformation, represented by $T_{\text{geo}}^K : \hat{K} \to K$, which effectively maps $\hat{K}$ into each cell K of the mesh $\mathcal{T}_h$ as depicted in Figure 9.4. A practical approach to implementing this mapping is through the use of the reference Lagrange polynomials when $k = 1$ (see e.g. [36, Section 8.1, p.89]), along with current vertices of the cell

K.

We recall from Sections 6.2.2 and 7.1 that the geometric regions $\hat{K}$ and K, typically take the form of geometric primitives like triangles in 2D or tetrahedra in 3D. These shapes are defined using the convex hulls of families of vertices in the reference and current elements, respectively.

```
Definition K_geom_cur : 'R^d → Prop := convex_envelop vtx_cur. (*K*)
Definition K_geom_ref : 'R^d → Prop := convex_envelop (vtx_simplex_ref d). (*Khat*)
```

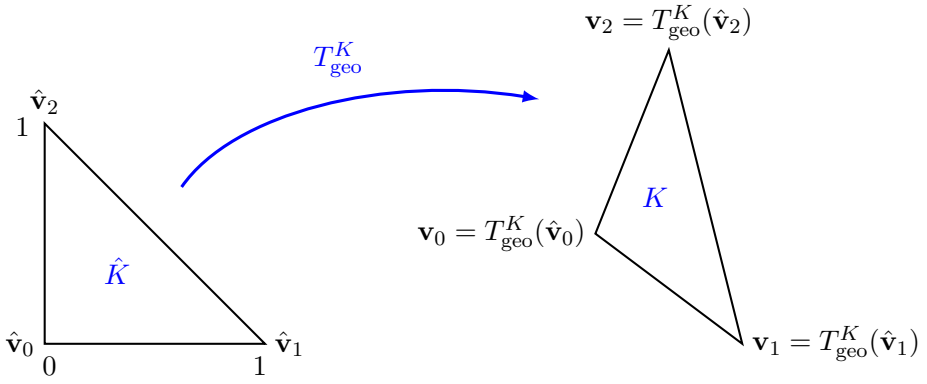

Figure 9.4: Example of a geometric transformation of elements from the reference geometric element to the current element. The unit triangle on the left, denoted as $\hat{K}$, represents the reference element. The vertices of the reference triangle, $\hat{\mathbf{v}}_0$, $\hat{\mathbf{v}}_1$, and $\hat{\mathbf{v}}_2$, correspond to the coordinates (0,0), (1,0), and (0,1) respectively. The arrow labeled T^K_{geo} represents a geometric transformation function. This function maps the reference element $\hat{K}$ to the corresponding current element K. The triangle on the right, denoted as K, represents the current geometric element. The vertices v_0, v_1, and v_2 of the current triangle are the images of the reference vertices under the transformation T^K_{geo}, expressed as $v_0 = T^K_{\text{geo}}(\hat{\mathbf{v}}_0)$, $v_1 = T^K_{\text{geo}}(\hat{\mathbf{v}}_1)$, and $v_2 = T^K_{\text{geo}}(\hat{\mathbf{v}}_2)$.

The geometric transformation T^K_{geo} is mathematically formulated in Equation (6.7) of Section 6.2.2, and is specified here using Lagrange polynomials $(\hat{\mathcal{L}}^{d,1}_i)_{i\in[0..d]}$ (refer to Equation (9.11) in Section 9.2) as the chosen shape functions.

$$T^K_{\text{geo}} : \hat{K} \ni \hat{\mathbf{x}} \longmapsto \sum_{i=0}^{d} \hat{\mathcal{L}}^{d,1}_i(\hat{\mathbf{x}})\mathbf{v}_i \in K. \tag{9.12}$$

where $\hat{\mathbf{x}}$ represents a point in $\hat{K}$, and $(\mathbf{v}_0, \ldots, \mathbf{v}_d)$ are the vertices of the current cell K in the mesh $\mathcal{T}_h$. This mapping is translated into Coq as a total function.

```
Definition T_geom : 'R^d → 'R^d := fun x_ref ⇒
  lin_comb (fun i : 'I_d.+1 ⇒ LagPd1_ref d i x_ref) vtx_cur.
```

This implementation effectively demonstrates how each point in the reference element is mapped to a point in the current element through a linear combination of the vertices weighted by the values of the reference Lagrange polynomials at $\hat{\mathbf{x}}$.

All transformations $(T^K_{\text{geo}})_{K\in\mathcal{T}_h}$ are affine, if and only if the geometric transformation includes a translation vector $b_K \in \mathbb{R}^d$ and a Jacobian matrix $\boldsymbol{J}^K_{\text{geo}} \in \mathbb{R}^{d,d}$ for all $K \in \mathcal{T}_h$ [36, Section 8.1, p.89], such that:

$$T^K_{\text{geo}} : \hat{K} \ni \hat{\mathbf{x}} \longmapsto \boldsymbol{J}^K_{\text{geo}}\hat{\mathbf{x}} + b_K, \tag{9.13}$$

This property has not been formalized in Coq; however, the lemma verifying that T^K_{geo} is an affine mapping is established as follows:

```
Lemma T_geom_am : aff_map T_geom.
```

To verify this, we define for all $\hat{\mathbf{x}} \in \mathbb{R}^d$,

$$g(\hat{\mathbf{x}}) = T^K_{\text{geo}}(\hat{\mathbf{x}}) - T^K_{\text{geo}}(0) = \sum_{i=0}^{d} \hat{\mathcal{L}}_i^{d,1}(\hat{\mathbf{x}})\mathbf{v}_i - \sum_{i=0}^{d} \hat{\mathcal{L}}_i^{d,1}(0)\mathbf{v}_i.$$

Given that an affine map can be expressed as a linear map plus a constant vector thanks to `am_lm_ms` lemma, it suffices to show that g is a linear map. Since each reference Lagrange polynomial $\hat{\mathcal{L}}_i^{d,1}$ is an affine map through the lemma `LagPd1_ref_am` (refer to Section 9.2), it follows that their differences $\hat{\mathcal{L}}_i^{d,1}(\hat{\mathbf{x}}) - \hat{\mathcal{L}}_i^{d,1}(0)$ represent linear maps. By applying the lemma `am_lm_0_rev` (see Section 5.3.6), we affirm the linearity of these differences. Finally, using the `fct_lc_l_lm` lemma, we confirm that a linear combination of linear maps remains linear, as detailed in Section 5.3.4.
□

The primary use of T^K_{geo} in finite element analysis is to map reference vertices $(\hat{\mathbf{v}}_i)_{i\in[0..d]}$ and nodes $(\hat{\mathbf{a}}_{\boldsymbol{\alpha}})_{\boldsymbol{\alpha}\in\mathcal{A}^d_k}$ to their corresponding current positions (refer to Section 9.1). Specifically, for each vertex within a simplex of dimension d, we state the following lemma,

```
Lemma T_geom_transports_vtx: ∀(vtx_cur : '('R^d)^d.+1) (i : 'I_d.+1),
  T_geom (vtx_simplex_ref d i) = vtx_cur i.
```

This lemma is easy and is straightforwardly validated by using the `LagPd1_ref_kron_vtx` lemma, which is detailed in Section 9.2, ensuring that the linear combination involving the Kronecker delta function simplifies to precisely the i-th vertex, thereby confirming the correct mapping of vertices from the reference to a current element.

Similarly, each reference node in dimension d, specified by the function `node_ref` in Coq, is mapped onto the current nodes `node_cur` with the same index. This is articulated through the following lemma:

```
Lemma T_geom_transports_node : ∀k (ipk : 'I_(pbinom d k).+1),
  T_geom (node_ref d k ipk) = node_cur d k vtx_cur ipk.
```

The proof of this lemma is also succinct as we proceed by applying the `lc_ind_l` lemma to express the linear combination of T^K_{geo} as the sum of the first component and the linear combination of the rest (see Section 5.3.4). Then, by employing the definitions of reference and current nodes outlined in Section 9.1—specifically through Equations (9.2) and (9.3) respectively—we obtain the desired result.

More generally, any point defined relative to the reference simplex will have the same relative position in the current simplex under the affine transformation T^K_{geo}. This means that T^K_{geo} applied to an affine combination of reference vertices equals the same affine combination of the current vertices, when the sum of the coefficients equals 1. In other terms, T^K_{geo} preserves the barycentric coordinates, which represent a point within a simplex as a weighted average of the vertices of the simplex (see Section 5.3.6). This is expressed as follows:

$$\forall \boldsymbol{c} \in \mathbb{R}^{d+1}, \quad \sum_{i=0}^{d} c_i = 1 \Longrightarrow T^K_{\text{geo}}\left(\sum_{i=0}^{d} c_i \hat{\mathbf{v}}_i\right) = \sum_{i=0}^{d} c_i \mathbf{v}_i. \tag{9.14}$$

Here, the condition $\sum_{i=0}^{d} c_i = 1$ ensures that these barycentric coordinates describe a convex combination of the vertices. Note that, if $\forall i \in [0..d], c_i \geq 0$, the barycentric point $\sum_{i=0}^{d} c_i \hat{\mathbf{v}}_i$ lies

inside the simplex $\hat{K}$.

Furthermore, it is important to ensure that the numbering of the nodes $(\mathbf{a}_{\boldsymbol{\alpha}})_{\boldsymbol{\alpha}\in\mathcal{A}_k^d}$ is compatible with that adopted in the geometric finite element. This property is guaranteed if the geometric transformation T_{geo}^K is a $\mathcal{C}^1$-diffeomorphism. In other words, T_{geo}^K must be bijective, with both T_{geo}^K and its inverse, $(T_{\text{geo}}^K)^{-1}$, being continuously differentiable, a property denoted as class $\mathcal{C}^1$. Moreover, as T_{geo}^K is an affine map, it suffices to prove that it is invertible. This holds when the vertices are affinely independent. Note that the $\mathcal{C}^1$ class of T_{geo}^K and $(T_{\text{geo}}^K)^{-1}$ has not been formalized in Coq.

To formally establish that T_{geo}^K is bijective, it is necessary to demonstrate both its surjectivity and injectivity on $\mathbb{R}^d$. These proofs are encapsulated in the following Coq lemmas:

```
Variable vtx_cur : '('R^d)^d.+1.
Hypothesis Hvtx : affine_independent vtx_cur.
Lemma T_geom_surj : ∀(x_cur : 'R^d), ∃ (x_ref : 'R^d), T_geom x_ref = x_cur.
```

Surjectivity ensures that all points in $\mathbb{R}^d$ can be covered by T_{geo}^K. The proof of the lemma is derived from the surjectivity of the reference Lagrange polynomials $(\hat{\mathcal{L}}_i^{d,1})_{i\in[0..d]}$, as demonstrated in Section 9.2. Indeed, applying a fundamental lemma about affine spaces named `affine_independent_generator`, discussed in Section 5.4.3, along with the affine independence of the vertex family `vtx_cur`, we confirm that the current vertices are affinely generating. This means that every point $\mathbf{x}$ in the space $\mathbb{R}^d$ can be expressed as an affine combination (i.e., $\forall \mathbf{x} \in \mathbb{R}^d, \exists L \in \mathbb{R}^{d+1}, \sum_{i=0}^d L_i = 1$ and $\mathbf{x} = \sum_{i=0}^d L_i \mathbf{v}_i$) of the current vertices. Hence, from the lemma `LagPd1_ref_surj`, we conclude that:

$$T_{\text{geo}}^K(\hat{\mathbf{x}}) \overset{\text{def.}}{=} \sum_{i=0}^{d} \hat{\mathcal{L}}_i^{d,1}(\hat{\mathbf{x}})\mathbf{v}_i = \sum_{i=0}^{d} L_i \mathbf{v}_i = \mathbf{x}.$$

□

Concerning the injectivity of T_{geo}^K, we need to prove that no two distinct points in $\mathbb{R}^d$ correspond to the same point.

```
Lemma T_geom_inj : ∀x_ref y_ref: 'R^d, T_geom y_ref = T_geom x_ref → y_ref = x_ref.
```

The injectivity of T_{geo}^K is confirmed by applying the `affine_independent_lc` lemma (refer to Section 5.4.3). Given an affinely independent family of vertices $(\mathbf{v}_i)_{i\in[0..d]}$, we suppose that $\sum_{i=0}^d \hat{\mathcal{L}}_i^{d,1}(\hat{\mathbf{y}})\mathbf{v}_i = \sum_{i=0}^d \hat{\mathcal{L}}_i^{d,1}(\hat{\mathbf{x}})\mathbf{v}_i$. Since both the sums of $\hat{\mathcal{L}}_i^{d,1}(\hat{\mathbf{y}})$ and $\hat{\mathcal{L}}_i^{d,1}(\hat{\mathbf{x}})$ are equal to 1, as deduced from the lemma `LagPd1_ref_sum_1` detailed in Section 9.2, it follows that $\hat{\mathcal{L}}_i^{d,1}(\hat{\mathbf{y}}) = \hat{\mathcal{L}}_i^{d,1}(\hat{\mathbf{x}})$ for all $i \in [0..d]$. Consequently, from the injectivity of the Lagrange polynomials, as outlined in the `LagPd1_ref_inj` lemma of Section 9.2, we conclude that $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ are identical in $\mathbb{R}^d$.
□

From the surjectivity and injectivity of T_{geo}^K, we conclude that:

```
Lemma T_geom_bij : bijective T_geom.
```

Having established that the transformation mapping T_{geo}^K is bijective, we can define the inverse function $(T_{\text{geo}}^K)^{-1}$ through the functional inverse `f_inv`, as detailed in Section 5.1.3.

```
Definition T_geom_inv : 'R^d → 'R^d := f_inv T_geom_bij.
```

The bijectivity of T_{geo}^K implies a specific property: when composed with its inverse $(T_{\text{geo}}^K)^{-1}$, the result is the identity function on the respective spaces. This can be expressed in two parts:

$$\forall \mathbf{x} \in \mathbb{R}^d,\ T^K_{\text{geo}} \circ (T^K_{\text{geo}})^{-1}(\mathbf{x}) = \mathbf{x} \quad (\textit{right identity}) \tag{9.15}$$

$$\forall \hat{\mathbf{x}} \in \mathbb{R}^d,\ (T^K_{\text{geo}})^{-1} \circ T^K_{\text{geo}}(\hat{\mathbf{x}}) = \hat{\mathbf{x}} \quad (\textit{left identity}) \tag{9.16}$$

As T^K_{geo} is a bijective affine function, its inverse, $(T^K_{\text{geo}})^{-1}$, naturally inherits several fundamental properties. More specifically, stemming from the affine properties of T^K_{geo}, expressed in Equation (9.13), and considering that the transformation is bijective, $(T^K_{\text{geo}})^{-1}$ also possesses affine characteristic. This is described by $(T^K_{\text{geo}})^{-1}(\mathbf{x}) = (\boldsymbol{J}^K_{\text{geo}})^{-1}(\mathbf{x} - b_K)$, where $\boldsymbol{J}^K_{\text{geo}}$ represents a Jacobian matrix in $\mathbb{R}^{d,d}$, a detail not formalized in Coq (see e.g. [42] and [36, Section 8.1, p.89]).

This ensures that $(T^K_{\text{geo}})^{-1}$ is an affine transformation due to the linearity of the inverse matrix operation and the translation adjustment.

```
Lemma T_geom_inv_am : aff_map T_geom_inv.
```

Additionally, if T^K_{geo} maps nodes, vertices, and linear combinations of vertices from a reference element $\hat{K}$ to a current element K, then $(T^K_{\text{geo}})^{-1}$ must necessarily map them back from K to $\hat{K}$. This reciprocal mapping ensures the following:

$$\forall \boldsymbol{c} \in \mathbb{R}^{d+1},\quad \sum_{i=0}^{d} c_i = 1 \quad \Longrightarrow \quad (T^K_{\text{geo}})^{-1}\left(\sum_{i=0}^{d} c_i \mathbf{v}_i\right) = \sum_{i=0}^{d} c_i \hat{\mathbf{v}}_i. \tag{9.17}$$

When $d = 1$, the current element K, as outlined in Equation (6.6) in Section 6.2.2, corresponds to the interval $[a, b]$ in $\mathbb{R}$. The two vertices (v_0, v_1) of this interval are affinely independent (i.e., $v_0 \neq v_1$), and its interior, $\overset{\circ}{K}$, is nonempty. The geometric mapping and its inverse associated with these vertices, illustrated in Figure 9.5, are specified as follows:

$$\forall \hat{x} \in \mathbb{R},\quad T^K_{\text{geo}} : \mathbb{R} \ni \hat{x} \longmapsto (v_1 - v_0)\ \hat{x} + v_0 \in \mathbb{R}. \tag{9.18}$$

$$\forall x \in \mathbb{R},\quad (T^K_{\text{geo}})^{-1} : \mathbb{R} \ni x \longmapsto \frac{x - v_0}{v_1 - v_0} \in \mathbb{R} \quad \text{if } v_0 \neq v_1. \tag{9.19}$$

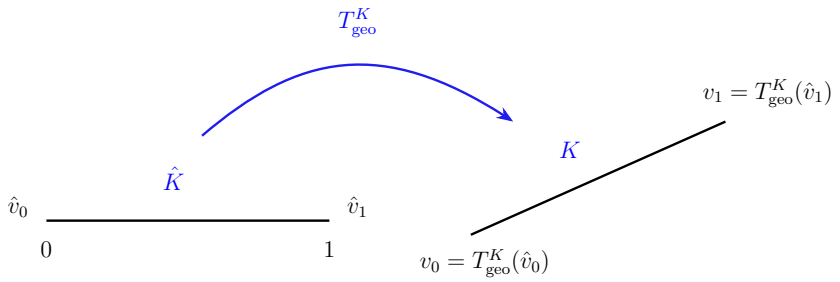

Figure 9.5: Representation of a geometric transformation, denoted as T^K_{geo}, in dimension 1 from the reference geometric element $\hat{K}$ to a current geometric element K. The left side of the figure represents $\hat{K}$, which is a line segment between two reference points $\hat{v}_0$ and $\hat{v}_1$, that correspond to the coordinates 0 and 1, respectively, on this line segment. The right side of the figure represents K, which is also a line segment between two points v_0 and v_1. These later are the images of $\hat{v}_0$ and $\hat{v}_1$ under the transformation T^K_{geo}.

9.4 Building Current Simplicial Finite Elements From Reference Finite Element

Constructing current finite elements from a reference finite element has several significant advantages and purposes. Using a reference finite element simplifies the mathematical formulation and the computational implementation since elements in the mesh may have different shapes and sizes, especially in complex geometries. This means that the basis functions, which are used to approximate solutions, for instance, the Lagrange polynomial bases defined earlier in Section 9.2, need to be defined and precomputed only once on the reference element rather than directly on each individual current finite element. If basis functions were to be defined individually for each current element, each element would require a unique set of basis functions based on its specific geometry. This approach would complicate the implementation and increase the computational cost significantly. Instead, by defining these functions on a simple reference element, they can be easily and efficiently transformed to any other current element in the mesh of the problem domain using a well-defined mapping transformation (refer to Section 9.3). Additionally, operations such as integration, necessary for assembling system matrices like stiffness and mass matrices, and differentiation needed for evaluating gradients of functions (refer to Section 6.2.3), are more straightforward on a reference element. The reference element typically has a regular shape (like a unit square or triangle), which simplifies these mathematical operations. For example, quadrature formulas can be readily applied to standard shapes, and these results are then transformed to fit the actual elements.

Let Ω be a domain in $\mathbb{R}^d$ with $d \geq 1$. Following Section 6.2.2, $\mathcal{T}_h = (K_m)_{m\in[1..N_c]}$ represents a mesh of Ω, where each K_m is a simplex (a generalization of triangles and tetrahedra to arbitrary dimensions) and N_c is the total number of cells in the mesh. Constructing a family of finite elements on the cells of $\mathcal{T}_h$ involves selecting a reference finite element $(\hat{K}, \hat{\mathcal{P}}, \hat{\Sigma})$.

9.4.1 Reference Finite Element

We start by choosing a reference finite element denoted by $(\hat{K}, \hat{\mathcal{P}}, \hat{\Sigma})$. This reference finite element is declared in Coq as a variable `FE_ref` of type `FE` (see Section 7.1), which contains the geometric shape (here a simplex), the polynomial space, and the family of degrees of freedom. More specifically, we use the elements of the record `FE` to define the reference finite element as a triplet as follows:

(i) $\hat{K}$ is the reference geometric element, defined as the convex hull of the $d+1$ reference vertices $(\hat{\mathbf{v}}_i)_{i\in[0..d]}$ defined in Coq as:

```
Let nvtx := S (d FE_ref).
Let vtx_ref : 'R^(d FE_ref))^nvtx := castF _ (vtx FE_ref).
Let K_geom_ref : 'R^(d FE_ref) → Prop := convex_envelop vtx_ref.
```

Here, `Let` is used for local definitions limited to the section enclosed by a begin/end pair, and `nvtx` is the number of reference vertices in the reference geometric element $\hat{K}$.

We assume that `FE_ref` forms indeed a reference finite element by the following hypothesis:

```
Hypothesis FE_ref_is_ref : ∀i : 'I_nvtx, vtx_ref i = vtx_simplex_ref (d FE_ref) i.
```

This hypothesis ensures that the vertices `vtx_ref` are the vertices of the reference simplex `vtx_simplex_ref` in dimension `d`, as detailed in Section 9.1.1.

(ii) $\hat{\mathcal{P}}$ is the approximation space defined on $\hat{K}$ with finite dimension.

```
Let P_approx_ref : FRd (d FE_ref) → Prop := P_approx FE_ref.
Let P_approx_has_dim_ref := P_approx_has_dim FE_ref.
```

(iii) $\hat{\Sigma}$ is the family of degrees of freedom $(\hat{\sigma}_0, \ldots, \hat{\sigma}_{n_{\text{dof}}-1})$ defined in Coq as:

```
Let Sigma_ref : '(FRd (d FE_ref) → R)^(ndof FE_ref) := Sigma FE_ref.
```

Here, `ndof` represents the number of degrees of freedom associated with the finite element `FE_ref`. We assume that both the spatial dimension `d` and the number of degrees of freedom `ndof` are positive, denoted by the constraints `d_pos` and `ndof_pos` respectively in Coq. Each of the linear forms is a function from the approximation space $\hat{\mathcal{P}}$ to $\mathbb{R}$ that satisfies the linearity and the unisolvence property detailed in Section 6.4, ensuring the shape functions are uniquely determined.

```
Let Sigma_lm_ref := Sigma_lm FE_ref.
Let unisolvence_ref := unisolvence FE_ref.
```

The shape functions of the reference finite element are designated by $\{\hat{\theta}_0, \ldots, \hat{\theta}_{n_{\text{dof}}-1}\} \in (\mathcal{F}(\mathbb{R}^d, \mathbb{R}))^{n_{\text{dof}}}$ (see Section 7.2).

```
Let shape_fun_ref : '(FRd (d FE_ref))^(ndof FE_ref) := shape_fun FE_ref.
```

We define the reference local interpolation operator (see Section 7.3) as:

$$\mathcal{I}_{\hat{K}} : \mathcal{F}(\mathbb{R}^d, \mathbb{R}) \ni \hat{v} \longmapsto \sum_{i=0}^{n_{\text{dof}}-1} \hat{\sigma}_i(\hat{v})\ \hat{\theta}_i \ \in \hat{\mathcal{P}}. \tag{9.20}$$

The Coq formalization of this concept is straightforward,

```
Let local_interp_ref : FRd (d FE_ref) → FRd (d FE_ref) :=
  fun v ⇒ local_interp FE_ref v.
```

9.4.2 Generating the Current Finite Elements

To generate a current finite element on a cell $K \in \mathcal{T}_h$, we need to transform functions defined on $\hat{K}$ to functions defined on K. First, we consider the functional space $\mathcal{F}(\mathbb{R}^d, \mathbb{R})$ on K. We define an isomorphism in Coq as follows:

```
Variable vtx_cur : '('R^(d FE_ref))^nvtx.
Hypothesis Hvtx : affine_independent vtx_cur.
Definition cur_to_ref : FRd (d FE_ref) → FRd (d FE_ref) :=
  fun (g_cur : FRd (d FE_ref)) (x_ref : 'R^(d FE_ref)) ⇒ g_cur (T_geom vtx_cur x_ref).
```

Here, `nvtx` is the total number of current vertices in the current geometric element K. The isomorphism property is detailed in Section 5.3.3. `cur_to_ref` is a linear function that transforms a function `g_cur` defined on the current geometric element to a function defined on the reference geometric element through the geometric transformation `T_geom`. Informally, this is expressed as:

$$\psi^K : \mathcal{F}(\mathbb{R}^d, \mathbb{R}) \ni g \longmapsto g \circ T^K_{\text{geo}} \ \in \mathcal{F}(\mathbb{R}^d, \mathbb{R}). \tag{9.21}$$

The function ψ^K is defined on the space $\mathcal{F}(\mathbb{R}^d, \mathbb{R})$, which represents the set of all functions from $\mathbb{R}^d$ to $\mathbb{R}$, and T^K_{geo} is the geometric transformation sending $\hat{K}$ into K (see section 9.3).

Inversely, we define the inverse of the function ψ^K which transforms functions defined on $\hat{K}$ into functions that operate on K. This is achieved through the inverse geometric transformation $(T^K_{\text{geo}})^{-1}$ as follows:

```
Definition ref_to_cur : FRd (d FE_ref) → FRd (d FE_ref) :=
  fun (g_ref : FRd (d FE_ref)) (x_cur : 'R^(d FE_ref)) ⇒
      g_ref (T_geom_inv vtx_cur Hvtx x_cur).
```

here, the hypothesis `Hvtx` is necessary because the bijectivity of T^K_{geo}, and consequently the existence of its inverse $(T^K_{\text{geo}})^{-1}$, indeed relies on the affine independence of the vertices `vtx_cur` (see also Section 5.1.3). This function is expressed mathematically as:

$$(\psi^K)^{-1} : \mathcal{F}(\mathbb{R}^d, \mathbb{R}) \ni \hat{g} \longmapsto \hat{g} \circ (T^K_{\text{geo}})^{-1} \ \in \mathcal{F}(\mathbb{R}^d, \mathbb{R}). \tag{9.22}$$

The function $(\psi^K)^{-1}$ is linear, similar to ψ^K. It is designed to be the inverse of ψ^K. This means that applying $(\psi^K)^{-1}$ after ψ^K yields the identity function and vice versa.

(i) first identity: $\forall \hat{g} \in \mathcal{F}(\mathbb{R}^d, \mathbb{R}),\ \forall \hat{\mathbf{x}} \in \mathbb{R}^d$ we have

$$\psi^K((\psi^K)^{-1}(\hat{g}))(\hat{\mathbf{x}}) = (\psi^K)^{-1}(\hat{g})(T^K_{\text{geo}}(\hat{\mathbf{x}})) = \hat{g}((T^K_{\text{geo}})^{-1}(T^K_{\text{geo}}(\hat{\mathbf{x}}))) = \hat{g}(\hat{\mathbf{x}}). \tag{9.23}$$

This relies on the bijective left identity (9.16) of the geometric mapping T^K_{geo} established in Section 9.3, which ensures that $(T^K_{\text{geo}})^{-1}$ and T^K_{geo} are inverse.

(ii) second identity: $\forall g \in \mathcal{F}(\mathbb{R}^d, \mathbb{R}),\ \forall \mathbf{x} \in \mathbb{R}^d$ we have

$$(\psi^K)^{-1}(\psi^K(g))(\mathbf{x}) = \psi^K(g)((T^K_{\text{geo}})^{-1}(\mathbf{x})) = g(T^K_{\text{geo}}((T^K_{\text{geo}})^{-1}(\mathbf{x}))) = g(\mathbf{x}). \tag{9.24}$$

This uses the bijective right identity (9.15) of the geometric mapping T^K_{geo} established in Section 9.3.

With all the necessary components defined for the reference finite element `FE_ref`, we are now ready to construct and formalize the current finite element $(K, \mathcal{P}_K, \Sigma_K)$, denoted in Coq as `FE_cur`. This process begins with declaring a set of affinely independent vertices, $(\mathbf{v}_i)_{i\in[0..d]}$. Mathematically, this triple $(K, \mathcal{P}_K, \Sigma_K)$ is represented as:

$$\begin{cases} K \overset{\text{def.}}{=} T^K_{\text{geo}}(\hat{K}) \in \mathcal{T}_h, \\ \mathcal{P}_K \overset{\text{def.}}{=} (\psi^K)^{-1}(\hat{\mathcal{P}}), \\ \Sigma_K \overset{\text{def.}}{=} \{(\sigma_{K,i})_{i\in[0..n_{\text{dof}}-1]};\ \sigma_{K,i} : \mathcal{F}(\mathbb{R}^d, \mathbb{R}) \ni p \longmapsto \hat{\sigma}_i(\psi^K(p)) \in \mathbb{R}^{n_{\text{dof}}}\}. \end{cases} \tag{9.25}$$

To confirm that this triple constitutes a finite element, we must demonstrate the unisolvence property that will be established further by the lemma `unisolvence_cur`. Before delving into this proof, let us formalize and explain each component of the triple (9.25) in detail.

Current Geometric Element

The current geometric element K is constructed as a convex hull of affinely independent current vertices.

```
Variable vtx_cur : '('R^(d FE_ref))^nvtx.
Definition K_geom_cur := convex_envelop vtx_cur.
```

The definition of the convex envelope is given in Section 6.2.2. The geometric element K can also be constructed as the image of $\hat{K}$ under the transformation T^K_{geo} in the mesh $\mathcal{T}_h$ (see Section 9.3). This is formalized in Coq as:

```
Lemma T_geom_image : K_geom_cur = image T_geom K_geom_ref.
```

Current Approximation Space

The polynomial space $\mathcal{P}_K$, denoted in Coq as `P_approx_cur`, is defined as the pre-image of $\hat{\mathcal{P}}$ under the transformation ψ^K in the current geometric element K. Formally:

```
Definition P_approx_cur : FRd (d FE_ref) → Prop := preimage cur_to_ref P_approx_ref.
```

The construction of `P_approx_cur` uses the concept of pre-image as detailed in Section 5.1.2. This definition establishes the following equivalence:

$$g \in \mathcal{P}_K \iff \psi^K(g) \in \hat{\mathcal{P}}. \tag{9.26}$$

Conversely, any function g that belongs to $\hat{\mathcal{P}}$ will map under the transformed counterpart via $(\psi^K)^{-1}$ to a function in $\mathcal{P}_K$ through the following lemma:

```
Lemma P_approx_cur_correct : ∀g_ref : FRd (d FE_ref),
  P_approx_ref g_ref → P_approx_cur (ref_to_cur g_ref).
```

The proof of this implication follows directly from the inverse property stated in Equation (9.23). Specifically, this means that for any function $\hat{g} \in \mathcal{F}(\mathbb{R}^d, \mathbb{R})$ in the reference approximation space $\hat{\mathcal{P}}$, we have $\hat{g} = \psi^K((\psi^K)^{-1}(\hat{g})) \in \hat{\mathcal{P}}$. This implies that $(\psi^K)^{-1}(\hat{g}) \in (\psi^K)^{-1}(\hat{\mathcal{P}})$. Hence, the desired result follows directly from the definition of $\mathcal{P}_K$ as the pre-image of $\hat{\mathcal{P}}$ under ψ^K.

As observed from the Equation (9.21), the transformation function ψ^K is bijective, implying a one-to-one correspondence between approximation spaces $\mathcal{P}_K$ and $\hat{\mathcal{P}}$.

```
Lemma cur_to_ref_bijS : bijS P_approx_cur P_approx_ref cur_to_ref.
```

where, `bijS` refers to the bijectivity on the subsets $\mathcal{P}_K$ and $\hat{\mathcal{P}}$ (see Section 5.1.4). The bijectivity of ψ^K is essential for proving that $\mathcal{P}_K$ is a finite dimensional space. To ensure this property, we use the lemma `bijS_ex`, also detailed in Section 5.1.4, which establishes that if an inverse function exists and satisfies four specific criteria, then ψ^K qualifies as a bijection. Specifically, by designating $(\psi^K)^{-1}$ as the inverse function to ψ^K, we first proceed by explicitly demontrating that $\psi^K(\mathcal{P}_K) = \hat{\mathcal{P}}$ and $(\psi^K)^{-1}(\hat{\mathcal{P}}) = \mathcal{P}_K$, which are affirmed by Equation (9.26) and lemma `P_approx_cur_correct`, respectively. Furthermore, we confirm that the compositions of ψ^K and $(\psi^K)^{-1}$ accurately return the original inputs. This confirmation is directly derived from the identity properties of ψ^K and $(\psi^K)^{-1}$, as delineated in Equations (9.23) and (9.24), thereby maintaining the bijectivity of the transformation function ψ^K.

The approximation space $\mathcal{P}_K$ which is defined as the pre-image of $\hat{\mathcal{P}}$ under the transformation ψ^K, is a vector space (see Section 5.3.3). This is derived from the linearity of the function ψ^K, and that $\hat{\mathcal{P}}$ is also a vector space, that maintains properties like closure under scalar multiplication and addition.

```
Lemma P_approx_cur_cms : compatible_ms P_approx_cur.
```

Another important aspect of $\mathcal{P}_K$ is its finite dimensionality (in this context, the dimension of $\mathcal{P}_K$ is the number of the degrees of freedom n_{dof}). This characteristic means that the space can be spanned by a finite number of basis functions. For instance, $\mathcal{P}_K$ might be spanned by polynomials up to a certain degree, such as all linear or quadratic polynomials defined over K.

```
Lemma P_approx_cur_has_dim : has_dim P_approx_cur (ndof FE_ref).
```

The proof of this lemma relies on the established properties discussed in this section. To prove the lemma, it is sufficient to demonstrate the existence of a basis in $\mathcal{P}_K = (\psi^K)^{-1}(\hat{\mathcal{P}})$ consisting of n_{dof} elements, denoted as $(\psi^K)^{-1}(\hat{\theta}_i)$, where $(\hat{\theta}_i)_{i\in[0..n_{\text{dof}}-1]}$ represents the family of reference shape functions from the reference approximation space $\hat{\mathcal{P}}$ (see Section 7.2). Indeed, given that

$\mathcal{P}_K$ is closed under linear operations, as established by the previous lemma `P_approx_cur_cms`, and considering the bijectivity of the function ψ^K as shown by `cur_to_ref_bijS`, together with the fact that the family $(\hat{\theta}_i)_{i\in[0..n_{\text{dof}}-1]}$ forms a basis for $\hat{\mathcal{P}}$, it follows that the space $\mathcal{P}_K$ is a finite dimensional space.

Family of Current Linear Forms

The family of current linear forms Σ contains n_{dof} linear forms acting on functions within the space $\mathcal{P}_K$.

$$\Sigma_K = \{(\sigma_{K,i})_{i\in[0..n_{\text{dof}}-1]};\ \sigma_{K,i} : \mathcal{F}(\mathbb{R}^d,\mathbb{R}) \ni p \longmapsto \hat{\sigma}_i(\psi^K(p)) \in \mathbb{R}^{n_{\text{dof}}}\}. \tag{9.27}$$

This is defined formally as:

```
Definition Sigma_cur : '(FRd (d FE_ref) → R)^(ndof FE_ref) :=
  fun (i : 'I_(ndof FE_ref)) p ⇒ Sigma_ref i (cur_to_ref p).
```

We recall that `Sigma_ref` represents the family $\hat{\Sigma}$ of reference degrees of freedom $(\hat{\sigma}_i)_{i\in[0..n_{\text{dof}}-1]}$ in $\hat{K}$ for the finite element `FE_ref`, as detailed in Section 9.4.1. The family `Sigma_cur` constitutes a basis for the dual space of linear forms $\mathcal{L}(\mathcal{P}_K,\mathbb{R})$. This is not formalized in Coq as it is not necessary for further developments. Furthermore, each function σ_i acts as a linear map on functions within the space `FRd d`, as discussed in Section 5.3.3. This property is characterized as follows:

```
Lemma Sigma_cur_lm : ∀i : 'I_(ndof FE_ref), lin_map (Sigma_cur i).
```

The proof derives straightforwardly from the linearity of the reference linear forms $(\hat{\sigma}_i)_{i\in[0..n_{\text{dof}}-1]}$ in the family $\hat{\Sigma}$.

Last but not least, for the triple (9.25) to qualify as a finite element, it must satisfy the `unisolvence` property.

```
Lemma unisolvence_cur : bijS P_approx_cur fullset (gather Sigma_cur).
```

Here, `bijS` refers to the bijectivity on subsets as discussed in Section 5.1.4, and the function `gather` is used here to transform the family `Sigma_cur` of linear forms of type `'( FRd d → R)^ndof`, into a single function that maps any input from `FRd d` to a vector in `'R^ndof`. Let us prove this lemma.

As discussed in Section 7.1, the unisolvence property establishes a bijection between the approximation space $\mathcal{P}_K$ and $\mathbb{R}^{n_{\text{dof}}}$. Specifically, we show that the mapping $\Phi_{\Sigma_K} : \mathcal{P}_K \ni p \longmapsto (\sigma_{K,i}(p))_{i\in[0..n_{\text{dof}}-1]} \in \mathbb{R}^{n_{\text{dof}}}$ is an isomorphism. To establish this, it suffices to prove the conditions specified in Equations (6.11) and (6.12) of Section 6.4, which state that the linear map Φ_{Σ_K} between the vector spaces $\mathcal{P}_K$ and $\mathbb{R}^{n_{\text{dof}}}$ is bijective if and only if two criteria are met: first, that the dimensions of these two spaces are equal, which is confirmed by Lemma `P_approx_cur_has_dim`, and second, that Φ_{Σ_K} is a linear injective map. The linearity of Φ_{Σ_K} follows from Lemma `Sigma_cur_lm`. To prove injectivity, it must be shown that:

$$\forall p \in \mathcal{P}_K,\ (\forall i \in [0..n_{\text{dof}}-1], \sigma_{K,i}(p) = 0) \implies (p = 0).$$

Consider $p \in \mathcal{P}_K$ such that $\sigma_{K,i}(p) = 0$. From Equation (9.27), it follows that $\hat{\sigma}_i(\psi^K(p)) = 0$. Given that $\psi^K(p) \in \hat{\mathcal{P}}$ and leveraging the unisolvence property of the reference linear forms $\hat{\Sigma}$, as established by `unisolvence_ref` in Section 9.4.1, we deduce that $\psi^K(p) = 0$. Consequently, since the transformation map ψ^K is injective, we conclude that $p = 0$.
□

Consequently, the current finite element, denoted as FE_cur, is constructed as a record using the mk_FE constructor defined in Section 7.1, incorporating all the properties of a finite element within a single value.

```
Definition FE_cur :=
  mk_FE (d FE_ref) (ndof FE_ref) (d_pos FE_ref)
  (ndof_pos FE_ref) (shape FE_ref) (castF _ vtx_cur)
  P_approx_cur P_approx_cur_has_dim
  Sigma_cur Sigma_cur_lm unisolvence_cur.
```

9.4.3 Current Shape Functions and Local Interpolation Operator

From the current finite element FE_cur, we construct the shape functions of the current geometric element $K \in \mathcal{T}_h$ as:

```
Let d_cur := d FE_cur.
Let ndof_cur := ndof FE_cur.
Let shape_fun_cur : '(FRd d_cur)^ndof_cur := shape_fun FE_cur.
```

Here, shape_fun is the shape function defined as a predual basis of $(\sigma_i)_{i\in[0..n_{\text{dof}}-1]}$ detailed in Section 7.2.

The shape functions $\theta_{K,i}$ are derived by applying the inverse of the transformation $(\psi^K)^{-1}$ to the reference shape functions $\hat{\theta}_i$. This function is represented both mathematically and in Coq as follows:

$$\theta_{K,i} = (\psi^K)^{-1}(\hat{\theta}_i), \quad \forall i \in [0..n_{\text{dof}} - 1]. \tag{9.28}$$

```
Lemma shape_fun_cur_correct : ∀i : 'I_(ndof FE_ref),
  shape_fun_cur i = ref_to_cur (shape_fun_ref i).
```

Furthermore, the local interpolation operator $\mathcal{I}_K$ is mathematically defined by the following expression:

$$\mathcal{I}_K : \mathcal{F}(\mathbb{R}^d, \mathbb{R}) \ni v \longmapsto \sum_{i=0}^{n_{\text{dof}}-1} \sigma_{K,i}(v)\ \theta_{K,i}\ \in \mathcal{P}_K. \tag{9.29}$$

The operator transforms v into an approximation function within the space $\mathcal{P}_K$ by linearly combining basis shape functions $\theta_{K,i}$, each weighted by coefficients $\sigma_{K,i}(v)$. In a formal setting, the operator is implemented as:

```
Let local_interp_cur : FRd d_cur → FRd d_cur := fun v ⇒ local_interp FE_cur v.
```

Here, local_interp_cur is an instantiation of $\mathcal{I}_h$ defined in Section 7.3.

An additional property of $\mathcal{I}_K$ is given in the following lemma:

```
Lemma local_interp_cur_ref : ∀v : FRd (d FE_ref),
  local_interp_ref (cur_to_ref v) = cur_to_ref (local_interp_cur v).
```

This lemma asserts that applying the transformation ψ^K to v before interpolating by $\mathcal{I}_{\hat{K}}$ is equivalent to interpolating v first and then applying ψ^K. This property is justified since the function ψ^K is a linear map, which ensures that the transformations and interpolations commute as follows:

$$\mathcal{I}_{\hat{K}} \circ \psi^K(v) = \sum_{i=0}^{n_{\text{dof}}-1} \hat{\sigma}_i(\psi^K(v))\ \hat{\theta}_i = \sum_{i=0}^{n_{\text{dof}}-1} \sigma_{K,i}(v)\ \psi^K(\theta_{K,i}) = \psi^K \circ \mathcal{I}_K(v).$$

9.5 $\mathcal{P}_1^d$ Lagrange Polynomials Basis on a Current Element

In the progression of our discussion on Lagrange polynomials and their application within the finite element method, this section delves into the formulation and properties of the current Lagrange polynomials $(\mathcal{L}_i^{d,1})_{i\in[0..d]}$, when the dimension is $d \geq 1$ and the degree of approximation is $k = 1$. The properties of these polynomials are built upon the concepts of the reference Lagrange polynomials $(\hat{\mathcal{L}}_i^{d,1})_{i\in[0..d]}$ established in the earlier Section 9.2, and the affine transformation mapping T_{geo}^K, along with its inverse, presented in Section 9.3.

Given a family of $d+1$ affinely independent points $(\mathbf{v}_i)_{i\in[0..d]}$ in $\mathbb{R}^d$, the current Lagrange polynomials $(\mathcal{L}_i^{d,1})_{i\in[0..d]}$ are defined as follows:

$$\mathcal{L}_i^{d,1}(\mathbf{x}) \stackrel{\text{def.}}{=} \hat{\mathcal{L}}_i^{d,1} \circ (T_{\text{geo}}^K)^{-1}(\mathbf{x}), \quad \forall i \in [0..d], \forall \mathbf{x} \in \mathbb{R}^d. \tag{9.30}$$

Here is the implementation definition in Coq:

```
Hypothesis Hvtx : affine_independent vtx_cur.
Definition LagPd1_cur (i : 'I_d.+1) (x_cur : 'R^d) :=
  LagPd1_ref d i (T_geom_inv vtx_cur Hvtx x_cur).
```

This definition leads to the property that $\hat{\mathcal{L}}_i^{d,1} = \mathcal{L}_i^{d,1} \circ T_{\text{geo}}^K$, thus establishing a bidirectional mapping between the reference and current elements. Each $\mathcal{L}_i^{d,1}$ is designated as the i-th current Lagrange polynomial of $\mathcal{P}_1^d$, associated with the points $(\mathbf{v}_i)_{i\in[0..d]}$.

The transition from $\hat{\mathcal{L}}^{d,1}$ to $\mathcal{L}^{d,1}$ preserves several properties similar to those established in Section 9.2. More specifically, each current Lagrange polynomial $\mathcal{L}_i^{d,1}$ is defined such that it also equals 1 at the corresponding vertex $\mathbf{v}_i$ and 0 at all other vertices of the current simplex.

```
Lemma LagPd1_cur_kron_vtx : ∀i j : 'I_d.+1, LagPd1_cur j (vtx_cur i) = kronecker i j.
```

Proving this lemma requires the use of the `T_geom_inv` property $(T_{\text{geo}}^K)^{-1}(\mathbf{v}_i) = \hat{\mathbf{v}}_i$ which maps vertices of the current element back to the corresponding vertices in the reference element (see Section 9.3). Consequently, we derive that $\mathcal{L}_j^{d,1}(\mathbf{v}_i) = \hat{\mathcal{L}}_j^{d,1}((T_{\text{geo}}^K)^{-1}(\mathbf{v}_i)) = \hat{\mathcal{L}}_j^{d,1}(\hat{\mathbf{v}}_i) = \delta_{i,j}$. This result is deduced from the Kronecker delta property of the reference Lagrange polynomials, as confirmed by the lemma `LagPd1_ref_kron_vtx` in Section 9.2.
□

As when $k = 1$, the nodes coincide with the vertices, we obviously have the same result for the current nodes (refer to Section 9.1.3).

```
Lemma LagPd1_cur_kron_node : ∀i j : 'I_d.+1
    LagPd1_cur j (node_cur d 1 vtx_cur i) = kronecker i j.
```

Additionally, within a simplex defined by affinely independent vertices $(\mathbf{v}_i)_{i\in[0..d]}$ in $\mathbb{R}^d$, any point $\mathbf{x} \in \mathbb{R}^d$ can be uniquely decomposed as an affine combination of these vertices, with coefficients derived from the current Lagrange polynomials $(\mathcal{L}_i^{d,1})_{i\in[0..d]}$ applied to $\mathbf{x}$. Specifically, this decomposition is expressed as:

$$\mathbf{x} = \sum_{i=0}^{d} \mathcal{L}_i^{d,1}(\mathbf{x})\mathbf{v}_i \quad \text{and} \quad \sum_{i=0}^{d} \mathcal{L}_i^{d,1}(\mathbf{x}) = 1. \tag{9.31}$$

These coefficients can be seen as the *barycentric coordinates* of $\mathbf{x}$ in the simplex (see Section 5.3.6), constrained by the condition $\sum_{i=0}^{d} \mathcal{L}_i^{d,1} = 1$. This condition is verified in Coq through the lemma `LagPd1_ref_sum_1` from Section 9.2. Moreover, if $\mathcal{L}_i^{d,1}(\mathbf{x}) \geq 0$ for all $i \in [0..d]$, then any point $\mathbf{x} \in \mathbb{R}^d$ will be positioned either inside or on the boundary of the

simplex K.

Another essential attribute of the current Lagrange polynomial family $(\mathcal{L}_i^{d,1})_{i\in[0..d]}$ is their role as a basis for the approximation space $\mathcal{P}_1^d$. This implies that any polynomial within this space can be uniquely represented by a linear combination of these polynomials. To establish this, the polynomials $(\mathcal{L}_i^{d,1})_{i\in[0..d]}$ must fulfill two requirements, first to be linearly independent and second to span the entire polynomial space $\mathcal{P}_1^d$.

The first requirement addresses their linear independence. The current Lagrange polynomials $(\mathcal{L}_i^{d,1})_{i\in[0..d]}$ maintain their linear independence when transformed from the current geometric element through the affine transformation T_{geo}^K. This is formally stated through this lemma as follows:

```
Variable vtx_cur : '('R^d)^d.+1.
Hypothesis Hvtx : affine_independent vtx_cur.
Lemma LagPd1_cur_lin_indep : lin_indep LagPd1_cur.
```

Establishing this property is succinct, it involves a change of variable using the left identity property of T_{geo}^K as outlined in Equation (9.16). Then, by invoking the linear independence of the reference Lagrange polynomials, verified by the lemma `LagPd1_ref_lin_indep` detailed earlier in Section 9.2, we confirm the linear independence of the current Lagrange polynomial $(\mathcal{L}_i^{d,1})_{i\in[0..d]}$.
□

The second requirement is that the polynomials $(\mathcal{L}_i^{d,1})_{i\in[0..d]}$ span the approximation space $\mathcal{P}_1^d$, allowing any polynomial within this space to be expressed as a linear combination of $\mathcal{L}^{d,1}$ polynomials associated with an affinely independent family of vertices `vtx_cur`. This is formally captured by:

```
Variable vtx_cur : '('R^d)^d.+1.
Hypothesis Hvtx : affine_independent vtx_cur.
Lemma Pd1_lin_span_LagPd1_cur : Pdk d 1 = lin_span (LagPd1_cur vtx_cur Hvtx).
```

The proof proceeds by applying the previously established lemma `Pdk_am_compose_basis` from Section 8.2.3, which demonstrates that composing the basis $(\hat{\mathcal{L}}_i^{d,1})_{i\in[0..d]}$, verified as a basis by the lemma `LagPd1_ref_basis` in Section 9.2, with the affine bijective map $(T_{\text{geo}}^K)^{-1}$, spans the polynomial space $\mathcal{P}_1^d$. The affine bijective properties of the $(T_{\text{geo}}^K)^{-1}$ map are derived from the lemmas `T_geom_inv_am` and `f_inv_bij`, detailed in Sections 9.3 and 5.1.3, respectively.
□

Consequently, the current Lagrange polynomial family $(\mathcal{L}_i^{d,1})_{i\in[0..d]}$ form a basis for the approximation space $\mathcal{P}_1^d$ implemented in Coq as follows:

```
Variable vtx_cur : '('R^d)^d.+1.
Hypothesis Hvtx : affine_independent vtx_cur.
Lemma LagPd1_cur_basis : basis (Pdk d 1) (LagPd1_cur vtx_cur Hvtx).
```

9.6 $\mathcal{P}_k^1$ Lagrange Polynomial Bases on a Segment

This section expands our discussion onto the specifics of $\mathcal{P}_k^1$ Lagrange basis polynomials (see [35, p.8]), specifically within the polynomial space $\mathcal{P}_k^1$ when $d = 1$ and the polynomial degree $k \geq 1$. The primary objective of these polynomials, denoted as $(\mathcal{L}_i^{1,k})_{i\in[0..k]}$, consists in approximating functions by constructing polynomials that pass through a family of $k+1$ interpolation nodes a_i in $\mathbb{R}$, which are previously defined by Equation (9.3) in Section 9.1.2. More specifically, when

$d = 1$, the mathematical formulation of these nodes is defined as follows:

Let (v_0, v_1) a pair of affinely independent vertices in $\mathbb{R}$, and $(a_i)_{i\in[0..k]}$ represent $k+1$ distinct points in $\mathbb{R}$, where each node a_i is computed as:

$$a_i = \begin{cases} \dfrac{v_0 + v_1}{2} & \text{for } i = 0 \\ v_0 + \dfrac{i}{k}(v_1 - v_0) & \forall i \in [0..k]. \end{cases}$$

Then, the Lagrange polynomial bases denoted by `LagP1k_cur` in Coq, and visually depicted in Figure 9.6, are mathematically defined as follows:

$$\forall i \in [0..k], \forall x \in \mathbb{R}, \quad \mathcal{L}_i^{1,k}(x) \stackrel{\text{def.}}{=} \prod_{j=0, j\neq i}^{k} \frac{x - a_j}{a_i - a_j}. \tag{9.32}$$

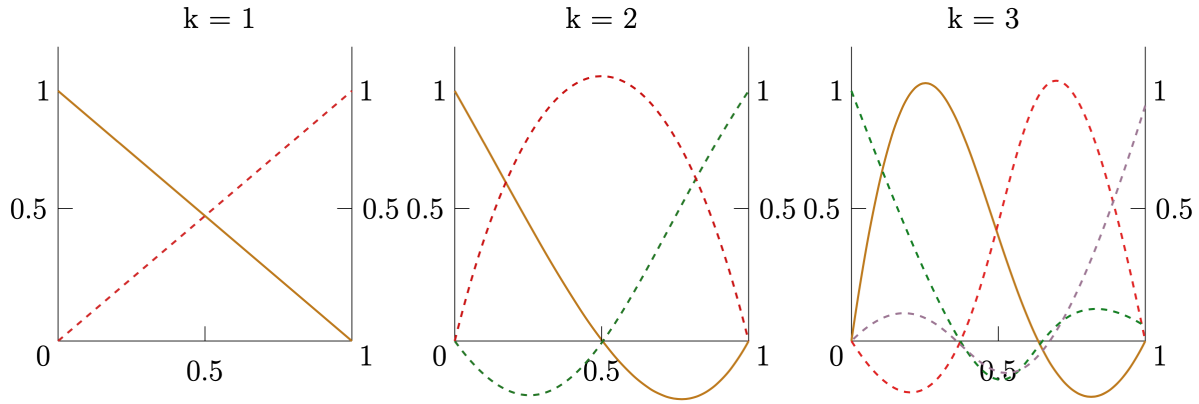

Figure 9.6: Graphical representation of Lagrange interpolation polynomials $(\mathcal{L}_i^{1,k})_{i\in[0..k]}$ to different values of $k \in [1..3]$, over a single variable $x \in \mathbb{R}$. For $k = 1$: There are two polynomials corresponding to current Lagrange nodes at a_0 and a_1. The solid line represents $\mathcal{L}_0^{1,1}(x)$ and the dashed line represents $\mathcal{L}_1^{1,1}(x)$. Each polynomial equals 1 at its respective node and 0 at the other node, forming linear polynomials. For $k = 2$: There are three polynomials, corresponding to three Lagrange nodes. The colors differentiate the basis polynomials $\mathcal{L}_0^{1,2}(x)$, $\mathcal{L}_1^{1,2}(x)$, and $\mathcal{L}_2^{1,2}(x)$. Each polynomial is 1 at its respective node and 0 at the others. The shapes of the polynomials are now quadratic. For $k = 3$: There are four polynomials (four nodes). Each plot here is a cubic polynomial. Again, each polynomial is 1 at its node and 0 at all others.

The Lagrange polynomials exhibit notable properties, especially the fact that the degree of each $\mathcal{L}_i^{1,k}$ polynomial is precisely k. This directly stems from the definition of the polynomial as outlined in Equation (9.32). By definition, each polynomial $\mathcal{L}_i^{1,k}$ is uniquely associated with one interpolation node a_i. It evaluates to 1 at node a_i and to zero at all other nodes. This relationship is represented in the following lemma:

```
Lemma LagP1k_cur_kron_node_alt : ∀(i j : 'I_(pbinom 1 k).+1) (vtx_cur : 'R^{2,1}),
  affine_independent vtx_cur →
  LagP1k_cur vtx_cur j (node_cur 1 k vtx_cur i) = kronecker i j.
```

Let us verify this lemma. Based on the definition of the current polynomials through the Equation (9.32) when evaluated at the current nodes a_i, we demonstrate that

$$\prod_{\ell=0, \ell\neq j}^{k} \frac{a_i - a_\ell}{a_j - a_\ell} = \delta_{i,j}.$$

We conduct a case analysis on whether the indices i and j are identical or not. We note that $a_j \neq a_\ell$ when $\ell \neq j$.

(i) Case 1: $i = j$
Here, since i equals j, we refer to the `kronecker_is_1` lemma from Section 5.3.5, which asserts that the Kronecker delta evaluates to 1. Consequently, the equation simplifies, as confirmed by the `prod_R_one_compat` lemma (see Section 5.3.2), indicating that a product over a family of ones results in 1.

(ii) Case 2: $i \neq j$
With i differing from j, we use the `kronecker_is_0` lemma, setting the Kronecker delta to 0. The expression then reduces to zero, a result validated by the `prod_R_zero` lemma (see Section 5.3.2), when there exists a zero in the family. Thus, the proof is achieved. □

As a direct consequence of the Kronecker delta property, we deduce the linear independence of the family of Lagrange polynomial basis $(\mathcal{L}_i^{1,k})_{i\in[0..k]}$. This step is important for establishing that these polynomials form a basis for the space of polynomial functions $\mathcal{P}_k^1$, which we aim to demonstrate at the end of this section.

```
Lemma LagP1k_cur_lin_indep : ∀(vtx_cur : 'R^{2,1}),
  affine_independent vtx_cur → lin_indep (LagP1k_cur vtx_cur).
```

In simpler terms, our goal is to show that for any coefficients $(c_j)_{j\in[0..k]} \in \mathbb{R}$, if $\sum_{j=0}^{k} c_j \mathcal{L}_j^{1,k}(x) = 0$ for all $x \in \mathbb{R}$, then each coefficient c_j must be zero. By evaluating the Lagrange polynomial $(\mathcal{L}_j^{1,k})_{j\in[0..k]}$ at each node $a_i \in \mathbb{R}$ for $i \in [0..k]$, and applying the previously established lemma `LagP1k_cur_kron_nodes_alt`, we achieve the desired result.

In this part of the section, we focus on demonstrating that the current Lagrange polynomials $(\mathcal{L}_i^{1,k})_{i\in[0..k]}$ indeed constitute a basis for the space $\mathcal{P}_k^1$. To achieve this, we initiate by constructing the family of reference Lagrange polynomials, denoted $(\hat{\mathcal{L}}_i^{1,k})_{i\in[0..k]}$. This involves detailing the necessary properties of these reference polynomials. Using these properties, we will then systematically derive the results confirming that the current Lagrange polynomials serve as a basis for the polynomial space $\mathcal{P}_k^1$.

The reference Lagrange polynomials are obtained by evaluating the current Lagrange polynomials at the reference vertices $(\hat{\mathbf{v}}_i)_{i\in\{0,1\}}$, discussed in Section 9.1.1, as follows:

```
Definition LagP1k_ref : ('R^1 → R)^((pbinom 1 k).+1) :=
  LagP1k_cur (vtx_simplex_ref 1).
```

These reference polynomials belong to the polynomial space $\mathcal{P}_k^1$. From this definition, we notice that the properties of the reference Lagrange polynomials $(\hat{\mathcal{L}}_i^{1,k})_{i\in[0..k]}$ are straightforwardly derived from those of the corresponding current polynomials $(\mathcal{L}_i^{1,k})_{i\in[0..k]}$. Specifically, at each interpolation node $\hat{a}_i$, the polynomial $\hat{\mathcal{L}}_i^{1,k}$ takes a value of 1, and 0 at all other nodes. This behavior is encapsulated by the following code snippet:

```
Lemma LagP1k_ref_kron_node : ∀i j : 'I_(pbinom 1 k).+1,
  LagP1k_ref j (node_ref 1 k i) = kronecker i j.
```

Moreover, the reference Lagrange polynomials $(\hat{\mathcal{L}}_i^{1,k})_{i\in[0..k]}$ are linearly independent, a conclusion drawn from the linear independence of the current polynomials $(\mathcal{L}_i^{1,k})_{i\in[0..k]}$, as supported by the lemma `LagP1k_cur_lin_indep`.

```
Lemma LagP1k_ref_lin_indep : lin_indep LagP1k_ref.
```

Given that $\mathcal{P}^1_k$ has a finite dimension of $k+1$, and that $(\hat{\mathcal{L}}^{1,k}_i)_{i\in[0..k]}$ are linearly independent belonging to $\mathcal{P}^1_k$, we apply the lemma `lin_indep_basis`, discussed in Section 5.4.2, to affirm that $(\hat{\mathcal{L}}^{1,k}_i)_{i\in[0..k]}$ indeed form a basis for the space $\mathcal{P}^1_k$.

```
Lemma LagP1k_ref_basis : basis (Pdk 1 k) (LagP1k_ref k).
```

The definition of $\hat{\mathcal{L}}^{1,k}$ allows us to explore the relationship between the current and reference Lagrange polynomials. Specifically, each current Lagrange polynomial $(\mathcal{L}^{d,1}_i)_{i\in[0..k]}$ can be expressed at any point as follows:

$$\mathcal{L}^{1,k}_i(x) \overset{\text{def.}}{=} \hat{\mathcal{L}}^{1,k}_i \circ (T^K_{\text{geo}})^{-1}(x), \quad \forall i \in [0,1],\ \forall x \in \mathbb{R}. \tag{9.33}$$

This equation is established relying on the results of the geometric transformations T^K_{geo} described in Section 9.3 as follows:

$$\hat{\mathcal{L}}^{1,k}_i \circ (T^K_{\text{geo}})^{-1}(x) = \prod_{j=0,j\neq i}^{k} \frac{(T^K_{\text{geo}})^{-1}(x) - \hat{a}_j}{\hat{a}_i - \hat{a}_j}.$$

Setting $\hat{x} = (T^K_{\text{geo}})^{-1}(x)$, where $(T^K_{\text{geo}})^{-1} : x \longmapsto \dfrac{x - v_0}{v_1 - v_0}$ (refer to Section 9.3), and multiplying both terms of the quotient by $(v_1 - v_0 \neq 0)$, where (v_0, v_1) is a pair of affinely independent points in $\mathbb{R}$, we get

$$\begin{aligned}
\hat{\mathcal{L}}^{1,k}_i \circ (T^K_{\text{geo}})^{-1}(x) &= \prod_{j=0,j\neq i}^{k} \frac{(\hat{x} - \hat{a}_j)(v_1 - v_0)}{(\hat{a}_i - \hat{a}_j)(v_1 - v_0)}, \\
&= \prod_{j=0,j\neq i}^{k} \frac{(\hat{x}(v_1 - v_0) + v_0) - (\hat{a}_j(v_1 - v_0) + v_0)}{(\hat{a}_i(v_1 - v_0) + v_0) - (\hat{a}_j(v_1 - v_0) + v_0)}.
\end{aligned}$$

Using the Equation (9.18) for $d = 1$, we obtain

$$\begin{aligned}
\hat{\mathcal{L}}^{1,k}_i \circ (T^K_{\text{geo}})^{-1}(x) &= \prod_{j=0,j\neq i}^{k} \frac{T^K_{\text{geo}}(\hat{x}) - T^K_{\text{geo}}(\hat{a}_j)}{T^K_{\text{geo}}(\hat{a}_i) - T^K_{\text{geo}}(\hat{a}_j)} \\
&= \prod_{j=0,j\neq i}^{k} \frac{T^K_{\text{geo}}((T^K_{\text{geo}})^{-1}(x)) - T^K_{\text{geo}}((T^K_{\text{geo}})^{-1}(a_j))}{T^K_{\text{geo}}((T^K_{\text{geo}})^{-1}(a_i)) - T^K_{\text{geo}}((T^K_{\text{geo}})^{-1}(a_j))}.
\end{aligned}$$

Finally, using the bijection right identity of T^K_{geo} shown in Equation (9.15), we validate our desired result

$$\hat{\mathcal{L}}^{1,k}_i \circ (T^K_{\text{geo}})^{-1}(\mathbf{x}) = \prod_{j=0,j\neq i}^{k} \frac{(x - a_j)}{(a_i - a_j)} = \mathcal{L}^{1,k}_i(x).$$

Thus, establishing Equation (9.33).
□

Using Equation (9.33), we establish that the current Lagrange polynomials span the approximation space $\mathcal{P}^1_k$.

```
Lemma P1k_lin_span_LagP1k_cur : ∀(vtx_cur : 'R^{2,1}),
  affine_independent vtx_cur → Pdk 1 k = lin_span (LagP1k_cur k vtx_cur).
```

This lemma is verified by directly applying the lemma `Pdk_am_compose_basis` (see Section 8.2.3), which confirms that the polynomial space $\mathcal{P}_k^1$ precisely corresponds to the linear span of the reference Lagrange polynomials $(\hat{\mathcal{L}}_i^{1,k})_{i\in[0..k]}$ when composed with the mapping $(T_{\text{geo}}^K)^{-1}$. This composition is valid under the conditions that $(\hat{\mathcal{L}}_i^{1,k})_{i\in[0..k]}$ already form a basis for $\mathcal{P}_k^1$ as confirmed by the lemma `LagP1k_ref_basis`, and that the transformation $(T_{\text{geo}}^K)^{-1}$ is both affine and bijective. These conditions are substantiated by the lemmas `T_geom_inv_am` and `f_inv_bij`, respectively (refer to Sections 9.3 and 5.1.3).

Consequently, from the linear independence established by the lemma `LagP1k_cur_lin_indep` and the fact that they span $\mathcal{P}_k^1$ as demonstrated by the lemma `P1k_lin_span_LagP1k_cur`, we can conclude that the current Lagrange polynomials $(\mathcal{L}_i^{1,k})_{i\in[0..k]}$ indeed form a basis for the polynomial space $\mathcal{P}_k^1$.

```
Lemma LagP1k_cur_basis : ∀(vtx_cur : 'R^{2,1}),
  affine_independent vtx_cur → basis (Pdk 1 k) (LagP1k_cur k vtx_cur).
```

We conclude that, in contrast to what was established previously, where we derived the properties of the reference Lagrange polynomials $(\hat{\mathcal{L}}_i^{1,k})_{i\in[0..k]}$ from the current ones, we are now reversing the process. We have deduced that the current Lagrange polynomials $(\mathcal{L}_i^{1,k})_{i\in[0..k]}$ form a basis, based on the reference polynomials $(\hat{\mathcal{L}}_i^{1,k})_{i\in[0..k]}$ constituting a basis.

Chapter 10

Simplicial Lagrange Finite Elements

The Lagrange family of finite elements is a fundamental concept in the field of numerical analysis, particularly within the framework of the finite element method used for solving partial differential equations [59]. Each Lagrange finite element is associated with a polynomial space and a family of nodal points, where the solution values are exactly represented by the polynomial approximation (see Section 6 for more details).

This chapter is structured as follows: initially, in Section 10.1 we define the face hyperplanes for both current and reference geometric elements and explore their interaction with Lagrange polynomials. Next, we introduce the geometric mappings, which consist of transitioning between the faces of the geometric elements, as elaborated in Section 10.2. We then proceed to discuss the construction of current simplicial Lagrange finite elements, detailed in Section 10.3. Finally, we outline the generation of the reference simplicial Lagrange finite element with fixed dimensions d and degree k, covered in Section 10.4.

10.1 Face Hyperplanes

In geometry, a *hyperplane* is an affine space whose dimension is one less than that of the space in which it resides. For example, in a three-dimensional space, a hyperplane is a plane. Additionally, a *hyperface* refers to a part of the face hyperplane that is a boundary or edge of a geometric structure. For example, in three-dimensional spaces, the hyperfaces of a tetrahedron are flat triangles, and in two-dimensional space, it is a straight line. We also call *face hyperplane* the hyperplane containing one of the hyperfaces of the simplicial geometric element. The concept of face hyperplanes will be needed later, particularly for proving the unisolvence of simplicial Lagrange finite elements (see Section 10.3.7), and when transforming nodes, for instance, from a reference element of $d-1$ to an hyperface of a current geometric element on d dimension using affine mappings, which will be discussed throughout this chapter. Some figures will accompany these explanations to aid in understanding the concepts.

In this chapter, we will introduce two face hyperplanes $\mathcal{H}_0^d$ of simplices, the face hyperplane opposite to the first vertex $\mathbf{v}_0$ of the current simplex K in $\mathbb{R}^d$, and the face hyperplane $\hat{\mathcal{H}}_d^d$ opposite to the last vertex $\hat{\mathbf{v}}_d$ of the reference simplex $\hat{K}$.

Consider a spatial dimension $d \geq 1$, and a family of affinely independent vertices $(\mathbf{v}_i)_{i\in[0..d]}$ (refer to Section 5.2.3 for more details on finite families). The face hyperplane $\mathcal{H}_0^d$ of the current simplex of vertices $(\mathbf{v}_i)_{i\in[0..d]}$ is a subspace in $\mathbb{R}^d$, formally defined as the kernel of the first current Lagrange polynomial:

$$\mathcal{H}_0^d \overset{\text{def.}}{=} \ker\left(\mathcal{L}_0^{d,1}\right) = \{\mathbf{x} \in \mathbb{R}^d \mid \mathcal{L}_0^{d,1}(\mathbf{x}) = 0\}. \tag{10.1}$$

Here, $(\mathcal{L}_i^{d,1})_{i\in[0..d]}$ represents the current Lagrange polynomials defined in Section 9.5. The face hyperplane $\mathcal{H}_0^d$ is the set of all points that belong to the hyperplane defined by $(\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_d)$ (i.e., containing the face opposite the first vertex $\mathbf{v}_0$ of the current simplex in $\mathbb{R}^d$) (see e.g. Figure 10.1). The face hyperplane $\mathcal{H}_0^d$ is formalized as follows:

```
Definition face0 d (vtx_cur : 'R^{d.+1,d}) (Hvtx: affine_independent vtx_cur):
  'R^d → Prop := fun x ⇒ LagPd1_cur vtx_cur Hvtx ord0 x = 0.
```

In this context, `'R^{d.+1,d}` is a notation that represents a function that maps each pair of indices from `'I_{d.+1}` to `'I_d` to a real number $\mathbb{R}$. We recall from Section 2.3 that `'I_d` denotes the type of all natural numbers less than `n`.

Similarly, the last reference hyperplane $\hat{\mathcal{H}}_d^d$ contains the face of the reference simplex $\hat{K}$ in $\mathbb{R}^d$ positioned opposite to the last vertex $\hat{\mathbf{v}}_d$.

$$\hat{\mathcal{H}}_d^d \overset{\text{def.}}{=} \ker\left(\hat{\mathcal{L}}_d^{d,1}\right) = \{\hat{\mathbf{x}} \in \mathbb{R}^d \mid \hat{\mathcal{L}}_d^{d,1}(\hat{\mathbf{x}}) = 0\}. \tag{10.2}$$

This definition is specified in Coq as follows:

```
Definition face_ref_d d : 'R^d → Prop := fun x ⇒ LagPd1_ref d ord_max x = 0.
```

Here, `LagPd1_ref` represents the reference Lagrange polynomials evaluated at the maximum ordinal `ord_max`, which corresponds to the last vertex of the simplex in $\mathbb{R}^d$ (refer to Section 9.2).

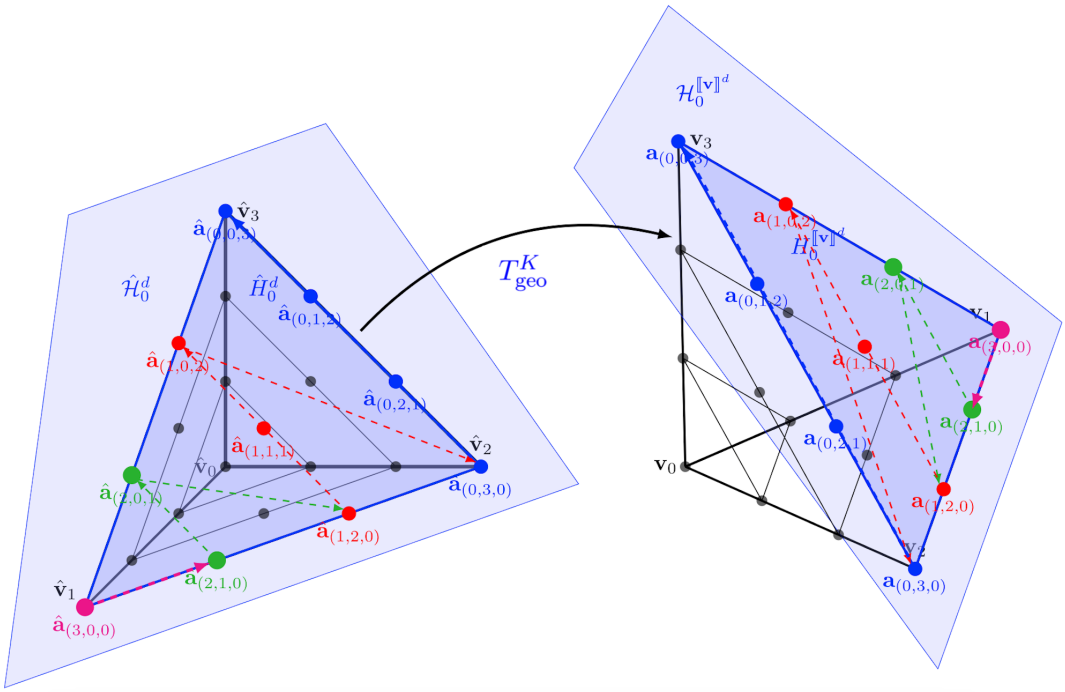

Figure 10.1: Geometric hyperface mapping T^K_{geo} in the case $d = k = 3$ with $d = d_1 + 1$. This figure depicts the transformation T^K_{geo} of a reference simplex $\hat{K}_d$ onto the current simplex K, and reference nodes $\hat{\mathbf{a}}_{\boldsymbol{\alpha}}$ onto current nodes $\mathbf{a}_{\boldsymbol{\alpha}}$, see Lemmas `T_geom_transports_vtx` and `T_geom_transports_node`. For instance, we show that the reference hyperplane $\hat{\mathcal{H}}_d^d$ is mapped onto $\mathcal{H}_0^d$. The nodes in these two faces are colored in order to help see the correspondence: for all $\boldsymbol{\alpha} \in \mathcal{C}_3^3$, we have $\mathbf{a}_{\boldsymbol{\alpha}} = T^K_{\text{geo}}(\hat{\mathbf{a}}_{\boldsymbol{\alpha}})$.

A point $\mathbf{x} \in \mathbb{R}^d$ belongs to the hyperplane $\mathcal{H}_0^d$ if and only if $\mathbf{x}$ can be represented as an affine combination of the vertices $(\mathbf{v}_i)_{i\in[1..d]}$ where the sum of the coefficients $(L_i)_{i\in[0..d-1]}$ equals 1.

```
Lemma face0_equiv : ∀d (vtx_cur : 'R^{d.+1,d}) (x : 'R^d)
  (Hvtx: affine_independent vtx_cur),
  face0 d vtx_cur Hvtx x ↔ (∃ L : 'R^d, sum L = 1 ∧ x = lin_comb L (liftF_S vtx_cur)).
```

`liftF_S` skips the first item of the argument family (refer to Section 5.2.3). Thus, (`liftF_S vtx_cur`) contains the vertices $\mathbf{v}_1$ to $\mathbf{v}_d$. The proof demonstrates both implications of the equivalence.

($\Rightarrow$) Let $\mathbf{x} \in \mathbb{R}^d$ lie on the hyperplane $\mathcal{H}_0^d$. The proof begins by selecting the coefficients L as the current Lagrange polynomials, $\mathcal{L}_{i+1}^{d,1}(\mathbf{x}) \in \mathbb{R}^d$, for $\mathbf{x} \in \mathbb{R}^d$. We must demonstrate:

$$\sum_{i=0}^{d-1} \mathcal{L}_{i+1}^{d,1}(\mathbf{x}) = 1 \quad \wedge \quad \mathbf{x} = \sum_{i=0}^{d-1} \mathcal{L}_{i+1}^{d,1}(\mathbf{x})\mathbf{v}_{i+1}. \tag{10.3}$$

Indeed,

(i) We verify the first part of this claim (10.3). Given the assumption, we know that for all $\mathbf{x} \in \mathcal{H}_0^d, \mathcal{L}_0^{d,1}(\mathbf{x}) = 0$. Since the sum of the Lagrange polynomials $(\mathcal{L}_i^{d,1})_{i\in[0..d]}$ equals 1 as established by the lemma `LagPd1_cur_sum_1` (see Section 9.5), we have

$$\sum_{i=0}^{d-1} \mathcal{L}_{i+1}^{d,1}(\mathbf{x}) = \sum_{i=0}^{d} \mathcal{L}_i^{d,1}(\mathbf{x}) - \mathcal{L}_0^{d,1}(\mathbf{x}) = 1 - 0 = 1.$$

(ii) As for the second part of the claim (10.3), we recognize that any point $\mathbf{x} \in \mathcal{H}_0^d$ can be uniquely decomposed into an affine combination of the vertices $(\mathbf{v}_i)_{i\in[0..d]}$, using coefficients derived from the Lagrange polynomials $(\mathcal{L}_i^{d,1})_{i\in[0..d]}$ applied to $\mathbf{x}$ as expressed by Equation (9.31) in Section 9.5. Given that for all $\mathbf{x} \in \mathcal{H}_0^d$ then $\mathcal{L}_0^{d,1}(\mathbf{x}) = 0$, the decomposition simplifies to:

$$\mathbf{x} = \sum_{i=0}^{d} \mathcal{L}_i^{d,1}(\mathbf{x})\mathbf{v}_i = \mathcal{L}_0^{d,1}(\mathbf{x})\mathbf{v}_0 + \sum_{i=0}^{d-1} \mathcal{L}_{i+1}^{d,1}(\mathbf{x})\mathbf{v}_{i+1} = \sum_{i=0}^{d-1} \mathcal{L}_{i+1}^{d,1}(\mathbf{x})\mathbf{v}_{i+1}.$$

($\Leftarrow$) Assuming that $\mathbf{x} \in \mathbb{R}^d$ is a linear combination of the vertices $(\mathbf{v}_i)_{i\in[1..d]}$, with coefficients $(L_i)_{i\in[0..d-1]}$ that sum to 1. We then define $(L'_j)_{j\in[0..d]} = (0, L_0, L_1, \ldots, L_{d-1}) \in \mathbb{R}^{d+1}$. Considering that the inverse geometric transformation $(T_{\text{geo}}^K)^{-1}$ preserves barycentric coordinates, as detailed in Equation (9.17) in Section 9.3, and maps the vertices $\mathbf{v}_i$ onto $\hat{\mathbf{v}}_i$, we can derive the following result:

$$\begin{aligned}
\mathcal{L}_0^{d,1}(\mathbf{x}) = \hat{\mathcal{L}}_0^{d,1} \circ (T_{\text{geo}}^K)^{-1}(\mathbf{x}) &= \hat{\mathcal{L}}_0^{d,1} \circ (T_{\text{geo}}^K)^{-1} \left(\sum_{i=0}^{d-1} L_i \mathbf{v}_{i+1} \right) \\
&= \hat{\mathcal{L}}_0^{d,1} \left(\sum_{i=0}^{d-1} L_i \hat{\mathbf{v}}_{i+1} \right) \\
&= \hat{\mathcal{L}}_0^{d,1} \left(\sum_{j=0}^{d} (0, L_0, L_1, \ldots, L_{d-1})\ \hat{\mathbf{v}}_j \right) \\
&= \hat{\mathcal{L}}_0^{d,1} \left(\sum_{j=0}^{d} L'_j\ \hat{\mathbf{v}}_j \right) = L'_0 = 0.
\end{aligned}$$

From the lemma `LagPd1_ref_lc`, elaborated in Section 9.2, we obtain the last equality. Thus, $\mathbf{x} \in \mathcal{H}_0^d$ and the verification of this equivalence is complete. □

In a d-dimensional simplex determined by the $d+1$ affinely independent vertices $(\mathbf{v}_i)_{i\in[0..d]}$ in $\mathbb{R}^d$, where $d \geq 1$, we consider the current nodes $(\mathbf{a}_{\boldsymbol{\alpha}})_{\boldsymbol{\alpha}\in\mathcal{A}_k^d}$, as detailed by the Equation 9.3 in Section 9.1.2. The placement of this node within the face hyperplane $\mathcal{H}_0^d$ is indicated by its index. Assuming $k \geq 1$ and a multi-index $\boldsymbol{\alpha} \in \mathcal{A}_k^d$, the relationship can be expressed as:

$$\mathbf{a}_{\boldsymbol{\alpha}} \in \mathcal{H}_0^d \iff \boldsymbol{\alpha} \in \mathcal{C}_k^d. \tag{10.4}$$

In this context, $\mathcal{C}_k^d$ denotes the family of multi-indices whose sum precisely equals the degree k of the polynomial approximation (see Equation (8.2) in Section 8.1.1). This is formalized in Coq as:

```
Lemma node_face0_in_Cdk : ∀d k (vtx_cur : 'R^d^d.+1)
  (ipk : 'I_((pbinom d k).+1)) (Hvtx: affine_independent vtx_cur),
  0 < d → 0 < k → (pbinom d k.-1).+1 ⩽ ipk ↔
    face0 d vtx_cur Hvtx (node_cur d k vtx_cur ipk).
```

In this context, `(pbinom d k.-1).+1 ⩽ ipk` signifies that $\boldsymbol{\alpha} \in \mathcal{A}_k^d \setminus \mathcal{A}_{k-1}^d = \mathcal{C}_k^d$ (refer to Section 8.1). The proof of this equivalence is intricate, employing various algebraic properties of Lagrange polynomials. By applying the lemma `T_geom_transports_node` discussed in Section 9.3, and using the fact that the inverse geometric mapping $(T_{\text{geo}}^K)^{-1}$ transports the barycentric coordinates as detailed in Equation (9.17) from Section 9.3, we simplify the proof process for the equivalence of the Equation (10.4) as:

$$\begin{aligned}
\mathbf{a}_{\boldsymbol{\alpha}} \in \mathcal{H}_0^d \iff 0 = \mathcal{L}_0^{d,1}(\mathbf{a}_{\boldsymbol{\alpha}}) &= \hat{\mathcal{L}}_0^{d,1} \circ (T_{\text{geo}}^K)^{-1}(\mathbf{a}_{\boldsymbol{\alpha}}) \\
&= \hat{\mathcal{L}}_0^{d,1}\left((T_{\text{geo}}^K)^{-1}(T_{\text{geo}}^K(\hat{\mathbf{a}}_{\boldsymbol{\alpha}}))\right) \\
&= \hat{\mathcal{L}}_0^{d,1}\left((T_{\text{geo}}^K)^{-1}\left(\sum_{i=0}^{d} \hat{\mathcal{L}}_i^{d,1}(\hat{\mathbf{a}}_{\boldsymbol{\alpha}})\mathbf{v}_i\right)\right) \\
&= \hat{\mathcal{L}}_0^{d,1}\left(\sum_{i=0}^{d} \hat{\mathcal{L}}_i^{d,1}(\hat{\mathbf{a}}_{\boldsymbol{\alpha}})\hat{\mathbf{v}}_i\right).
\end{aligned}$$

This leads us to demonstrate that:

$$\hat{\mathcal{L}}_0^{d,1}\left(\sum_{i=0}^{d} \hat{\mathcal{L}}_i^{d,1}(\hat{\mathbf{a}}_{\boldsymbol{\alpha}})\hat{\mathbf{v}}_i\right) = 0 \iff \sum_{i=0}^{d-1} \alpha_i = k. \tag{10.5}$$

The proof of this equivalence is established as follows:

Using the property of linear combinations `lc_ind_l`, which extracts the first item of the linear combination (refer to Section 5.3.4), and drawing upon the definitions of the reference vertices where $\hat{\mathbf{v}}_0 = \mathbf{0}$ and $\hat{\mathbf{v}}_i = \boldsymbol{\delta}_{i-1}$ for $i \in [1, d]$ as discussed in Section 9.1.1, and considering the definition of the reference Lagrange polynomials outlined in Equation (9.11) in Section 9.2, we express the left-hand side of the left equation of the equivalence (10.5) as follows:

$$\begin{aligned}
\hat{\mathcal{L}}_0^{d,1}\left(\sum_{i=0}^{d}\hat{\mathcal{L}}_i^{d,1}(\hat{\mathbf{a}}_{\boldsymbol{\alpha}})\hat{\mathbf{v}}_i\right) &= \hat{\mathcal{L}}_0^{d,1}\left(\hat{\mathcal{L}}_0^{d,1}(\hat{\mathbf{a}}_{\boldsymbol{\alpha}})\hat{\mathbf{v}}_0 + \sum_{i=0}^{d-1}\hat{\mathcal{L}}_{i+1}^{d,1}(\hat{\mathbf{a}}_{\boldsymbol{\alpha}})\hat{\mathbf{v}}_{i+1}\right) \\
&= \hat{\mathcal{L}}_0^{d,1}\left(\left(1-\sum_{i=0}^{d-1}(\hat{\mathbf{a}}_{\boldsymbol{\alpha}})_i\right)\hat{\mathbf{v}}_0 + \sum_{i=0}^{d-1}\hat{\mathcal{L}}_{i+1}^{d,1}(\hat{\mathbf{a}}_{\boldsymbol{\alpha}})\hat{\mathbf{v}}_{i+1}\right) \\
&= 1-\sum_{j=0}^{d-1}\left(0+\sum_{i=0}^{d-1}\hat{\mathcal{L}}_{i+1}^{d,1}(\hat{\mathbf{a}}_{\boldsymbol{\alpha}})\delta_{i,j}\right), \quad \text{since } \hat{\mathbf{v}}_0 = \mathbf{0} \\
&= 1-\sum_{j=0}^{d-1}\hat{\mathcal{L}}_{j+1}^{d,1}(\hat{\mathbf{a}}_{\boldsymbol{\alpha}}) = 1-\sum_{j=0}^{d-1}(\hat{\mathbf{a}}_{\boldsymbol{\alpha}})_j = 1-\sum_{j=0}^{d-1}\frac{\alpha_j}{k} \\
&= \frac{1}{k}\left(k-\sum_{j=0}^{d-1}\alpha_j\right).
\end{aligned}$$

Thus, its cancellation when $k > 0$ is equivalent to $\sum_{j=0}^{d-1}\alpha_j = k$. This yields the desired result. □

From the properties (8.2) and (10.4), we conclude that the current sub-nodes $\check{\mathbf{a}}_{\boldsymbol{\alpha}}$ corresponding to all $\boldsymbol{\alpha} \in \mathcal{A}_{k-1}^d$ for $k \geq 1$, do not lie on the face hyperplane $\mathcal{H}_0^d$. This conclusion is supported by the equality of the sub-nodes with the current nodes, as detailed in Equation (9.10) in Section 9.1.4.

```
Lemma sub_node_out_face0 : ∀d k (vtx_cur : 'R^d.+1) (ipk : 'I_((pbinom d k).+1))
  (Hvtx: affine_independent vtx_cur),
  0 < d → 0 < k → ¬ face0 d vtx_cur Hvtx (sub_node d k.+1 vtx_cur ipk).
```

Here, the symbol ¬ represents logical negation used to mean "not".

This lemma leads to the following implication:

$$\boldsymbol{\alpha} \in \mathcal{A}_{k-1}^d \Longrightarrow \check{\mathbf{a}}_{\boldsymbol{\alpha}} \notin \mathcal{H}_0^d. \tag{10.6}$$

10.2 Geometric Mappings

This section explores geometric hyperface mapping with an index shift function, as discussed in Section 10.2.1, and geometric mapping using a permutation function, explained in Section 10.2.2. These mappings are needed in the subsequent Section 10.3.7, particularly for establishing the unisolvence of simplicial Lagrange finite elements, which is fundamental to the reliability of the finite element method.

10.2.1 Geometric Hyperface Mapping

We introduce the concept of geometric hyperface mapping with an index shift function. This section explains the process of transforming the reference geometric element $\hat{K}$ in dimension d_1 to current geometric elements K in dimension $d_1 + 1$.

For an index $i \in [0..d_1 + 1]$, we introduce the mapping $\theta_i^{d_1}$ from $[0..d_1]$ to $[0..d_1 + 1]$, defined by:

$$\forall j \in [0..d_1], \quad \theta_i^{d_1}(j) \stackrel{\text{def.}}{=} \begin{cases} j & \text{if } j < i, \\ j+1 & \text{if } j \geq i. \end{cases}$$

The mapping $\theta_i^{d_1}$ corresponds precisely to the `lift i` function, and is the `lift_S` function when `i = ord0` (refer to Sections 2.3 and 5.2.2, respectively). This function shifts indices by inserting a gap immediately after index i for those j values that are greater than or equal to i. The function is injective, and is illustrated in Figure 10.2.

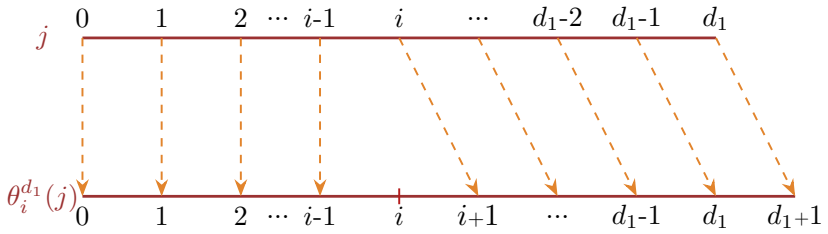

Figure 10.2: Illustration of the function $\theta_i^{d_1}$: It maps an index $j \in [0..d_1]$ from the top horizontal axis to a new index on the bottom horizontal axis, which ranges in $[0..d_1+1]$. For indices where $j < i$, the function preserves the value, i.e., $\theta_i^{d_1}(j) = j$. For indices where $j \geq i$, the function increments j by 1, meaning $\theta_i^{d_1}(j) = j+1$. This behavior is represented in the figure using dashed arrows.

When $i = 0$, the mapping $\theta_0^{d_1}$ shifts all indices by 1 in the set $[0..d_1]$ as $\theta_0^{d_1} = j+1$, for all $j \in [0..d_1]$.

Consider a family of $d_1 + 2$ affinely independent vertices, denoted as $(\mathbf{v}_i)_{i\in[0..d_1+1]}$, in the space $\mathbb{R}^{d_1+1}$. The hyperface mapping $\phi_{\theta_0^{d_1}}$ is defined to transform coordinates from the reference geometric element $\hat{K}$ in a lower-dimensional space $\mathbb{R}^{d_1}$, to a current geometric element K in a higher-dimensional space $\mathbb{R}^{d_1+1}$ applying the *index shift* function defined by $\theta_0^{d_1}$ as illustrated in the Figure 10.1. We represent this function in Coq as `T_geom_face0` and it is mathematically expressed as:

$$\forall \hat{x} \in \mathbb{R}^{d_1}, \quad \phi_{\theta_0^{d_1}}(\hat{\mathbf{x}}) = \mathbf{v}_1 + \sum_{i=0}^{d_1-1} \hat{x}_i \ (\mathbf{v}_{i+2} - \mathbf{v}_1). \tag{10.7}$$

This is translated in Coq as:

```
Definition T_geom_face0 : 'R^d1 → 'R^d1.+1 := fun (x_ref : 'R^d1) ⇒
  vtx_cur ord1 + lin_comb x_ref (liftF_S (liftF_S vtx_cur) − constF d1 (vtx_cur ord1)).
```

The Equation (10.7) implies that $\phi_{\theta_0^{d_1}}$ is an affine map.

```
Lemma T_geom_face0_am : aff_map T_geom_face0
```

The mapping $\phi_{\theta_0^{d_1}}$ consists of a constant vector $\mathbf{v}_1$ and a linear combination of the terms $\hat{x}_i \ (\mathbf{v}_{i+2} - \mathbf{v}_1)$, which constitutes a linear map. This result follows directly from the lemma `am_lm_ms` (refer to Section 5.3.6), which establishes that adding a constant vector to a linear map yields an affine map. The fact that $\phi_{\theta_0^{d_1}}$ is an affine map is essential, as it will be needed further to prove its bijectivity, which in turn is essential to establishing the unisolvence of the Lagrange finite elements.

A convenient alternate definition for the function $\phi_{\theta_0^{d_1}}$ is:

$$\phi_{\theta_0^{d_1}}(\hat{\mathbf{x}}) = \sum_{i=0}^{d_1} \hat{\mathcal{L}}_i^{d_1,1}(\hat{\mathbf{x}}) \ \mathbf{v}_{i+1}. \tag{10.8}$$

This is formulated in Coq as follows:

```
Definition T_geom_face0_alt :'R^d1 → 'R^d1.+1 := fun x_ref : 'R^d1 ⇒
  lin_comb (fun i : 'I_d1.+1 ⇒ LagPd1_ref d1 i x_ref) (liftF_S vtx_cur).
```

Instead of directly manipulating the vertices, this version (10.8) uses the reference Lagrange polynomials $(\hat{\mathcal{L}}_i^{d_1,1})_{i\in[0..d_1]}$.

To demonstrate the equivalence between these two approaches, yielding identical results, we present the following lemma:

```
Lemma T_geom_face0_eq : T_geom_face0_alt = T_geom_face0.
```

The proof of this lemma relies on the application of the lemma `lc_ind_1`, as detailed in Section 5.3.4. We have

$$\begin{aligned}\sum_{i=0}^{d_1} \hat{\mathcal{L}}_i^{d_1,1}(\hat{\mathbf{x}})\mathbf{v}_{i+1} &= \hat{\mathcal{L}}_0^{d_1,1}(\hat{\mathbf{x}})\mathbf{v}_1 + \sum_{i=0}^{d_1-1} \hat{\mathcal{L}}_{i+1}^{d_1,1}(\hat{\mathbf{x}})\mathbf{v}_{i+2} \\ &= \left(1 - \sum_{i=0}^{d_1-1} \hat{x}_i\right)\mathbf{v}_1 + \sum_{i=0}^{d_1-1} \hat{x}_i\mathbf{v}_{i+2} \\ &= \mathbf{v}_1 + \sum_{i=0}^{d_1-1} \hat{x}_i\ (\mathbf{v}_{i+2} - \mathbf{v}_1) = \phi_{\theta_0^{d_1}}(\hat{\mathbf{x}}).\end{aligned}$$

□

We present two equivalent forms of the expressions for the $\phi_{\theta_0^{d_1}}$ mapping because each form is useful depending on the context. When the goal is to manipulate equations directly with points $\mathbf{v}_i$, we utilize Equation (10.7). Alternatively, when working with equations involving Lagrange polynomials, Equation (10.8) is more appropriate. Specifically, Equation (10.8) will be instrumental later on to demonstrate that the image of $\mathbb{R}^{d_1}$ under the $\phi_{\theta_0^{d_1}}$ mapping is included in the face hyperplane $\mathcal{H}_0^d$.

Composing any polynomial p from a higher-dimensional polynomial approximation space $\mathcal{P}_k^{d_1+1}$ with the geometric hyperface map $\phi_{\theta_0^{d_1}} : \mathbb{R}^{d_1} \to \mathbb{R}^{d_1+1}$ effectively transitions it to a lower-dimensional space $\mathcal{P}_k^{d_1}$, while preserving its polynomial degree k.

$$\forall d_1 \geq 0, \forall p \in \mathcal{P}_k^{d_1+1}, \quad p \circ \phi_{\theta_0^{d_1}} \in \mathcal{P}_k^{d_1}.$$

This is encapsulated in the following lemma:

```
Lemma T_geom_face0_compose : ∀k (p : FRd d1.+1),
  (Pdk d1.+1 k) p → Pdk d1 k (compose p T_geom_face0).
```

The validity of this lemma follows straightforwardly from the lemma `Pdk_compose_am` demonstrated in Section 8.2.3, using the fact that $\phi_{\theta_0^{d_1}}$ is an affine map by the lemma `T_geom_face0_am`.

An important aspect of the mapping $\phi_{\theta_0^{d_1}}$, akin to the property discussed in the lemma `T_geom_transports_node` for the geometric mapping T_{geo}^K as detailed in Section 9.3, is that it transforms a reference node into a corresponding current node on the face of the current geometric element K. This is visually represented in Figure 10.3. This transformation is verified through the following lemma:

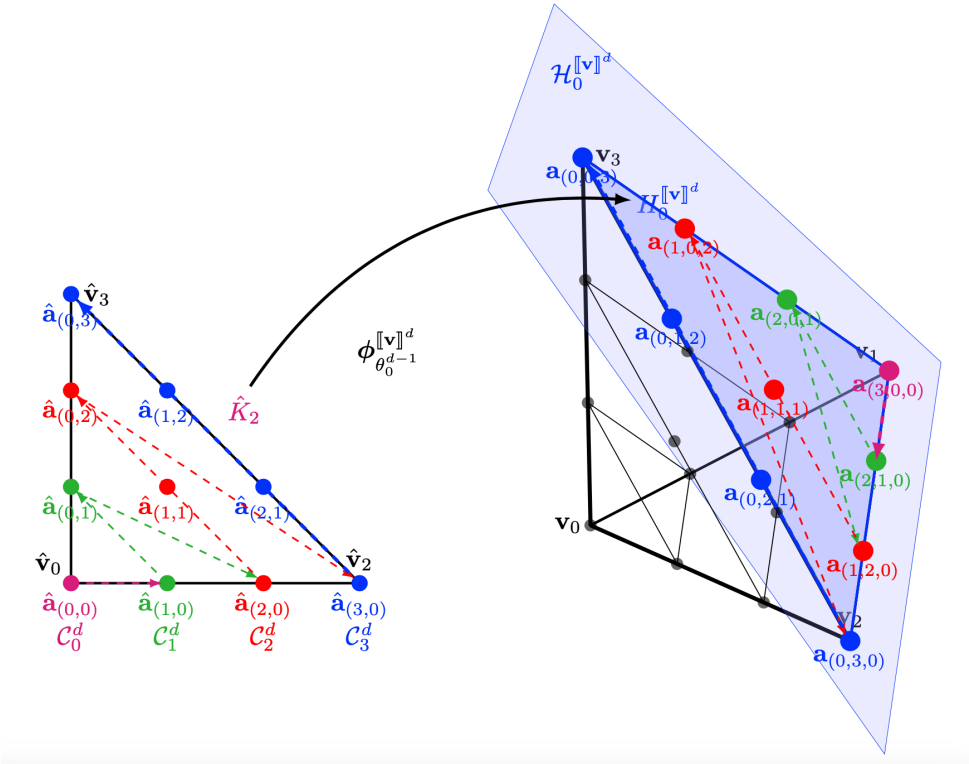

Figure 10.3: Geometric hyperface mapping $\phi_{\theta_0^{d_1}}$ in the case $d_1 + 1 = k = 3$. Illustration of the transformation $\phi_{\theta_0^{d_1}}$ of a reference triangle $\hat{K}_2$ onto $\mathcal{H}_0^{d_1+1}$, which is highlighted as the 0-th face opposite the vertex $\mathbf{v}_0$ depicted in blue. The mapping $\phi_{\theta_0^{d_1}}$ showcases the correspondence, illustrated by the colors, between the reference nodes in the reference triangle $\hat{K}_2$ and the face nodes of the tetrahedron: From Equation (10.9), we have $\phi_{\theta_0^{d_1}}(\hat{\mathbf{a}}_{(i,j)}) = \mathbf{a}_{(3-(i+j),i,j)}$, for all $(i,j) \in \mathcal{A}_2^3$. The hyperplane is labeled $\mathcal{H}_0^{[[v]]^d}$ in the figure, indicating that the definition of this hyperplane depends on the vertex configuration denoted by $\mathbf{v}$ in dimension $d = d_1+1$. However, for simplicity in the text, this notation is abbreviated to $\mathcal{H}_0^{d+1}$.

```
Lemma T_geom_face0_map_node : ∀k (ipk : 'I_(pbinom d1 k).+1), 0 < k →
    T_geom_face0 (node_ref d1 k ipk) =
      node_cur d1.+1 k vtx_cur (Adk_inv d1.+1 k (T_node_face0 d1 k ipk)).
```

The definitions of `node_ref` and `node_cur` are detailed in Sections 9.1.1 and 9.1.2, respectively. `T_node_face0` is the function $f_{k,0}^{d_1}$ defined by the Equation (8.6) in Section 8.1, and `Adk_inv` is the inverse function of `Adk`, computing indices corresponding to the multi-indices in the family $\mathcal{A}_k^{d_1}$, as defined in Section 8.1. This relationship is mathematically captured for all $k > 0$ as follows:

$$\forall \boldsymbol{\alpha} \in \mathcal{A}_k^{d_1}, \quad \phi_{\theta_0^{d_1}}(\hat{\mathbf{a}}_{\boldsymbol{\alpha}}^{d_1}) = \mathbf{a}_{f_{k,0}^{d_1}(\boldsymbol{\alpha})}^{d_1+1}. \tag{10.9}$$

In this context, $\hat{\mathbf{a}}_{\boldsymbol{\alpha}}^{d_1}$ denotes the reference Lagrange nodes in the geometric element $\hat{K}$ within $\mathbb{R}^{d_1}$, and $\mathbf{a}_{f_{k,0}^{d_1}(\boldsymbol{\alpha})}^{d_1+1}$ represents the current Lagrange nodes in K within the higher-dimensional space $\mathbb{R}^{d_1+1}$. The mapping function $f_{k,0}^{d_1} \overset{\text{def.}}{=} (\boldsymbol{\alpha} \longmapsto (k - |\boldsymbol{\alpha}|, \boldsymbol{\alpha}))$ takes its values in $\mathcal{C}_k^{d_1+1}$, a subset of $\mathcal{A}_k^{d_1+1}$. This function is bijective from $\mathcal{A}_k^{d_1}$ to $\mathcal{C}_k^{d_1+1}$, ensuring that $|f_{k,0}^{d_1}(\boldsymbol{\alpha})| = k$ as substantiated by the lemma `T_node_face0_sum_eq` discussed in Section 8.1.1.

Let us detail the validity of the Equation (10.9) in more detail. We have from (9.3) and (10.7):

$$\begin{aligned}
\forall \boldsymbol{\alpha} \in \mathcal{A}_k^{d_1}, \quad \mathbf{a}^{d_1+1}_{f^{d_1}_{k,0}(\boldsymbol{\alpha})} &= \left(1 - \frac{|f^{d_1}_{k,0}(\boldsymbol{\alpha})|}{k}\right) \mathbf{v}_0 + \sum_{i=0}^{d_1} \frac{(f^{d_1}_{k,0}(\boldsymbol{\alpha}))_i}{k} \mathbf{v}_{i+1} \\
&= \left(1 - \frac{k - |\boldsymbol{\alpha}| + |\boldsymbol{\alpha}|}{k}\right) \mathbf{v}_0 + \frac{1}{k}\left((k - |\boldsymbol{\alpha}|)\mathbf{v}_1 + \sum_{i=0}^{d_1-1} \alpha_i \mathbf{v}_{i+2}\right) \\
&= 0 + \frac{1}{k}\left((k - |\boldsymbol{\alpha}|)\mathbf{v}_1 + \sum_{i=0}^{d_1-1} \alpha_i \mathbf{v}_{i+2}\right) \\
&= \mathbf{v}_1 + \sum_{i=0}^{d_1-1} \frac{\alpha_i}{k} (\mathbf{v}_{i+2} - \mathbf{v}_1) \\
&= \phi_{\theta_0^{d_1}}(\hat{\mathbf{a}}^{d_1}_{\boldsymbol{\alpha}}).
\end{aligned}$$

□

The image of the full set $\mathbb{R}^{d_1}$ under the geometric hyperface map $\phi_{\theta_0^{d_1}}$ remains within the face hyperplane $\mathcal{H}_0^{d_1+1}$ as:

$$\forall d_1 \geq 0, \quad \phi_{\theta_0^{d_1}}(\mathbb{R}^{d_1}) \subset \mathcal{H}_0^{d_1+1}. \tag{10.10}$$

This relationship is formalized in the following lemma:

```
Lemma T_geom_face0_in_face0 : ∀(vtx_cur : 'R^{d1.+2,d1.+1})
  (Hvtx : affine_independent vtx_cur)
  incl (image T_geom_face0 fullset) (face0 d1.+1 vtx_cur Hvtx)
```

To demonstrate this property, we consider each point $\hat{\mathbf{x}} \in \mathbb{R}^{d_1}$ and show that $\phi_{\theta_0^{d_1}}(\hat{\mathbf{x}})$ belongs to $\mathcal{H}_0^{d_1+1}$. This proof uses the lemma `face0_equiv` from Section 10.1, which offers an equivalent characterization of the hyperplane $\mathcal{H}_0^{d_1+1}$. This involves proving that:

$$\exists \boldsymbol{L} \in \mathbb{R}^{d_1+1}, \quad \sum_{i=0}^{d_1} L_i = 1 \ \wedge \ \phi_{\theta_0^{d_1}}(\hat{\mathbf{x}}) = \sum_{i=0}^{d_1} L_i(\mathbf{x}) \mathbf{v}_{i+1}.$$

To prove this statement, we begin by choosing the reference Lagrange polynomials $(\hat{\mathcal{L}}_i^{d_1,1})_{i\in[0..d_1]} \in \mathbb{R}^{d_1+1}$ as the appropriate candidate for the coefficients $(L_i)_{i\in[0..d_1]}$ of the linear combination of $\phi_{\theta_0^{d_1}}$. The proof consists of verifying two claims: first, we affirm that the sum of the Lagrange polynomials equals 1, which is supported by the lemma `LagPd1_ref_sum_1` discussed in Section 9.2. Second, we verify that $\phi_{\theta_0^{d_1}}(\hat{\mathbf{x}}) = \sum_{i=0}^{d_1} \hat{\mathcal{L}}_i^{d_1,1}(\hat{\mathbf{x}}) \mathbf{v}_{i+1}$, which is provided by the lemma `T_geom_face0_eq`. By establishing these two points, we demonstrate that the image of $\mathbb{R}^{d_1}$ under $\phi_{\theta_0^{d_1}}$ is contained within $\mathcal{H}_0^{d_1+1}$. □

Moreover, since the vertices $(\mathbf{v}_i)_{i\in[0..d_1+1]}$ are affinely independent, then $\phi_{\theta_0^{d_1}}$ is bijective from $\mathbb{R}^{d_1}$ to $\mathcal{H}_0^{d_1+1}$, and its inverse function $\phi^{-1}_{\theta_0^{d_1}}$ is bijective from $\mathcal{H}_0^{d_1+1}$ to $\mathbb{R}^{d_1}$.

```
Lemma T_geom_face0_bijS : ∀(vtx_cur : 'R^{d1.+2,d1.+1})
  (Hvtx : affine_independent vtx_cur)
   bijS fullset (face0 d1.+1 vtx_cur Hvtx) T_geom_face0.

Definition T_geom_face0_inv : 'R^d1.+1 → 'R^d1 := f_invS (T_geom_face0_bijS).
```

Where `f_invS` is the inverse function defined in Section 5.1.4 for the proof of injectivity.

10.2.2 Geometric Mapping with Permutation

The focus here shifts to geometric mappings applying a permutation function. We provide some necessary properties that will be further exploited to prove the lemma of the unisolvence of the Lagrange finite elements in Section 10.3. The practical application of these concepts is illustrated through detailed examples and figures.

We consider the spatial dimension $d \geq 1$. For an index $i \in [0..d]$, we introduce a permutation mapping π_i^d from $[0..d]$ to $[0..d]$, defined by:

$$\forall j \in [0..d], \quad \pi_i^d(j) \overset{\text{def.}}{=} \begin{cases} d & \text{if } j = i, \\ i & \text{if } j = d, \\ j & \text{else .} \end{cases} \tag{10.11}$$

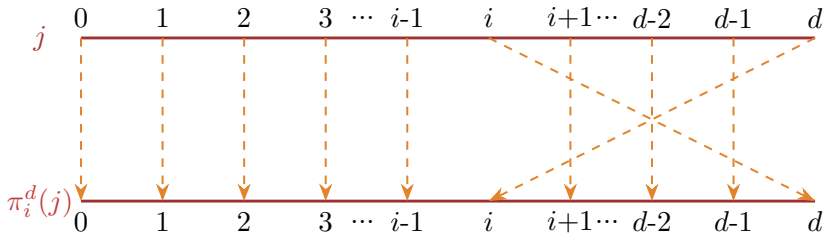

Figure 10.4: Illustration of the permutation function π_i^d, which represents transpositions within the set of indices $[0..d]$ for a spatial dimension $d \geq 1$. The upper horizontal axis represents the input index $j \in [0..d]$, while the lower horizontal axis represents the output index $\pi_i^d(j)$. The permutation mapping π_i^d specifically swaps the index i with d, and maintains the positions of all other indices. This type of permutation is called a *transposition* because it swaps exactly two elements while leaving all others fixed.

The mapping π_i^d is a *transposition*, where exactly two elements swap positions (here i and d), and all other elements remain unchanged. Like all permutations, π_i^d is a bijective function from the set $[0..d]$ onto itself and is involutive, and is illustrated in the Figure 10.4. The bijectivity of the transposition π_i^d ensures that it has an inverse function $(\pi_i^d)^{-1} = \pi_i^d$.

When $i = 0$, the mapping π_0^d can be seen as a simple form of inverting the order of the endpoints (first and last elements) of the set $[0..d]$ while leaving the rest of the indices intact. This consists in swapping the first and last elements.

Consider a family of $d+1$ affinely independent vertices, denoted as $(\mathbf{v}_i)_{i\in[0..d]}$, in $\mathbb{R}^d$. We define a geometric transformation $\phi^K_{\pi_0^d}$ that transforms coordinates from the reference geometric element $\hat{K}$ in $\mathbb{R}^d$ to current geometric elements K in $\mathbb{R}^d$. This mapping uses the transposition function π_0^d, as depicted in Figure 10.4.

The function $\phi^K_{\pi_0^d}$ is formalized in Coq as follows, which corresponds to the geometric mapping T^K_{geo} outlined in Section 9.3, though with a permutation of the vertices.

```
Definition T_geom_d_0 : 'Rˆd → 'Rˆd := T_geom (transpF vtx_cur ord_max ord0).
```

In this context, `transpF` denotes the transposition function, which specifically swaps here the vertices between the last position `ord_max` and the first position `ord0`, as detailed in Section 2.3.

Similarly to (9.12), the mathematical expression for the geometric mapping $\phi^K_{\pi_0^d}$ is given by:

$$\forall \hat{\mathbf{x}} \in \mathbb{R}^d, \quad \phi^K_{\pi^d_0}(\hat{\mathbf{x}}) \overset{\text{def.}}{=} \sum_{i=0}^{d} \hat{\mathcal{L}}^{d,1}_{(\pi^d_0)^{-1}(i)}(\hat{\mathbf{x}})\ \mathbf{v}_i. \tag{10.12}$$

The mapping $\phi^K_{\pi^d_0}$ inherits the properties of the geometric mapping T^K_{geo}. It is an affine map.

```
Lemma T_geom_d_0_am : aff_map T_geom_d_0.
```

The proof that $\phi^K_{\pi^d_0}$ is affine follows directly from the lemma `T_geom_am`, as previously established in Section 9.3. This highlights the preservation of affine properties despite the permutation of vertices.

Additionally, the mapping $\phi^K_{\pi^d_0} : \mathbb{R}^d \to \mathbb{R}^d$ is verified to be bijective.

```
Lemma T_geom_d_0_bij : bijective T_geom_d_0.
```

Since the permutation of vertices by the function π^d_0 preserves the affine independence of the vertices, the bijectivity of $\phi^K_{\pi^d_0}$ is substantiated by the established bijectivity of T^K_{geo} through the lemma `T_geom_bij` referenced in Section 9.3.

Thus, the geometric mapping $\phi^K_{\pi^d_0}$ has an inverse function $(\phi^K_{\pi^d_0})^{-1}$ that maps $\mathbb{R}^d$ to $\mathbb{R}^d$, which is also an affine mapping (refer to Section 5.1.3 for more details on inverse functions).

```
Definition T_geom_d_0_inv : 'R^d → 'R^d := f_inv (T_geom_d_0_bij).
```

Moreover, two properties demonstrate the correctness of $(\phi^K_{\pi^d_0})^{-1}$, ensuring that the application of $(\phi^K_{\pi^d_0})^{-1}$ followed by $\phi^K_{\pi^d_0}$, or vice versa, returns the original input.

$$\forall \mathbf{x} \in \mathbb{R}^d,\ \phi^K_{\pi^d_0} \circ (\phi^K_{\pi^d_0})^{-1}(\mathbf{x}) = \mathbf{x} \quad (\textit{right identity}) \tag{10.13}$$

$$\forall \hat{\mathbf{x}} \in \mathbb{R}^d,\ (\phi^K_{\pi^d_0})^{-1} \circ \phi^K_{\pi^d_0}(\hat{\mathbf{x}}) = \hat{\mathbf{x}} \quad (\textit{left identity}) \tag{10.14}$$

A further attribute of the geometric mapping $\phi^K_{\pi^d_0}$ is that the image of the reference face hyperplane $\hat{\mathcal{H}}^d_d$ of $\hat{K}$ under the geometric mapping $\phi^K_{\pi^d_0}$ is equal to the face hyperplane $\mathcal{H}^d_0$ of the current geometric element K.

$$\phi^K_{\pi^d_0}(\hat{\mathcal{H}}^d_d) = \mathcal{H}^d_0. \tag{10.15}$$

Where $\mathcal{H}^d_0$ and $\hat{\mathcal{H}}^d_d$ are the face hyperplanes defined in Section 10.1.

Additionally, composing any polynomial p from the polynomial space $\mathcal{P}^d_k$ with the transformation mapping $\phi^K_{\pi^d_0}$, the resulting function remains within the same polynomial space. This property is specified by the lemma:

```
Lemma T_geom_d_0_compose : ∀k (p : FRd d),
  Pdk d k p → Pdk d k (compose p T_geom_d_0).
```

This result is established by applying the lemma `Pdk_compose_am`, which was demonstrated in Section 8.2.3. It asserts that since $\phi^K_{\pi^d_0}$ is an affine map, as confirmed by the lemma `T_geom_d_0_am`, composing p with $\phi^K_{\pi^d_0}$ maintains the function within $\mathcal{P}^d_k$.

Similarly, the property holds true for the inverse transformation $(\phi^K_{\pi^d_0})^{-1}$. The relevant lemma is:

```
Lemma T_geom_d_0_inv_compose : ∀k (p : FRd d),
  Pdk d k p → Pdk d k (compose p T_geom_d_0_inv).
```

The proof follows similarly, relying on the condition that $(\phi^K_{\pi^d_0})^{-1}$ is also an affine map.

Now, building on the definition of current Lagrange polynomials $(\mathcal{L}_i^{d,1})_{i\in[0..d]}$ as detailed in Equation (9.30) from Section 9.5, we can demonstrate that these polynomials are also decomposed by the corresponding reference Lagrange polynomials $(\hat{\mathcal{L}}_i^{d,1})_{i\in[0..d]}$ with the inverse transformation mapping $(\phi^K_{\pi^d_0})^{-1}$. This is mathematically and formally represented as follows:

$$\forall \mathbf{x} \in \mathbb{R}^d, \forall i \in [0..d], \quad \mathcal{L}_i^{d,1}(\mathbf{x}) = \hat{\mathcal{L}}^{d,1}_{(\pi^d_0)^{-1}(i)} \circ (\phi^K_{\pi^d_0})^{-1}(\mathbf{x}). \tag{10.16}$$

This is translated in Coq as:

```
Lemma LagPd1_cur_T_geom_d_0_inv : ∀i x,
  LagPd1_cur vtx_cur Hvtx i x =
    transpF (LagPd1_ref d) ord0 ord_max i (T_geom_d_0_inv x).
```

This lemma is verified given that any point $\mathbf{x}$ in the current d-dimensional simplex has a unique decomposition of affinely independent vertices $(\mathbf{v}_i)_{i\in[0..d]}$, ensuring that $\sum_{i=0}^d \mathcal{L}_i^{d,1}(\mathbf{x}) = 1$ such that $\mathbf{x} = \sum_{i=0}^d \mathcal{L}_i^{d,1}\mathbf{v}_i$ as explained by Equation (9.31) in Section 9.5. Let $\mathbf{x} \in \mathbb{R}^d$ and $\hat{\mathbf{y}} \overset{\text{def.}}{=} (\phi^K_{\pi^d_0})^{-1}(\mathbf{x}) \in \mathbb{R}^d$, i.e., such that $\hat{\mathbf{y}} \overset{\text{def.}}{=} \phi^K_{\pi^d_0}(\hat{\mathbf{y}}) = \mathbf{x}$. Using the Equation (10.12), we obtain

$$\mathbf{x} = \sum_{i=0}^{d} \hat{\mathcal{L}}^{d,1}_{(\pi^d_0)^{-1}(i)}(\hat{\mathbf{y}})\mathbf{v}_i = \sum_{i=0}^{d} \hat{\mathcal{L}}^{d,1}_{(\pi^d_0)^{-1}(i)}((\phi^K_{\pi^d_0})^{-1}(\mathbf{x}))\mathbf{v}_i.$$

Thus, from the uniqueness of barycentric coordinates with the sum of coefficients that equals one, we establish that for all $i \in [0..d]$, $\mathcal{L}_i^{d,1} = \hat{\mathcal{L}}^{d,1}_{(\pi^d_0)^{-1}(i)} \circ (\phi^K_{\pi^d_0})^{-1}$.
□

It is important to emphasize that the definition of the function $\phi^K_{\pi^d_0}$ and all its properties apply to any permutation, not limited solely to transpositions π^d_i.

10.3 Construction of the Current Simplicial Lagrange Finite Elements

The objective of this section is to construct the simplicial Lagrange finite element, which builds upon the nodal finite element framework outlined in Section 10.3.1. From the formalization point of view, the primary motivation for formalizing Lagrange finite elements lies in ensuring that our formalization within the `FE` record, detailed in Section 7.1, accurately represents the mathematical principles of finite elements and proves to be both useful and applicable. Moreover, this formalization guarantees that the type `FE` is non-empty for any given cell K.

Before we delve into the details of this part of the work, it is essential to highlight an important aspect concerning the construction of the current Lagrange finite elements (FEs). The transition from the reference finite element (FE) to the current FEs, as detailed in Section 9.4, might have been a logical approach for constructing simplicial Lagrange finite elements using the affine geometric transformation. However, we will first establish the proof of the unisolvence of the $\mathbb{P}^d_k$ Lagrange finite elements for the current FE. We will then proceed to derive properties for the reference FE, as elaborated in Section 10.3.7 for further details on this subject.

In Section 10.3.1, we introduce the linear forms associated with nodes, which are essential for the construction of simplicial Lagrange finite elements. Next, in section 10.3.2 we present definitions and specific properties of the $\mathbb{P}_k^d$ Lagrange finite elements, where k is the degree of the polynomials and d represents the dimensionality. In Section 10.3.3, we analyze cases where the degree k of the polynomial space $\mathcal{P}_k^d$ is zero, establishing the unisolvence of Lagrange finite elements for this condition. Following this, in Sections 10.3.4 and 10.3.5, we extend our examination to scenarios where either the degree k or the dimension d is equal to one, respectively, analyzing each situation's specific characteristics and implications for unisolvence. We particularly study these two cases separately, as they will be needed when we address the proof of the unisolvence principle of the $\mathbb{P}_k^d$ Lagrange finite element employing the double induction on both d and k in Section 10.3.7.

In the sequel, we consider a family of $d+1$ affinely independent vertices $(\mathbf{v}_i)_{i\in[0..d]}$, in the space $\mathbb{R}^d$ with $d \geq 1$ that define a simplex K.

10.3.1 Nodal Linear Forms

This section introduces the nodal linear forms Σ^{nod}, which serve in the construction of Lagrange linear forms in the upcoming Section 10.3.2. Let K a geometric element in the vector space $\mathbb{R}^d$. Nodes within K are typed as follows:

```
Variable node : 'R^d)^nnode.
```

The number of nodes, denoted in Coq as `nnode`, where each node represents a specific location in the domain where the solution is computed. We assume that the number of nodes `nnode` is a positive integer.

As discussed in Section 7.1, n_{dof} is the number of degrees of freedom within a finite element. Consider a finite dimensional space $\mathcal{P}$ of functions in $\mathcal{F}(\mathbb{R}^d, \mathbb{R})$. Let $n \geq 1$, the dimension of $\mathcal{P}$ equals n, is a necessary ingredient required for establishing the unisolvence property of the finite element. In Coq, n is represented as the number of nodes `nnode`, the dimension of $\mathcal{P}$ is `ndof_nodal`, and we define `ndof_nodal` := `nnode`.

Given n nodes $(\mathbf{a}_i)_{i\in[0..n-1]} \in \mathbb{R}^d$, the associated $n_{\text{dof}} \overset{\text{def.}}{=} n$ nodal linear forms in $\mathcal{L}(\mathcal{P}, \mathbb{R})$ are defined by $\Sigma^{\text{nodal}}((\mathbf{a}_i)_{i\in[0..n-1]}) \overset{\text{def.}}{=} (\sigma_i^{\text{nodal}})_{i\in[0..n_{\text{dof}}-1]}$ with:

$$\forall i \in [0..n_{\text{dof}}-1], \forall p \in \mathcal{P}, \quad \sigma_i^{\text{nodal}} : p \longmapsto p(\mathbf{a}_i) \ \in \mathbb{R}. \tag{10.17}$$

This definition is implemented in Coq as:

```
Definition Sigma_nodal : 'FRd d → R)^ndof_nodal :=
  fun (i : 'I_ndof_nodal) (p : FRd d) ⇒ p (node i).
```

Each σ_i^{nodal} is proved to be a linear map through this lemma:

```
Lemma Sigma_nodal_lm : ∀i : 'I_ndof_nodal, lin_map (Sigma_nodal i).
```

Where `lin_map` means that a function is linear if it is compatible with both addition and scalar multiplication, as detailed in Section 5.3.3. The proof of this lemma is straightforward, using the lemma `lm_pt_eval`, which establishes that evaluating a function defined in a module space at a point preserves linearity.

10.3.2 Specifics of the $\mathbb{P}_k^d$ Lagrange Finite Elements

The construction of the $\mathbb{P}_k^d$ Lagrange finite element triple $(K, \mathcal{P}_k^d, \Sigma^{Lag})$ within a d-dimensional space, where $d \geq 1$ and $k \geq 0$, builds upon a nodal framework detailed in the previous section 10.3.1. This involves constructing the geometric element, degrees of freedom, and polynomial approximation space, exclusively in terms of the nodes. It is important to distinguish between $\mathbb{P}_k^d$ and $\mathcal{P}_k^d$, as each term has a unique meaning. $\mathcal{P}_k^d$ refers to the polynomial approximation space, while $\mathbb{P}_k^d$ represents a triplet in the finite element framework, which includes $\mathcal{P}_k^d$ as one of its components.

(i) For the first component of the triplet of the Lagrange finite element, the geometric element K, a subset of $\mathbb{R}^d$, is given as a non degenerate simplex, which is the convex hull of $d+1$ affinely independent vertices in $\mathbb{R}^d$, adhering to the specifications set out in the generic finite element record described in Section 7.1.

```
Definition shape_LagPdk_is_Simplex := Simplex.
Local Definition nvtx_LagPdk dL : nat := nvtx_of_shape dL shape_LagPdk_is_Simplex.
```

where, `nvtx_of_shape` is the number of vertices depending on the dimension d and the shape of K. It is $d+1$ for simplices. In this context, the `Local Definition` keyword is used similarly to `Definition`, but its scope is restricted. It defines, for instance, a function that is only available within the section or file where it is declared.

Lagrange nodes $(\mathbf{a}_{\boldsymbol{\alpha}})_{\boldsymbol{\alpha} \in \mathcal{A}_k^d}$ in the $\mathbb{P}_k^d$ Lagrange finite element are positioned evenly in K. For further detail about the Lagrange nodes, refer to Section 9.1.2. These nodes are the specific locations within each geometric element K where the solution of the model is approximated. The number of nodes, a positive integer, also depends on the approximation degree k. It is the cardinal of the family $\mathcal{A}_k^d$ (refer to Section 8.1), is defined as follows:

```
Definition nnode_LagPdk d k : nat := (pbinom d k).+1.
```

Here, `pbinom` refers to the binomial coefficient function, as discussed in Section 5.5.

(ii) As for the second component of the triplet of the Lagrange finite element, $\mathcal{P}_k^d$ is the finite-dimensional vector space of polynomials defined on K constructed in Sections 8.2. The dimension of $\mathcal{P}_k^d$ matches the number of the local degrees of freedom n_{dof} of Lagrange.

```
Lemma Pdk_has_dim_ndof_LagPdk : ∀d k, has_dim (Pdk d k) (ndof_LagPdk d k).
```

The monomial basis of the polynomial space $\mathcal{P}_{dk}$ is defined in Section 8.2. Additionally, the Lagrange polynomial bases for the cases when d or k equals 1 are formalized in Sections 9.2 and 9.6, respectively.

(iii) Finally, the last component of the triplet is the family of Lagrange linear forms, which is exactly the family of nodal linear forms $\Sigma^{Lag} = \Sigma^{\text{nodal}}((\mathbf{a}_{\boldsymbol{\alpha}})_{\boldsymbol{\alpha} \in \mathcal{A}_k^d})$ defined in Section 10.3.1. Correspondingly, the number of these Lagrange degrees of freedom, corresponds to the number of the nodal degrees of freedom previously defined in section 10.3.1 as follows:

```
Definition ndof_LagPdk d k : nat := ndof_nodal (nnode_LagPdk d k).
```

The next equality underscores the principle that in nodal Lagrange elements, the number of Lagrange degrees of freedom directly corresponds to the number of Lagrange nodes.

```
Lemma ndof_is_dimPdk : ndof_LagPdk = nnode_LagPdk.
```

Each Lagrange node $\mathbf{a}_{\boldsymbol{\alpha}}$, where $\boldsymbol{\alpha} \in \mathcal{A}_k^d$, corresponds to a specific Lagrange degree of freedom σ_i^{nodal}, which are essentially the independent values that the finite element solution seeks to determine. This is formalized in Coq as:

```
Definition Sigma_LagPdSk_cur d k vtx_cur : '(FRd d → R)^(ndof_LagPdk d k) :=
  Sigma_nodal d (nnode_LagPdk d k) (node_cur d k vtx_cur).
```

10.3.3 Unisolvence of the $\mathbb{P}_0^d$ Lagrange Finite Elements

When constructing the $\mathbb{P}_0^d$ Lagrange finite element, where the polynomial degree k is zero, the mathematical complexity is significantly reduced compared to formulations involving higher-degree polynomials, which requires handling more complex terms. This scenario represents the simplest form of Lagrange elements. Specifically, for $k = 0$, there is a single node, and consequently, the family of multi-indices simplifies to $\mathcal{A}_0^d = \{\mathbf{0}\}$. This case is distinctively treated because the node in this instance does not meet the requirements of Equation (9.3).

Let us delve into the mathematical constructs that support the $\mathbb{P}_0^d$ Lagrange finite element.

Based on the construction of the polynomial space $\mathcal{P}_k^d$ detailed in Section 8.2, we deduce $\mathcal{P}_0^d$ for $k = 0$ and any spatial dimension $d \geq 1$ as follows:

$$\mathcal{P}_0^d = \left\{ p \in \mathbb{R}^d \to \mathbb{R} \;\middle|\; \exists c \in \mathbb{R},\ p(\mathbf{x}) = c,\ \forall \mathbf{x} \in \mathbb{R}^d \right\}. \tag{10.18}$$

This space consists of all functions that are constant in $\mathbb{R}^d$. In this case, there is typically just one nodal point per element. Consider a family of $d+1$ affinely independent vertices $(\mathbf{v}_i)_{i\in[0..d]}$, in the space $\mathbb{R}^d$, this node is chosen as the *isobarycenter* (geometric center) of the vertices defining the simplex (refer to Section 5.3.6). This point is computed as the average of the vertices and is placed geometrically at the center of the simplex. This is represented formally by:

```
Definition node_iso : 'R^d := isobarycenter (vtx_cur : 'R^d.+1).
```

In mathematical terms, the isobarycenter `node_iso` is given by:

$$\mathbf{a_0} \overset{\text{def.}}{=} \frac{1}{d+1} \sum_{i=0}^{d} \mathbf{v}_i.$$

The nodal basis function $\Phi_{\Sigma^0} : \mathcal{F}(\mathbb{R}^d, \mathbb{R}) \to \mathbb{R}$ for the $\mathbb{P}_0^d$ finite element is defined such that for any function f in the function space $\mathcal{F}(\mathbb{R}^d, \mathbb{R})$, Φ_{Σ^0} evaluates f at the isobarycenter $\mathbf{a_0}$. This is expressed by:

```
Definition Sigma_0 : '(FRd d → R)^(ndof_LagPdk d 0) := fun _ p ⇒ p node_iso.
```

We assert that each component of the vector Σ^0 is a linear map,

```
Lemma Sigma_0_lm : ∀i : 'I_(ndof_LagPdk dL 0), lin_map (Sigma_0 i).
```

This lemma confirms that the evaluations at a specific point (in this case, the isobarycenter) preserve linearity. This conclusion is drawn directly from the lemma `Sigma_nodal_lm`, as detailed in the previous Section 10.3.1.

We further assert that the map Φ_{Σ^0} establishes a bijective correspondence between the polynomial space $\mathcal{P}_0^d$ and the vector space $\mathbb{R}$. The unisolvence, as explained in Section 6.4, asserts that the degrees of freedom are adequate to uniquely determine any polynomial within $\mathcal{P}_k^d$. Here, for $k = 0$ the isobarycenter suffices to entirely fix a $p \in \mathcal{P}_0^d$. This is established through the following lemma:

```
Lemma unisolvence_LagPd0 : bijS (Pdk d 0) fullset (gather (Sigma_0)).
```

Here, `fullset` and `gather` are defined in Sections 5.1.1 and 5.4.2, respectively.

This assertion is substantiated by initially applying the lemma `lmS_bijS_val_gather_equiv`, outlined in Section 5.4.2. This lemma establishes that the bijective condition is fulfilled because the number of nodal points, specifically, the single isobarycentric node, matches the dimension of the polynomial space $\mathcal{P}_0^d$, which is one. Additionally, the mapping Φ_{Σ^0} is demonstrated to be injective, thereby establishing bijectivity. The criterion for injectivity is easily verified: for any polynomial p in $\mathcal{P}_0^d$, if $p(\mathbf{a_0})$ evaluates to zero, and given that p is a constant c (as detailed in Equation (10.18)), it necessarily follows that p is the zero function. Therefore, the proof concludes effectively, confirming the unique correspondence between the polynomial values and their evaluations at the nodal point.

10.3.4 Unisolvence of the $\mathbb{P}_1^d$ Lagrange Finite Element

For any dimension d where $d \geq 1$, the polynomial space $\mathcal{P}_1^d$ encompasses affine polynomials, which include terms up to the first power (refer to Section 8.3). The nodal points, $(\mathbf{a}_{\boldsymbol{\alpha}})_{\boldsymbol{\alpha}\in\mathcal{A}_1^d}$, coincide with $d+1$ vertices, $(\mathbf{v}_i)_{i\in[0..d]}$, as established by the lemma `vtx_node_Pd1_cur` in Section 9.1.3. We recall from Section 5.3.2, that $\mathcal{A}_1^d = \{\mathbf{0}, \boldsymbol{\delta}_0, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_{d-1}\}$.

Let $(\mathbf{v}_i)_{i\in[0..d]}$ a $d+1$ affinely independent vertices that define a simplex in the multidimensional space $\mathbb{R}^d$. The unisolvence of the $\mathcal{P}_1^d$ Lagrange finite elements guarantees that each polynomial in $\mathcal{P}_1^d$ is uniquely determined by its values at the Lagrange nodes. This property is formalized as follows:

```
Lemma unisolvence_LagPd1_cur : ∀(d : nat) (vtx_cur : 'Rˆ{d.+1,d}),
  0 < d → affine_independent vtx_cur →
   bijS (Pdk d 1) fullset (gather (Sigma_LagPdSk_cur d 1 vtx_cur)).
```

The proof begins by addressing the case where $d = 0$, which is trivial due to the precondition $d > 0$. Moving to the case $d+1$, similar to the approach detailed in Section 10.3.3 for $\mathcal{P}_0^d$, we employ the lemma `lmS_bijS_val_gather_equiv`. This lemma confirms that the mapping $\Phi_{\Sigma^{Lag}}$ is bijective if it is injective, provided that the dimension of $\mathcal{P}_1^{d+1}$ aligns with that of $\mathbb{R}^{d+2}$ (i.e., $\dim \mathcal{P}_1^{d+1} = d+2 = \text{ card } \Sigma^{Lag}$).

To establish injectivity, it is necessary to demonstrate that for any polynomial $p \in \mathcal{P}_1^{d+1}$ and if for all $\boldsymbol{\alpha} \in \mathcal{A}_1^{d+1}$, $p(\mathbf{a}_{\boldsymbol{\alpha}}) = 0$ where $\mathbf{a}_{\boldsymbol{\alpha}} \in \mathbb{R}^{d+1}$, then p itself must be zero. This requirement is underpinned by the fact that the Lagrange polynomials $(\mathcal{L}_i^{d,1})_{i\in[0..d+1]}$ constitute a linearly independent family for $\mathcal{P}_1^{d+1}$, as established by the lemma `LagPd1_cur_lin_indep` in Section 9.5. Therefore, any polynomial p can be expressed as a linear combination of the elements of this family, written as $p = \sum_{i=0}^{d+1} c_i \mathcal{L}_i^{d,1}$ where $\boldsymbol{c} \in \mathbb{R}^{d+2}$. By applying the lemma `LagPd1_cur_kron_node`, as detailed in Section 9.5, we obtain for all $\boldsymbol{\alpha} \in \mathcal{A}_1^{d+1}$,

$$0 = p(\mathbf{a}_{\boldsymbol{\alpha}}) = \sum_{i=0}^{d+1} c_i \mathcal{L}_i^{d,1}(\mathbf{a}_{\boldsymbol{\alpha}}) = \begin{cases} \sum_{i=0}^{d+1} c_i \mathcal{L}_i^{d,1}(\mathbf{a_0}) = c_0 \\ \sum_{i=0}^{d+1} c_i \mathcal{L}_i^{d,1}(\mathbf{a}_{\boldsymbol{\delta}_{j-1}}) = c_j \quad \text{for all } j \in [1..d+1]. \end{cases}$$

This result confirms that each coefficient c_j must be zero for all $j \in [0..d+1]$. Hence, we deduce that $p = 0$, proving that the polynomial space $\mathcal{P}_1^{d+1}$ is unisolvent at degree one. □

10.3.5 Unisolvence of the $\mathbb{P}_k^1$ Lagrange Finite Element

The proof of the unisolvence for higher-degree polynomials of $\mathbb{P}_k^1$ Lagrange finite elements when the spatial dimension $d = 1$ is similar to the previous case when $k = 1$.

```
Lemma unisolvence_LagP1k_cur : ∀k vtx_cur,
  0 < k → affine_independent vtx_cur →
 bijS (Pdk 1 k) fullset (gather (Sigma_LagPdSk_cur 1 k vtx_cur)).
```

The proof is based on the fact that $(\mathcal{L}_i^{1,k})_{i\in[0..k]}$ is a basis of the space $\mathcal{P}_k^1$, and that $\dim \mathcal{P}_k^1 = k + 1 =$ card Σ^{Lag}.

10.3.6 Factorization of Polynomials

Before tackling the main topic of this chapter, the unisolvence proof of the $\mathbb{P}_k^d$ Lagrange finite element [36, Section 7.4 p.79], it is necessary to establish essential theorems that address the factorization of polynomials that vanish on hyperplanes [16, Section 3.1 p.71]. These theorems provide essential support for understanding and proving the unisolvence property of this finite element.

The first lemma focuses specifically on how a polynomial can be factorized when it vanishes on the last reference hyperplane, $\hat{\mathcal{H}}_d^d$, of the reference simplex $\hat{K}$, detailed in Section 10.1. Let $\hat{p} \in \mathcal{P}_{k+1}^d$ for any integers d and k with $d > 0$, then we have the equivalence:

$$\hat{p}_{|\hat{\mathcal{H}}_d^d} = 0 \Longleftrightarrow \exists \hat{q} \in \mathcal{P}_k^d, \quad \hat{p} = \hat{\mathcal{L}}_d^{d,1} \times \hat{q}. \tag{10.19}$$

This is formalized in Coq as:

```
Lemma factorize_poly_zero_on_face_ref_d : ∀(d k : nat), 0 < d →
  ∀ p_ref : FRd d, Pdk d k.+1 p_ref →
 (∀ x_ref : 'R^d, face_ref_d d x_ref → p_ref x_ref = 0) ↔
  ∃ q_ref : FRd d, Pdk d k q_ref ∧ p_ref = (LagPd1_ref d ord_max) * q_ref.
```

The proof of this lemma unfolds in two parts, each proving a direction of the equivalence.

($\Rightarrow$) We start by assuming for every $\hat{\mathbf{x}} \in \hat{\mathcal{H}}_d^d$, the condition $\hat{p}(\hat{\mathbf{x}}) = 0$ holds. Proceeding by cases on the dimension d, the trivial case $d = 0$ is immediately satisfied, because of the hypothesis $d > 0$. For $d + 1$, using the decomposition property of polynomials as described by Equation (8.18) in Section 8.2.3, for all $\hat{p} \in \mathcal{P}_{k+1}^{d+1}$, there exists two polynomials $\tilde{p}_0 \in \mathcal{P}_{k+1}^d$ and $\hat{q} \in \mathcal{P}_k^{d+1}$, leading to the expression:

$$\hat{p} = \tilde{p}_0 + \hat{x}_d \; \hat{q}.$$

Thus, for any $\hat{\mathbf{x}} = (\hat{x}_0, \ldots, \hat{x}_{d-1}, 0) \in \mathbb{R}^{d+1}$ from (10.2), we have $\hat{\mathbf{x}} \in \hat{\mathcal{H}}_{d+1}^{d+1}$ and

$$\hat{p}(\hat{\mathbf{x}}) = \tilde{p}_0(\hat{x}_0, \ldots, \hat{x}_{d-1}) + 0 \; \hat{q}(\hat{x}_0, \ldots, \hat{x}_{d-1}, 0).$$

From the assumption, it follows that $\tilde{p}_0(x_0, \ldots, x_{d-1}) = 0$. This result enables us to establish the factorization relationship, which is derived from the definition of the reference Lagrange polynomial in Section 9.2, as shown in Equation (9.11):

$$\hat{p} = \hat{x}_d \times \hat{q} = \hat{\mathcal{L}}_{d+1}^{d+1,1} \times \hat{q}.$$

($\Leftarrow$) Assuming the existence of $\hat{q} \in \mathcal{P}_k^d$, such that $\hat{p} = \hat{\mathcal{L}}_d^{d,1} \times \hat{q}$, the proof is immediate. Given $\hat{\mathbf{x}} \in \hat{\mathcal{H}}_d^d$ from Equation (10.2), we have $\hat{\mathcal{L}}_d^{d,1}(\hat{\mathbf{x}}) = 0$ by definition, the multiplication by zero ensures that the polynomial $\hat{p}$ necessarily vanishes on $\hat{\mathcal{H}}_d^d$.

□

The second lemma addresses the factorization of a polynomial that vanishes on the zeroth hyperplane $\mathcal{H}_0^d$ on a current nondegenerate simplex K [36, Lemma 7.10 p.78], as defined in Section 10.1. Let $p \in \mathcal{P}_{k+1}^d$ for a dimension d and the polynomial degree k, then we have the equivalence:

$$p_{|\mathcal{H}_0^d} = 0 \iff \exists q \in \mathcal{P}_k^d, \quad p = \mathcal{L}_0^{d,1} \times q. \tag{10.20}$$

```
Lemma factorize_poly_zero_on_face0 :
  ∀ (d k : nat) (vtx_cur : 'R^{d.+1,d}) (Hvtx: affine_independent vtx_cur),
  0 < d → ∀p : FRd d, Pdk d k.+1 p →
  (∀ x : 'R^d, face0 d vtx_cur Hvtx x → p x = 0) ↔
    ∃ q : FRd d, Pdk d k q ∧ p = (LagPd1_cur vtx_cur Hvtx ord0) * q.
```

The proof of this lemma follows by demonstrating both implications of the equivalence.

($\Rightarrow$) We assume that for every $\mathbf{x} \in \mathcal{H}_0^d$ (i.e., $\mathcal{L}_0^{d,1}(\mathbf{x}) = 0$), $p(\mathbf{x}) = 0$ is satisfied.

We pose $\hat{p} \overset{\text{def.}}{=} p \circ \phi_{\pi_0^d}^K$ such that the polynomial p is transformed by the geometric transformation $\phi_{\pi_0^d}^K$. This transformation maintains $\hat{p}$ within $\mathcal{P}_{k+1}^d$ using the lemma `T_geom_d_0_compose` outlined in Section 10.2.2.

Using the previously established lemma `factorize_poly_zero_on_face_ref_d`, $\hat{p}$ is factorized as the product of another polynomial $\hat{q} \in \mathcal{P}_k^d$ and the reference Lagrange polynomial, such that $\hat{p} = \hat{\mathcal{L}}_d^{d,1} \times \hat{q}$. This is applicable under the condition that $\hat{p}$ vanishes on the last reference hyperplane $\hat{\mathcal{H}}_d^d$. This condition is verified since for all $\hat{\mathbf{x}} \in \hat{\mathcal{H}}_d^d$, $\phi_{\pi_0^d}^K(\hat{\mathbf{x}}) \in \mathcal{H}_0^d$ deduced from the Equation (10.15), then we have by assumption $\hat{p}(\hat{\mathbf{x}}) = p \circ \phi_{\pi_0^d}^K(\hat{\mathbf{x}}) = 0$.

Setting $q = \hat{q} \circ (\phi_{\pi_0^d}^K)^{-1}$ and applying the lemma `T_geom_d_0_inv_compose` established in Section 10.2.2, we obtain $q \in \mathcal{P}_k^d$. Subsequently, the polynomial p can be expressed as:

$$p = \hat{p} \circ (\phi_{\pi_0^d}^K)^{-1} = (\hat{\mathcal{L}}_d^{d,1} \times \hat{q}) \circ (\phi_{\pi_0^d}^K)^{-1} = \hat{\mathcal{L}}_d^{d,1} \circ (\phi_{\pi_0^d}^K)^{-1} \times \hat{q} \circ (\phi_{\pi_0^d}^K)^{-1}.$$

According to lemma `LagPd1_cur_T_geom_d_0_inv` (refer to Equations (10.16) and (10.11) in Section 10.2.2), we have

$$p = \mathcal{L}_0^{d,1} \times \hat{q} \circ (\phi_{\pi_0^d}^K)^{-1} = \mathcal{L}_0^{d,1} \times q.$$

($\Leftarrow$) This implication is straightforward. Assuming the existence of $q \in \mathcal{P}_k^d$ such that $p = \mathcal{L}_0^{d,1} \times q$. Given $\mathbf{x} \in \mathcal{H}_0^d$, since $\mathcal{L}_0^{d,1}(\mathbf{x}) = 0$ by definition, the multiplication by zero ensures the polynomial p vanishes on $\mathcal{H}_0^d$.

□

10.3.7 Unisolvence of the $\mathbb{P}_k^d$ Lagrange Finite Elements

We now proceed to establish the unisolvence property for the $\mathbb{P}_k^d$ Lagrange finite element, where the geometric element K is defined by a simplex with affinely independent vertices $(\mathbf{v}_i)_{i\in[0..d]}$. In this context, both the spatial dimension d and the polynomial degree k are greater than zero.

```
Theorem unisolvence_LagPdSk_cur : ∀d k (vtx_cur :'R^{d.+1,d}),
  0 < d → 0 < k → affine_independent vtx_cur →
  bijS (Pdk d k) fullset (gather (Sigma_LagPdSk_cur d k vtx_cur)).
```

The proof begins by defining a predicate P that captures the statement of the theorem for all $d > 0$ and $k > 0$,

$$P(d,k) \stackrel{\text{def.}}{=} \left[\forall (\mathbf{v}_i)_{i\in[0..d]} \in \mathbb{R}^d \text{ affinely independent }, \ \Phi_{\Sigma^{Lag}} : \mathcal{P}_k^d \to \mathbb{R}^{n_{\text{dof}}} \text{ is bijective } \right].$$

Here, $\Phi_{\Sigma^{Lag}}$ represents the mapping associated with the family of degrees of freedom Σ^{Lag}, as defined in Section 10.3.2. We proceed by applying a double induction technique `nat_ind2_alt_11`, first on the spatial dimension d and subsequently on the polynomial degree k, as detailed in Section 5.2.1.

(i) **Base Case: for all** $k \geq 1, P(1,k)$**.**
This case follows directly from the lemma `unisolvence_LagP1k_cur`, which has been proven in Section 10.3.5. This lemma establishes unisolvence for 1-dimensional spaces for all degrees $k \geq 1$.

(ii) **Base Case: for all** $d \geq 1, P(d,1)$**.**
Similarly, this assertion is supported by the lemma `unisolvence_LagPd1_cur`, demonstrated in Section 10.3.4. This lemma establishes the unisolvence property for affine polynomials for any spatial dimension $d \geq 1$.

(iii) **Double induction: for all** $d,k \geq 1, P(d,k+1) \wedge P(d+1,k) \to P(d+1,k+1)$**.**

Consider $d \geq 1$ and $k \geq 1$. Assuming that $P(d,k+1)$ and $P(d+1,k)$ hold, we aim to establish $P(d+1,k+1)$. Let $(\mathbf{v}_i)_{i\in[0..d+1]}$ be a family of $d+2$ affinely independent points in $\mathbb{R}^{d+1}$. By employing the lemma `lmS_bijS_val_gather_equiv` from Section 5.4.2, it is established that the mapping $\Phi_{\Sigma^{Lag}} : \mathcal{P}_{k+1}^{d+1} \to \mathbb{R}^{n_{\text{dof}}}$ is bijective if it is injective, provided that the dimension of $\mathcal{P}_{k+1}^{d+1}$ aligns with that of $\mathbb{R}^{n_{\text{dof}}}$ since

$$\dim \mathcal{P}_{k+1}^{d+1} = \text{ card } \Sigma^{Lag} = n_{\text{dof}} = \binom{d+k+2}{k+1}.$$

Let $p \in \mathcal{P}_{k+1}^{d+1}$ be such that $p(\mathbf{a}_{\boldsymbol{\alpha}}) = 0$ for all the Lagrange nodes $(\mathbf{a}_{\boldsymbol{\alpha}})_{\boldsymbol{\alpha}\in\mathcal{A}_{k+1}^{d+1}}$ as defined in the Equation (9.3) in Section 9.1.2. We prove that $p = 0$.

- **Factorization using** $P(d,k+1)$**:** This process begins by applying the property $P(d,k+1)$ to the reference vertices $(\hat{\mathbf{v}}_i)_{i\in[0..d]}$, which are affinely independent as discussed in Section 9.1.1. We define $p_0 \stackrel{\text{def.}}{=} p \circ \phi_{\theta_0^d}$. According to the lemma `T_geom_face0_compose` established in Section 10.2.1, p_0 belongs to $\mathcal{P}_{k+1}^d$. Furthermore, from (8.6) in Section 8.1.1 and the lemma `T_geom_face0_map_node` in section 10.2.1, it is established that for all $\boldsymbol{\beta} \in \mathcal{A}_{k+1}^d$, $f_{k+1,0}^{d+1}(\boldsymbol{\beta}) \in \mathcal{C}_{k+1}^{d+1} \subset \mathcal{A}_{k+1}^{d+1}$, leading to

$$\forall \boldsymbol{\beta} \in \mathcal{A}^d_{k+1}, \quad p_0(\hat{\mathbf{a}}_{\boldsymbol{\beta}}) = p \circ \phi_{\theta^d_0}(\hat{\mathbf{a}}_{\boldsymbol{\beta}}) = p(\mathbf{a}_{f^{d+1}_{k+1,0}(\boldsymbol{\beta})}) = 0.$$

By applying lemma `lmS_bijS_val_gather_equiv` on the property $P(d, k+1)$, we conclude that $p_0 = 0$. Given that $\phi_{\theta^d_0} : \mathbb{R}^d \to \mathcal{H}^{d+1}_0$ is bijective (refer to Lemma `T_geom_face0_bijS` in Section 10.2.1), it follows that for every $\mathbf{x} \in \mathcal{H}^{d+1}_0$,

$$p(\mathbf{x}) = p_0 \circ \phi^{-1}_{\theta^d_0}(\mathbf{x}) = 0 \quad (\text{i.e., } p_{|\mathcal{H}^{d+1}_0} = 0).$$

Then, using the previously established lemma `factorize_poly_zero_on_face0` in Section 10.3.6, p is factorized into the product of another polynomial $q \in \mathcal{P}^{d+1}_k$ and the zeroth current Lagrange polynomial, resulting in the expression $p = \mathcal{L}^{d,1}_0 \times q$.

- **Cancellation using $P(d+1, k)$.** We apply the property $P(d+1, k)$ to sub-vertices $(\check{\mathbf{v}}_i)_{i\in[0..d+1]}$, which are affinely independent as detailed in Section 9.1.4. Then, for all $\boldsymbol{\gamma} \in \mathcal{A}^{d+1}_k \subset \mathcal{A}^{d+1}_{k+1}$, according to lemma `sub_node_cur_eq` and Euqation (9.10) from Section 9.1.4, we have $\check{\mathbf{a}}_{\boldsymbol{\gamma}} = \mathbf{a}^{\check{\mathbf{v}}}_{\boldsymbol{\gamma}}$. Therefore,

$$p(\check{\mathbf{a}}_{\boldsymbol{\gamma}}) = \mathcal{L}^{d,1}_0(\check{\mathbf{a}}_{\boldsymbol{\gamma}}) \times q(\check{\mathbf{a}}_{\boldsymbol{\gamma}}) = 0.$$

where the Lagrange sub-nodes $(\check{\mathbf{a}})_{\boldsymbol{\gamma}\in\mathcal{A}^{d+1}_k}$ are defined in the Equation (9.10) in Section 9.1.4. Based on the lemma `sub_node_out_face0` discussed in Section 10.1, $\check{\mathbf{a}}_{\boldsymbol{\gamma}} \notin \mathcal{H}^{d+1}_0$, implying $\mathcal{L}^{d,1}_0(\check{\mathbf{a}}_{\boldsymbol{\gamma}}) \neq 0$. This leads us to conclude that $q(\check{\mathbf{a}}_{\boldsymbol{\gamma}}) = 0$. By reapplying the lemma `lmS_bijS_val_gather_equiv` on the property $P(d+1, k)$, we deduce that $q = 0$, which subsequently provides $p = 0$.

Finally, we have established the unisolvence principle, for all approximation degrees $k \geq 1$, and spatial dimensions $d \geq 1$.
□

The Lagrange linear forms, denoted Σ^{Lag} are defined as gathering both the base case when $k = 0$ and the general case for $k > 0$, as illustrated below:

```
Definition Sigma_LagPdk_cur : '(FRd d → R)^(ndof_LagPdk d k)
 := match k with
   | 0 ⇒ Sigma_0 d vtx_cur
   | S n ⇒ Sigma_LagPdSk_cur d (S n) vtx_cur
end.
```

Each component of Σ^{Lag} is a linear map. This assertion is based on the established linearity of Σ^0 and Σ^n, which are defined in Sections 10.3.3 and 10.3.1, respectively.

```
Lemma Sigma_LagPdk_cur_lm : ∀i : 'I_(ndof_LagPdk d k), lin_map (Sigma_LagPdk_cur i).
```

We prove the unisolvence property of these linear forms for both cases. Specifically, for the case when $k = 0$, unisolvence is established by the lemma `unisolvence_LagPd0`, as detailed in Section 10.3.3. For cases when $k \geq 1$, the property is substantiated by the lemma `unisolvence_LagPdSk_cur`, which is thoroughly discussed earlier.

```
Lemma unisolvence_LagPdk_cur : bijS (Pdk d k) fullset (gather Sigma_LagPdk_cur).
```

Finally, the simplicial current Lagrange finite element is constructed as a record of type FE as follows:

```
Definition FE_LagPdk_cur :=
  mk_FE d (nnode_LagPdk d k) d_pos (nnode_LagPdk_pos d k)
  Simplex vtx_cur (Pdk d k) (Pdk_has_dim_ndof_LagPdk d k)
  Sigma_LagPdk_cur Sigma_LagPdk_cur_lm unisolvence_LagPdk_cur.
```

where, `mk_FE` is the constructor of the record `FE` (see Section 7.1).

10.4 Construction of the Reference Simplicial Lagrange Finite Elements

This section focuses on the construction of the $\hat{\mathbb{P}}_k^d$ reference Lagrange finite element $(\hat{K}, \hat{\mathcal{P}}_k^d, \hat{\Sigma}^{Lag})$ within a d-dimensional space, where $d \geq 1$ and the polynomial approximation degree $k \geq 0$. This construction is generated from the current finite element of Lagrange using the `FE_LagPdk_cur` constructor detailed in the previous section 10.3, considering a family of reference vertices $(\hat{\mathbf{v}}_i)_{i\in[0..d]}$ as an argument, which are affinely independent as established by the lemma `vtx_simplex_ref_affine_independent` (see Section 9.1.1). The reference FE of Lagrange is formalized as:

```
Definition FE_LagPdk_ref := FE_LagPdk_cur d k d_pos (vtx_simplex_ref d)
  (vtx_simplex_ref_affine_independent d).
```

From this definition, we can derive all the fields from the record `FE_LagPdk_cur` for the reference FE of Lagrange. For example, the family of linear forms $\hat{\Sigma}^{Lag}$ can be defined as follows:

```
Definition Sigma_LagPdk_ref d k : '(FRd d → R)^(ndof_LagPdk d k) :=
  Sigma_LagPdk_cur d k (vtx_simplex_ref d).
```

The unisolvence of these forms follows directly from the record `FE_LagPdk_ref`.

In summary, this chapter was dedicated to the construction of both current and reference simplicial Lagrange finite elements, building on the concepts of nodal finite elements introduced in Section 10.3.1. The primary objective of this development is to establish a reliable and useful framework for the finite elements `FE`, which are formally defined as a record in Section 7.1. This framework includes detailed proofs to ensure that we are indeed capable of constructing a finite element.

In essence, we observe that transitioning from the reference finite element (FE) to the current FEs, as discussed in Section 9.4, could have been an interesting approach for constructing simplicial Lagrange finite elements. Specifically, this method would involve initially constructing the reference Lagrange FE, followed by the current Lagrange FE using the affine geometric transformation. However, the proof strategy outlined in Section 10.3.7 to establish the unisolvence of the $\mathbb{P}_k^d$ Lagrange finite elements cannot be directly applied to the construction of the reference Lagrange FE. This is because, during the second step of the proof (the cancellation phase), the induction property depends on sub-vertices. Sub-vertices of a reference geometric element do not qualify as valid reference vertices. Consequently, it is more practical to state and prove the unisolvence theorem for the current FE first, and subsequently derive properties for the reference FE.

Chapter 11

Conclusions and Perspectives

Conclusions

This thesis centers on the formalization of mathematical concepts using the Coq proof assistant, focusing primarily on the Tonelli theorem and simplicial finite elements within the finite element method (FEM). The first part of this work involved formalizing the Tonelli Theorem, beginning with the Lebesgue Induction Principle. We represented the set of nonnegative measurable functions as an inductive type and established the equivalence of various inductive types, such as `SFp` and `SFplus`, along with `Mp` and `Mplus`. Delving into the intricate details of product measures on product spaces, we specified and constructed the product σ-algebras, demonstrating the measurability and uniqueness of these measures. This work culminated in a comprehensive formal proof of the Tonelli theorem, including the formalization of iterated integrals and changes in measure to swap the roles of the variables.

In the second part of the thesis, we shifted our focus to the simplicial finite elements of the FEM, beginning with an exploration of basic algebraic properties and progressing to the formalization of finite elements. We have thoroughly specified and formalized a range of algebraic concepts and structures in Coq, with a particular emphasis on the finite families and the formalization of some algebraic structures, including monoids and affine spaces. Further, the study continued with a brief analysis of binomial coefficients, essential for computing the dimensions of certain polynomial spaces and the cardinality of specific families. After laying the foundational algebraic properties, we proceeded to develop a detailed mathematical framework to understand the finite element method. This passage was provided to enhance our comprehension of FEM, although not all concepts presented were formalized in Coq. We carried this out specifically through the study of the strong and weak formulations of the Poisson problem. We then discussed the Lax-Milgram theorem to state the existence and uniqueness of the solution to the problem being addressed. We highlighted the transition from continuous to discrete formulations by discretizing the domain of our problem to construct a mesh. This mesh was carefully designed to meet the criteria of a conforming mesh, setting the stage for the final step of solving the linear system of equations to obtain the approximate solution to the problem. Furthermore, we explored the Coq formalization of finite elements (FE) defined as a triplet $(K, \mathcal{P}, \Sigma)$ within individual mesh cells using a record-based approach, where K represents the simplicial geometric element, $\mathcal{P}$ the polynomial approximation space, and Σ the family of linear forms. Each component of the finite element triplet was constructed and examined in more detail. We then defined local shape functions as a basis for the space $\mathcal{P}$ to approximate the solution. We concluded with an emphasis on constructing a local interpolation operator associated with this finite element. The work further progressed with the formalization of the

polynomial space $\mathcal{P}_k^d$, which comprises polynomials of degrees less than k on $\mathbb{R}^d$. This space was constructed as linear combinations of monomials indexed by the family of multi-indices $\mathcal{A}_k^d$. We explored the ordering of monomials within this family and established a monomial basis, $\mathcal{B}^{d,k}$, for $\mathcal{P}_k^d$. Additionally, we reviewed key properties of the $\mathcal{P}_k^d$ spaces and provided formal definitions of the Lagrange polynomial spaces $\mathcal{P}_1^d$ and $\mathcal{P}_k^1$ when either the degree k or the spatial dimension d is 1. We constructed the Lagrange bases of these spaces and thoroughly examined how these polynomials are formalized in Coq and used across both reference and current geometrical elements. Following this, we continued our study to the simplicial geometric aspects of K, defining both reference vertices and Lagrange nodes in reference and current element, where solution values are evaluated. We established connections between vertices and nodes and introduced the concepts of sub-vertices and sub-nodes. Additionally, we formalized the affine geometric transformations of finite elements and their inverse functions, which map the reference element to current elements and vice versa. This affine mapping set the stage for the construction of current finite elements from the reference finite elements. Moving forward to the culmination of our work, we focused exclusively on the construction of the simplicial Lagrange finite element in both current and reference geometric elements. This final phase involved the detailed exploration of each component of the triplet $(K, \mathcal{P}_k^d, \Sigma^{Lag})$ which was built upon the concepts of nodal finite elements. Through these efforts, we provided a comprehensive formal proof of the fundamental property of unisolvence of Lagrange degrees of freedom. These linear forms were constructed and represented as the evaluation of functions at specific nodes. The formal proof of unisolvence was notably complicated, involving the use of all previously mentioned properties and additional elements. For instance, it included the construction of some affine geometric mappings essential for transforming nodes from the reference geometric element in dimension $d-1$ to a hyperplane of the current geometric element. Overall, these developments aim at fulfilling the main objective of the thesis: to establish a robust, reliable mathematical framework for finite elements that can pave the way to be effectively applied in computational simulations without inconsistencies. Constructing the Lagrange finite element ensures there are no inconsistencies inside the record. Through detailed formalization, particularly in the implementation and verification of finite elements and their properties, this collaborative research has resulted in significant developments within the `coq-num-analysis` library in Coq. This includes the creation of the `Lebesgue` directory, which contains a total of 34 kloc (lines of code), and the new `FEM` directory, consisting of 32 kloc.

The work presented in this thesis contributes to both the formal proof community and the field of numerical analysis, with a focus on improving the reliability and accuracy of finite element method (FEM) simulations by enabling the identification and correction of potential errors. Additionally, we aim to make these developments accessible and usable by others, which presents its own challenges. This is not simply a matter of providing sufficient lemmas, but rather of formulating and proving them in a way that is easily applicable, whether by computer scientists or numerical analysts. Achieving this often requires revisiting and refining the proofs to make them clearer and more comprehensible. By integrating formal proofs with mathematical numerical techniques such as FEM, this work offers a framework that can be taught and incorporated into academic programs as a long-term goal. This approach will not only deepen the understanding of FEM but also help future engineers and scientists recognize the importance of combining formal proofs with numerical computations to achieve precise and reliable results.

Throughout this thesis, multiple challenges have been encountered that require solutions. These challenges primarily revolved around the formalization process within the Coq proof as-

sistant. A significant initial hurdle involved determining the appropriate formulations for definitions and theorems. Often, after defining a concept or stating a lemma, we would realize that it did not meet the intended criteria, prompting frequent revisions and re-proofs. An example of this was the formalization of the Lebesgue induction principle, reffered in Coq as `Mp_ind`, to support the proof of the Tonelli theorem. The complexity of this task stemmed from the absence of this principle in the existing literature, along with challenges in its definition. Initially, it was considered in terms of simple functions, which complicated the formulation. However, it was ultimately defined as an inductive type applicable to all measurable functions. This decision was important because it influenced the scope and applicability of the theorem within our formalizations. Another challenge was related to the algebraic structures required for formalizing finite elements. We initially sought to leverage existing libraries, such as the Mathematical Components (math-comp) library to nourish our development, which is a great source of insight into the formalization of mathematics. We used their formalizations of finite types (ordinals), iterated operators (`bigop`), and binomial coefficients. However, we encountered compatibility issues between the canonical structures of Coquelicot and those defined within math-comp. Additionally, our work required the use of multinomials and infinite-dimensional vector spaces, concepts that, to the best of our knowledge, were not fully developed in the math-comp library. These conflicts introduced significant challenges in the development of our algebraic structures.

Future Works

As discussed in Part I of the thesis, the Tonelli Theorem is applied to manage multivariate integrals, facilitating the extension of FEM to address problems in higher dimensions. To effectively handle PDEs involving multiple variables and functions of arbitrary signs, an extended version of this theorem, known as the Tonelli-Fubini theorem, will be utilized. Future work will aim to formally prove this theorem in Coq, which requires extending the properties used to prove the Tonelli theorem, to integrable functions of arbitrary signs.

Additionally, as elaborated in Chapter 6, FEM is particularly applicable to the weak formulations of PDEs, which involve integrals and function spaces. They notably require the use of Sobolev spaces $W^{k,p}$ [1, 51]. These spaces are extensively used in numerous fields, including functional analysis [72, 17, 67], and statistical and probabilistic mathematics [37, 8, 32], they are not yet developed in Coq but are planned for future work. Sobolev spaces are essential for handling the regularity of solutions to PDEs, helping in the analysis of the differentiability and integrability of solutions. This appears in the study of the case of the Poisson Equation as discussed in Section 6.1.3), where solutions are often explored within a specific subspace of the H^1 Sobolev space (see e.g. [35, Section 3.2 p.120] for details). We recall that the H^1 space is defined to include functions in L^2 having a weak derivative in L^2. The construction of Sobolev spaces involves developing and formalizing also in Coq the L^p Lebesgue spaces as Banach spaces, which are complete normed vector spaces. They are defined as the quotient of $\mathcal{L}^p$, the space of functions whose absolute values raised to the power $p \geq 1$ are integrable, by the subspace of functions that are zero almost everywhere. A significant challenge in this formalization process lies in handling the quotient structure and its associated class of equivalence. Future work will focus on defining and verifying that these spaces meet all criteria to be considered complete normed vector spaces.

Following the formalization of simplicial finite elements, the next phase of this research involves the formalization of quadrangular finite elements, specifically those of Lagrange. The scope may further expand to include formal verification of other finite element types such as Crouzeix–Raviart, Raviart–Thomas, and Nédélec [34]. Subsequently, an intriguing area of future

work will involve exploring simple *quadrature formulas* and establishing their formal proofs. Quadrature formulas are numerical methods used to approximate the integral of a function. They are needed in the future for the computational aspects of this work, particularly in solving the linear system required to find an approximate solution to the model under study [34, Chapter 9]. To illustrate these concepts, consider the Poisson equation, expressed as $-\Delta u = f$ over a domain Ω with suitable boundary conditions. Although this problem appears straightforward, it encapsulates the fundamental attributes of a broad class of elliptic PDEs. The solution process involves meshing Ω into geometric elements K with specific properties and applying the standard Finite Element Method (FEM) with conventional finite elements, such as Lagrange elements on a conforming mesh of Ω. This process requires calculating expressions like:

$$\begin{aligned}\int_\Omega \boldsymbol{\nabla}\phi_j \cdot \boldsymbol{\nabla}\phi_i &= \sum_{K\in\mathcal{T}_h} \int_K \boldsymbol{\nabla}\phi_j \cdot \boldsymbol{\nabla}\phi_i \\ &= \sum_{K\in\mathcal{T}_h} \int_{\hat{K}} (\boldsymbol{J}^K_{\text{geo}})^{-T}\hat{\boldsymbol{\nabla}}\hat{\phi}_j \cdot (\boldsymbol{J}^K_{\text{geo}})^{-T}\hat{\boldsymbol{\nabla}}\hat{\phi}_i \,|\det \boldsymbol{J}^K_{\text{geo}}| \\ &\approx \sum_{K\in\mathcal{T}_h} \sum_{i_g=1}^{N_q} \left((\boldsymbol{J}^K_{\text{geo}})^{-T}\hat{\boldsymbol{\nabla}}\hat{\phi}_j \cdot (\boldsymbol{J}^K_{\text{geo}})^{-T}\hat{\boldsymbol{\nabla}}\hat{\phi}_i \,|\det \boldsymbol{J}^K_{\text{geo}}|\right)(\hat{\boldsymbol{\xi}}_{i_g})\,\hat{\omega}_{i_g}.\end{aligned} \tag{11.1}$$

Here, $(\phi_i)_{i\in[1..N_{\text{dof}}]}$ represents a basis of the approximation space V_h and $\boldsymbol{J}^K_{\text{geo}}$ is the Jacobian matrix transforming the reference element $\hat{K}$ to the current element K. The points $\hat{\boldsymbol{\xi}}_{i_g}$ and weights $\hat{\omega}_{i_g}$ are the Gaussian quadrature points and weights on $\hat{K}$. Achieving this goal requires constructing and formalizing a conforming mesh $\mathcal{T}_h$ in Coq and defining the corresponding ranges of quadrature formulas as:

$$\int_{\hat{K}} f(x) \approx \sum_{i_g} f(x_{i_g})\omega_{i_g}.$$

Additionally, another focus is on how to generate accurate quadrature formulas by construction, accompanied by a proof or certificate of correctness. Besides, considering the real programs compute using floating-point numbers, we therefore wish to bound rounding errors [60]. Potential methods include extracting a program from Coq proof λ-term or `Why3`[1] environment or incorporating annotations within the `C++` code to specify properties that are subsequently verified using tools like `Frama–C` [27], `Why3` [39], `Gappa` [28], or Coq. This approach, similar to one previously applied to a one-dimensional wave equation as documented in [12], aims to deepen our understanding of integrating formal proofs with computational programs. The collective efforts described in this thesis are aimed towards a significant long-term objective: the formal verification of scientific computing programs and parts of `C++` libraries that implement the Finite Element Method (FEM), particularly those such as FreeFEM++[2] and XLiFE++[3]. FreeFEM++ is a widely used open-source package that provides a high level of abstraction to easily solve partial differential equations via FEM, and XLiFE++ is another powerful tool that supports extended finite element methods, designed to handle a variety of problem types with high precision.

[1] https://www.why3.org/
[2] https://freefem.org/
[3] https://uma.ensta-paris.fr/soft/XLiFE++/

List of Figures

Chapter 12

Bibliography

[1] Robert A. Adams. *Sobolev Spaces*, volume 65 of *Pure and Applied Mathematics*. Academic Press, New York - San Francisco - London, 1975. URL: https://fr.scribd.com/document/373760111.

[2] Reynald Affeldt and Cyril Cohen. Measure construction by extension in dependent type theory with application to integration. *Journal of Automated Reasoning*, 67(3):28, 2023. URL: https://doi.org/10.48550/arXiv.2209.02345.

[3] Wolfgang Arendt and Mahamadi Warma. Dirichlet and Neumann boundary conditions: What is in between? *Nonlinear Evolution Equations and Related Topics: Dedicated to Philippe Bénilan*, pages 119–135, 2004. URL: https://doi.org/10.1007/978-3-0348-7924-8_6.

[4] Duane Arnold. The Sleipner Platform Disaster, 2021. Accessed: 2023-09-20. URL: https://www-users.cse.umn.edu/~arnold/disasters/sleipner.html.

[5] Robert G. Bartle. *A Modern Theory of Integration*, volume 32 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, 2001. URL: https://doi.org/10.1090/gsm/032.

[6] Yves Bertot. Formal verification of a geometry algorithm: a quest for abstract views and symmetry in Coq proofs. In *International Colloquium on Theoretical Aspects of Computing*, pages 3–10. Springer, 2018. URL: https://doi.org/10.1007/978-3-030-02508-3_1.

[7] Yves Bertot, Georges Gonthier, Sidi Ould Biha, and Ioana Pasca. Canonical big operators. In *International Conference on Theorem Proving in Higher Order Logics*, pages 86–101. Springer, 2008. URL: https://inria.hal.science/inria-00331193v1.

[8] Patrick Billingsley. *Probability and Measure*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, Inc., New York, 3rd edition, 1995. URL: https://www.colorado.edu/amath/sites/default/files/attached-files/billingsley.pdf.

[9] Sylvie Boldo, François Clément, Vincent Martin, Micaela Mayero, and Houda Mouhcine. Lebesgue Induction and Tonelli's Theorem in Coq. In *Formal Methods - 25th International Symposium, FM, Proceedings*, volume 14000 of *LNCS*, pages 39–55. Springer, 2023. URL: https://inria.hal.science/hal-03889276.

[10] Sylvie Boldo, François Clément, Florian Faissole, Vincent Martin, and Micaela Mayero. A Coq Formal Proof of the Lax–Milgram Theorem. In *Proc. of the 6th ACM SIGPLAN Internat. Conf. on Certified Programs and Proofs (CPP 2017)*, pages 79–89. Association for Computing Machinery, New York, 2017. URL: https://hal.inria.fr/hal-01391578/.

[11] Sylvie Boldo, François Clément, Florian Faissole, Vincent Martin, and Micaela Mayero. A Coq Formalization of Lebesgue Integration of Nonnegative Functions. *J. Autom. Reason.*, 66(2):175–213, 2021. URL: https://hal.inria.fr/hal-03471095/.

[12] Sylvie Boldo, François Clément, Jean-Christophe Filliâtre, Micaela Mayero, Guillaume Melquiond, and P. Weis. Wave Equation Numerical Resolution: a Comprehensive Mechanized Proof of a C Program. *J. Autom. Reason.*, 50(4):423–456, 2013. URL: https://hal.inria.fr/hal-00649240/.

[13] Sylvie Boldo, François Clément, Jean-Christophe Filliâtre, Micaela Mayero, Guillaume Melquiond, and Pierre Weis. Trusting computations: a mechanized proof from partial differential equations to actual program. *Comput. Math. with Appl.*, 68(3):325–352, 2014. URL: https://hal.inria.fr/hal-00769201/.

[14] Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Coquelicot: A user-friendly library of real analysis for Coq. *Mathematics in Computer Science*, 9:41–62, 2015. URL: https://doi.org/10.1007/s11786-014-0181-1.

[15] Sylvie Boldo and Guillaume Melquiond. Flocq: A Unified Library for Proving Floating-point Algorithms in Coq. In *Proc. of the IEEE 20th Symposium on Computer Arithmetic (ARITH-20)*, pages 243–252. IEEE, 2011. URL: https://doi.org/10.1109/ARITH.2011.40.

[16] Susanne Brenner and L. Ridgway Scott. *The mathematical theory of finite element methods*. Springer, 2008. URL: https://doi.org/10.1007/978-0-387-75934-0.

[17] Haïm Brezis. *Analyse fonctionnelle—Théorie et applications*. Collection Mathématiques Appliquées pour la Maîtrise. Masson, Paris, 1983. In French. URL: https://fr.scribd.com/doc/251853015.

[18] Frank Burk. *Lebesgue measure and integration: an introduction*. John Wiley & Sons, 2011.

[19] Philippe G. Ciarlet. *The Finite Element Method for Elliptic Problems*, volume 40 of *Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 2002. Reprint of the 1978 original [North-Holland, Amsterdam]. URL: https://doi.org/10.1137/1.9780898719208.

[20] François Clément and Vincent Martin. Lebesgue integration. Detailed Proofs to be Formalized in Coq. Research Report RR-9386, Inria, Paris, 2021. Version 2. URL: https://hal.inria.fr/hal-03105815v2.

[21] François Clément and Vincent Martin. Finite Element Method. Detailed proofs to be formalized in Coq. Research Report RR-9557, Inria, Paris, 2023. Version 1. URL: https://inria.hal.science/hal-04713897.

[22] Donald L. Cohn. *Measure Theory*. Birkhäuser, New York, 2nd edition, 2013. URL: https://doi.org/10.1007/978-1-4614-6956-8.

[23] The Coq Reference Manual. URL: https://coq.inria.fr/refman/.

[24] Thierry Coquand and Gerard Huet. The calculus of constructions. *Information And Computation*, 76(2-3):95–120, 1988. URL: https://inria.hal.science/inria-00076024/document.

[25] Patrick Cousot and Radhia Cousot. Systematic Design of Program Analysis Frameworks. In *Proceedings of the 6th Annual ACM Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, USA, 1979. ACM. doi:10.1145/567752.567778.

[26] Luís Cruz-Filipe, Herman Geuvers, and Freek Wiedijk. C-CoRN, the constructive Coq repository at Nijmegen. In *Mathematical Knowledge Management: Third International Conference, MKM 2004, Białowieża, Poland, September 19-21, 2004. Proceedings 3*, pages 88–103. Springer, 2004. URL: https://doi.org/10.1007/978-3-540-27818-4_7.

[27] Pascal Cuoq, Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. Frama-c: A software analysis perspective. In *International conference on software engineering and formal methods*, pages 233–247. Springer, 2012. URL: https://doi.org/10.1007/s00165-014-0326-7.

[28] Florent de Dinechin, Christoph Lauter, and Guillaume Melquiond. Assisted verification of elementary functions using Gappa. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, pages 1318–1322, Dijon, France, 2006. URL: http://www.lri.fr/~melquion/doc/06-mcms-article.pdf%7D.

[29] Leonardo De Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*, pages 378–388. Springer, 2015. URL: https://doi.org/10.1007/978-3-319-21401-6_26.

[30] Elif Deniz, Adnan Rashid, Osman Hasan, and Sofiène Tahar. On the formalization of the heat conduction problem in HOL. In *International Conference on Intelligent Computer Mathematics*, pages 21–37. Springer, 2022. URL: https://doi.org/10.48550/arXiv.2208.06642.

[31] Jean-François Dufourd and Yves Bertot. Formal study of plane Delaunay triangulation. In *International Conference on Interactive Theorem Proving*, pages 211–226. Springer, 2010. URL: https://doi.org/10.1007/978-3-642-14052-5_16.

[32] Rick Durrett. *Probability—Theory and Examples*, volume 49 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, Cambridge, 5th edition, 2019. URL: https://doi.org/10.1017/9781108591034.

[33] Noboru Endou. Fubini's Theorem. *Formaliz. Math.*, 27(1):67–74, 2019. URL: https://doi.org/10.2478/forma-2019-0007.

[34] Alexandre Ern. *Aide-mémoire des éléments finis*, volume 360. Dunod, 2005. In French. URL: http://idirsadani.d.i.f.unblog.fr/files/2010/06/elmentsfinis.pdf.

[35] Alexandre Ern and Jean-Luc Guermond. *Theory and Practice of Finite Elements*, volume 159 of *Applied Mathematical Sciences*. Springer, New York, 2004. URL: https://doi.org/10.1007/978-1-4757-4355-5.

[36] Alexandre Ern and Jean-Luc Guermond. *Finite elements I: Approximation and interpolation*, volume 72. Springer Nature, 2021. URL: https://doi.org/10.1007/978-3-030-56341-7.

[37] William Feller. *An Introduction to Probability Theory and its Applications. Vol. I.* John Wiley & Sons, Inc., New York - London - Sydney, 3rd edition, 1968. URL: https://archive.org/details/dli.ernet.5666/page/203/mode/2up.

[38] Miguel A. Fernández, Jean-Frédéric Gerbeau, and Vincent Martin. Numerical simulation of blood flows through a porous interface. *M2AN Math. Model. Numer. Anal.*, 42(6):961–990, 2008. URL: https://doi.org/10.1051/m2an:2008031.

[39] Jean-Christophe Filliâtre and Andrei Paskevich. Why3 – where programs meet provers. *Proceedings of the 22nd European Symposium on Programming*, 7792:125–128, 2013. doi: http://tertium.org/papers/esop-13.pdf.

[40] Sashikumaar Ganesan and Lutz Tobiska. *Finite Element Spaces*. Cambridge University Press, 2017. URL: https://doi.org/10.1017/9781108235013.004.

[41] Bernard Gostiaux. *Cours de mathématiques spéciales: Algèbre*, volume 1. Presses Universitaires de France-PUF, 1993. In French. URL: http://livre21.com/LIVREF/F12/F012073.pdf.

[42] Bernard Gostiaux. *Géométrie: arcs et nappes*, volume 5. Presses universitaires de France, 1995. In French. URL: https://archive.org/details/e-ramis.-cours-de-mathematiques-speciales-t-5.

[43] Cordelia V Hall, Kevin Hammond, Simon L Peyton Jones, and Philip L Wadler. Type classes in Haskell. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 18(2):109–138, 1996. URL: https://doi.org/10.1145/227699.227700.

[44] John Harrison. HOL Light: A tutorial introduction. In *International Conference on Formal Methods in Computer-Aided Design*, pages 265–269. Springer, 1996. URL: https://doi.org/10.1007/BFb0031814.

[45] John Harrison. Formal proof—theory and practice. *Notices of the AMS*, 55(11):1395–1406, 2008. URL: https://www.ams.org/notices/200811/tx081101395p.pdf.

[46] Johannes Hölzl and Armin Heller. Three Chapters of Measure Theory in Isabelle/HOL. In Marko van Eekelen, Herman Geuvers, Julien Schmaltz, and Freek Wiedijk, editors, *Proc. of the 2nd Internat. Conf. on Interactive Theorem Proving*, volume 6898 of *LNCS*, pages 135–151. Springer, Berlin - Heidelberg, 2011. URL: https://doi.org/10.1007/978-3-642-22863-6_12.

[47] Fabian Immler and Johannes Hölzl. Numerical analysis of ordinary differential equations in Isabelle/HOL. In *Interactive Theorem Proving: Third International Conference, ITP 2012, Princeton, NJ, USA, August 13-15, 2012. Proceedings 3*, pages 377–392. Springer, 2012. URL: https://doi.org/10.1007/978-3-642-32347-8_26.

[48] Claes Johnson. *Numerical solution of partial differential equations by the finite element method*. Courier Corporation, 2012. URL: https://archive.org/details/numericalsolutio0000john.

[49] Rahul Karmakar. Symbolic model checking: a comprehensive review for critical system design. *Advances in Data and Information Sciences: Proceedings of ICDIS 2021*, pages 693–703, 2022. URL: https://doi.org/10.1007/978-981-16-5689-7_62.

[50] Matt Kaufmann, Panagiotis Manolios, and J Strother Moore. *Computer-aided reasoning: ACL2 case studies*, volume 4. Springer Science & Business Media, 2013. URL: https://doi.org/10.1007/978-1-4757-3188-0.

[51] Semen Samsonovich Kutateladze. *Fundamentals of functional analysis*, volume 12. Springer Science & Business Media, 2013. URL: http://old.math.nsc.ru/LBRT/g2/english/ssk/fa_e.pdf.

[52] Peter D Lax. *Functional analysis*, volume 55. John Wiley & Sons, 2002. URL: http://users.math.uoc.gr/~frantzikinakis/FunctionalGrad2015/Lax.pdf.

[53] Henri Léon Lebesgue. *Leçons sur l'intégration et la recherche des fonctions primitives professées au Collège de France*. Cambridge University Press, Cambridge, 2009. Reprint of the 1904 original [Gauthier-Villars, Paris]. In French. URL: https://doi.org/10.1017/CBO9780511701825.

[54] David R Lester. Topology in PVS: continuous mathematics with applications. In *Proc. of the 2nd Workshop on Automated Formal Methods (AFM 2007)*, pages 11–20, 2007. URL: https://doi.org/10.1145/1345169.1345171.

[55] Anders Logg, Kent-Andre Mardal, and Garth Wells. *Automated solution of differential equations by the finite element method: The FEniCS book*, volume 84. Springer Science & Business Media, 2012. URL: https://doi.org/10.1007/978-3-642-23099-8.

[56] Assia Mahboubi and Enrico Tassi. Mathematical components. *Online book*, 2021. URL: https://doi.org/10.5281/zenodo.7118596.

[57] Micaela Mayero. *Formalisation et automatisation de preuves en analyses réelle et numérique*. Thèse de doctorat, Université Paris VI, 2001. In French. URL: https://www-lipn.univ-paris13.fr/~mayero/publis/these-mayero.ps.gz.

[58] Laura I Meikle and Jacques D Fleuriot. Automation for Geometry in Isabelle/HOL. In *PAAR@ IJCAR*, pages 84–94. Citeseer, 2010. URL: https://doi.org/10.29007/r5k7.

[59] Peter Monk. *Finite element methods for Maxwell's equations*. Oxford university press, 2003. URL: https://books.google.fr/books/about/Finite_Element_Methods_for_Maxwell_s_Equ.html?id=zI7Y1jT9pCwC&redir_esc=y.

[60] Jean-Michel Muller, Nicolas Brunie, Florent de Dinechin, Claude-Pierre Jeannerod, Mioara Joldes, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, and Serge Torres. *Handbook of Floating-Point Arithmetic*. Birkhauser Basel, 2 edition, 2018. doi:10.1007/978-3-319-76526-6.

[61] Adam Naumowicz and Artur Korniłowicz. A brief overview of Mizar. In *International Conference on Theorem Proving in Higher Order Logics*, pages 67–72. Springer, 2009. URL: https://doi.org/10.1007/978-3-642-03359-9_5.

[62] Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer, 2002. URL: https://doi.org/10.1007/3-540-45949-9.

[63] Sam Owre, John M Rushby, and Natarajan Shankar. PVS: A prototype verification system. In *International Conference on Automated Deduction*, pages 748–752. Springer, 1992. URL: https://doi.org/10.1007/3-540-55602-8_217.

[64] Pablo Pedregal. *Functional Analysis, Sobolev Spaces, and Calculus of Variations*. Springer, 2024. URL: https://doi.org/10.1007/978-3-031-49246-4.

[65] Frank Pfenning and Christine Paulin-Mohring. Inductively defined types in the Calculus of Constructions. In *Mathematical Foundations of Programming Semantics: 5th International Conference Tulane University, New Orleans, Louisiana, USA March 29–April 1, 1989 Proceedings 5*, pages 209–228. Springer, 1990. URL: https://doi.org/10.1007/BFb0040259.

[66] Alfio Quarteroni and Alberto Valli. *Numerical approximation of partial differential equations*, volume 23 of *Springer Series in Computational Mathematics*. Springer, Berlin, 1994. URL: https://doi.org/10.1007/978-3-540-85268-1.

[67] Walter Rudin. *Real and Complex Analysis*. McGraw-Hill Book Co., New York, 3rd edition, 1987. URL: https://59clc.wordpress.com/wp-content/uploads/2011/01/real-and-complex-analysis.pdf.

[68] Amokrane Saibi. Typing algorithm in type theory with inheritance. In *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 292–301, 1997.

[69] Mohit Tekriwal, Karthik Duraisamy, and Jean-Baptiste Jeannin. A formal proof of the Lax equivalence theorem for finite difference schemes. In *NASA Formal Methods Symposium*, pages 322–339. Springer, 2021. URL: https://doi.org/10.48550/arXiv.2103.13534.

[70] Gerald Teschl. *Topics in Real Analysis*. 2021. URL: https://www.mat.univie.ac.at/~gerald/ftp/book-ra/ra.pdf.

[71] Floris van Doorn. Formalized Haar Measure. In L. Cohen and C. Kaliszyk, editors, *Proc. of the 12th Internat. Conf. on Interactive Theorem Proving*, volume 193 of *LIPIcs*, pages 18:1–18:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. URL: https://doi.org/10.4230/LIPIcs.ITP.2021.18.

[72] Kōsaku Yosida. *Functional Analysis*. Classics in Mathematics. Springer, Berlin, 1995. Reprint of the 6th (1980) edition [Springer, Berlin - New York]. URL: https://doi.org/10.1007/978-3-642-61859-8.

[73] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu. *The Finite Element Method: its Basis and Fundamentals*. Elsevier/Butterworth Heinemann, Amsterdam, 7th edition, 2013. URL: https://doi.org/10.1016/C2009-0-24909-9.