Filomat 39:18 (2025), 6103–6121
https://doi.org/10.2298/FIL2518103I

Published by Faculty of Sciences and Mathematics,
University of Niš, Serbia
Available at: http://www.pmf.ni.ac.rs/filomat

Towards mathematical foundations of projects: Construction and formalization

Ivana D. Ilić[a,*], Jelena M. Višnjić[a]

[a]*Department of mathematics and informatics, Medical Faculty, Bulevar Dr. Zorana Djindjića 81, 18000 Nis, Serbia*

Abstract. This paper develops a formal axiomatic framework for Project Theory, introducing key concepts such as task dependencies, hierarchies, and project structures. The motivation behind this work is to achieve a global applicability by using universal mathematical language and logic, aiming to standardize the analysis of project management systems. We define projects in terms of well-founded relations and set-theoretic principles to rigorously describe their properties. Central to this theory are the Axiom of Compatibility, which ensures consistency across subprojects; the Axiom of Regularity, which prohibits infinite regress in project hierarchies and Mostowski's Collapsing Theorem, which is applied to project structures to establish isomorphisms between transitive sets and projects. These theoretical results provide a robust foundation for modeling complex project management systems, enabling applications in areas such as operations research, workflow optimization, and software engineering.

1. Introduction

1.1. *Some Projects Definitions*

A project can be defined in a variety of ways, revealing a range of perspectives. For example, some authors characterize a project as a goal that must be achieved while meeting specific requirements, including well-defined initiation and completion dates, financial boundaries, and the utilization of diverse resources such as finances, personnel, and equipment (see, for instance, Kerzner (1998) and Heagney (2016)).

The BS 6079 Guide to Project Management (2019) defines a project as a unique set of coordinated activities, with definite starting and finishing points, undertaken by an individual or organization to meet specific objectives within defined schedule, cost, and performance parameters. The Gower Handbook of Project Management (Turner 2014) states that a project is a cycle of activities with the purpose of definite start and completion dates, a unique product, service, or set of information to a specified quality and cost. The PMI Guide to the Project Management Body of Knowledge (2012) defines a project as a temporary endeavor

2020 *Mathematics Subject Classification*. Primary 03E75; Secondary 03E65, 03E30, 03E47.
Keywords. Set theory, axiomatization, project theory, modeling project systems, project revision.
Received: 07 October 2024; Revised: 12 March 2025; Accepted: 19 March 2025
Communicated by Dragan S. Djordjević
Research supported by the Ministry of Science, Technological Development and Innovation of the Republic of Serbia under Grant No. 451-03-137/2025-03/200113.
* Corresponding author: Ivana D. Ilić
Email addresses: ivanailicc3@gmail.com (Ivana D. Ilić), jvisnjic@gmail.com (Jelena M. Višnjić)
ORCID iDs: https://orcid.org/0000-0002-6887-3268 (Ivana D. Ilić), https://orcid.org/0000-0002-1580-6969 (Jelena M. Višnjić)

undertaken to create a unique product or service. In recent years, several studies have contributed to the formalization of complex systems, providing a stronger mathematical foundation for various applications, including project management. For example, Park (2021) discusses approaches to discovering and justifying mathematical axioms. Although not directly related to project management, this work provides insights into the foundational aspects of axiomatization that are relevant to various fields, including project theory. Similarly, Bagaria (2023) examines how axioms serve as the foundation for constructing mathematical systems, which is particularly relevant for establishing the mathematical foundations of project theories.

However, previous research has primarily analyzed mathematical axioms and the theory of projects and complex systems from a broader theoretical perspective. By comparison, this work advances the theory to a deeper level by applying it more concretely to projects. Furthermore, this study offers a more explicit and structured application of the axiomatic framework, enriched with numerous examples that illustrate its practical relevance and implications.

1.2. *Why Axiomatic Project Theory*

Intuitively, a project is a collection of all tasks that satisfy a certain property. In other words, we might be tempted to postulate the following rule of formation for projects:

Axiom Schema of Comprehension. If ϕ is a project property, then there exists a project $Y = \{x : \phi(x)\}$.

However, we'll show that this principle of project construction is incomplete. In fact, we'll prove that there are certain project properties that cannot be aggregated into a project.

In what follows, we establish project theory as a formal mathematical theory, in order to define the vague notion of a project property and also to distinguish those project properties that lead to the formation of a project from those that do not.

Almost all significant formal mathematical theories, concepts, methods, and results are built within axiomatic set theory, which represents one of the greatest achievements of modern mathematics. Therefore, we develop project theory within the framework of set theory, utilizing its language and constructions. In that sense, we'll clarify all project notions by expressing them in terms of set theory. For example, we may discuss functions with domains of projects or project families. Additionally, the concepts of finiteness or infiniteness in a project may arise, as well as different types of relations on sets of tasks.

First, we build the language of project theory, and express project properties using formulas defined within this language. Another key aspect of project theory is the construction of its axiomatic system. The system of axioms defines the notion of a project, as the foundational element of project theory. Each axiom is either existential or expresses a principle that we use when dealing with projects. The final component in developing the theory is its logic. The logic of project theory is set theory logic. Starting from projects, whose existence is guaranteed by the axioms, we construct new projects, thereby building a universe of projects and project constructions. For further insight, please see, for example, Jech (2003), which covers fundamental and advanced topics in set theory.

2. Language of Project Theory and Formulas

To formalize the axioms of Project Theory, we develop the theory within the framework of first-order predicate calculus. The language of Project Theory consists of the equality predicate $=_p$ and a binary predicate $\prec$, which represents the dependency relation among tasks.

The formulas of Project Theory are constructed from the atomic formulas $x =_p y$ and $x \prec y$, using the standard logical connectives:

$$\wedge,\ \vee,\ \neg,\ \rightarrow,\ \leftrightarrow$$

and quantifiers:

$$\forall x, \exists x.$$

In practice, we may utilize additional symbols, such as defined predicates, operations and constants, within our formulas. However, it is implicitly understood that any such formula can be rewritten in a form that involves only the non-logical symbols $=_p$ and $\prec$.

Regarding formulas containing free variables, we adopt the convention that all free variables in a formula $\varphi(u_1, \dots, u_n)$ are among the variables $u_1, \dots, u_n$ (with the understanding that some u_i may not be free, or may not even appear in φ). A formula without free variables is termed a sentence.

2.1. *Classes of Tasks*

In Project Theory, we introduce the concept of task classes within a project. This concept is particularly informative because working with classes often proves more manageable than directly dealing with the formulas in the language of Project Theory.

Every project can be viewed as a class of tasks. If $\mathcal{P}$ is a project, consider the formula $\phi(x, p)$ that characterizes the tasks belonging to that project. The assertion that a task x satisfies the formula $\phi(x, p)$ is denoted by $x \in \mathcal{P}$, where $\mathcal{P} = \{x : \phi(x, p)\}$. Thus, the project $\mathcal{P}$ is uniquely defined by the set of tasks it encompasses.

Remark 2.1. *The completion of a given project is contingent upon the successful completion of all tasks within that project.*

3. Axioms of Project Theory

3.1. *Axiom of Dependency (Dependency Schema for projects)*

The fundamental relationship in project management is the dependency relation of tasks. We will use the Dependency Schema instead of a single Dependency axiom, so that we can express the dependency relation that can be applied to any project, no matter how complex or simple its structure may be. A single Dependency Axiom would impose one universal rule for all projects, which might not be flexible enough to accommodate the diversity in task relationships. So, to perform certain task in a project means the completion of preceding tasks within the project. The successful execution of a project is contingent upon the completion of all its tasks. The dependency relation of tasks $\prec$, is a strict ordering on the class of all tasks within a project which is formulated in the *Dependency Schema for projects*:

$$(\forall p)(p \in \mathcal{P} \to p \nprec p) \tag{1}$$

$$(\forall p)(\forall q)(\forall r)(p \in \mathcal{P} \wedge q \in \mathcal{P} \wedge r \in \mathcal{P}) \to (p \prec q \wedge q \prec r \to p \prec r), \tag{2}$$

for any project $\mathcal{P}$.

3.2. *Axiom of Extensionality (Extensionality Schema for projects)*

Before formulating the Extensionality Schema for projects, we must define the equality of two projects.

Definition 3.1. *Let $\mathcal{X}$ and $\mathcal{Y}$ be two projects, such that there exists a project $\mathcal{P}$ in which they are tasks. $\mathcal{X}$ and $\mathcal{Y}$ are equal in the project $\mathcal{P}$, denoted by $\mathcal{X} =_p \mathcal{Y}$, if:*

$$(\forall z)(z \in \mathcal{P} \to (\mathcal{X} \prec z \leftrightarrow \mathcal{Y} \prec z)). \tag{3}$$

The equality of tasks in a project, as defined above, leads to the following *Extensionality Schema for projects*:

Let $\mathcal{X}$ and $\mathcal{Y}$ be two projects such that there exists a project $\mathcal{P}$ in which they are tasks. If projects $\mathcal{X}$ and $\mathcal{Y}$ have the same tasks, then $\mathcal{X} =_p \mathcal{Y}$:

$$(\forall u)(u \in \mathcal{X} \leftrightarrow u \in \mathcal{Y}) \rightarrow \mathcal{X} =_p \mathcal{Y}. \tag{4}$$

The converse, namely, if $\mathcal{X} =_p \mathcal{Y}$ then $u \in \mathcal{X} \leftrightarrow u \in \mathcal{Y}$, follows as an axiom of predicate calculus. Thus, we have:

$$\mathcal{X} =_p \mathcal{Y} \leftrightarrow (\forall u)(u \in \mathcal{X} \leftrightarrow u \in \mathcal{Y}). \tag{5}$$

This axiom expresses the fundamental idea that a project is determined by its tasks.

The definition of task equality in a project, along with the Extensionality Schema for projects, demonstrates that tasks x and y are in the relation $=_p$ if they occupy equivalent positions within the set of all interdependent tasks that constitute the given project. A task in a project is a planned activity that must be completed for the project to be realized. Each task in a project can be performed in multiple ways to ensure the project's continuation. Thus, tasks related by $=_p$ can be viewed as different ways of performing the same task, representing different options for a single task.

According to the above, it is evident that for each project we have corresponding axioms and definitions for the equality of tasks within that project. Therefore, we refer to these as Schemas of Axioms, as we mentioned earlier.

Definition 3.2. *Let $x \in \mathcal{P}$ be a task in a project. The **intentionality** of x in $\mathcal{P}$ is the class:*

$$int_{\mathcal{P}}(x) = \{y \mid x \prec y\} = (x, \rightarrow),$$

*and the **extensionality** of x in $\mathcal{P}$ is the class:*

$$ext_{\mathcal{P}}(x) = \{y \mid y \prec x\} = (\leftarrow, x).$$

Thus, the essence of the equality of tasks in a given project is that two tasks in a project are equal if the equality of their intentions is equivalent to the equality of their extensions, i.e., $x =_p y$ if and only if:

$$ext_{\mathcal{P}}(x) = ext_{\mathcal{P}}(y) \leftrightarrow int_{\mathcal{P}}(x) = int_{\mathcal{P}}(y).$$

Definition 3.3. *A formula $\phi(x, p)$ in the language of Project Theory is **consistent** with the relation $\prec$ in a given project $\mathcal{P}$ if the following holds:*

$$x \in \mathcal{P} \wedge \phi(x, p) \rightarrow (\forall y)(y \prec x \rightarrow \phi(y, p)). \tag{6}$$

3.3. Axiom of Separation (Separation Schema for Projects)

Let $\phi(x, p)$ be a formula consistent with the relation $\prec$ in some project $\mathcal{P}$. Then, for project $\mathcal{P}$ there exists a project $\mathcal{Y} = \{u \in \mathcal{P} \mid \phi(u, p)\}$ such that:

$$(\forall u)\ (u \in \mathcal{Y} \leftrightarrow (u \in \mathcal{P} \wedge \phi(u, p))). \tag{7}$$

For each formula $\phi(x, p)$, equation 7 is an Axiom of Separation. The project $\mathcal{Y}$ in 7 is unique by the Axiom of Extensionality for projects.

Remark 3.4. *We can give the Separation Axiom for projects the following form. Consider the class of projects:*

$$C = \{u \mid \phi(u, p_1, \dots, p_n)\},$$

where $\phi(u, p_1, \dots, p_n)$ is a formula consistent with the relation $\prec$ in a given project $\mathcal{P}$. Then, by 7, it holds that:

$$\exists \mathcal{Y} \, (\mathcal{P} \cap C = \mathcal{Y}). \tag{8}$$

In particular, if the class C is a subclass of the project $\mathcal{P}$, than the project $\mathcal{Y}$ is called a subproject of the project $\mathcal{P}$ and is denoted $\mathcal{Y} \subset_p \mathcal{P}$. The Separation Schema for projects allows us to select subprojects from a given project, ensuring that project properties remain consistent with the structure of $\mathcal{P}$.

If $\phi(x, p)$ is a project property of $\mathcal{X}$ consistent with the relation $\prec$ on $\mathcal{Y}$, and the formula $\psi(x, q)$ is a project property of $\mathcal{Y}$ consistent with the relation $\prec$ on $\mathcal{X}$, then one consequence of the Separation Schema for projects is that the class $\mathcal{X} \cap \mathcal{Y}$ is a subproject of both $\mathcal{X}$ and $\mathcal{Y}$. In this case, the project $\mathcal{X} \cap \mathcal{Y}$ is the intersection of the projects $\mathcal{X}$ and $\mathcal{Y}$, and we can define the operation:

$$\mathcal{X} \cap_p \mathcal{Y} = \{u \mid u \in \mathcal{X} \wedge u \in \mathcal{Y}\} = \mathcal{X} \cap \mathcal{Y}.$$

3.4. *Axiom of Project Existence*

Let $\mathcal{X}$ be a given project. The formula $\phi(x, p) : \neg(x \prec p)$ is not consistent with the relation $\prec$ with respect to the project $\mathcal{X}$. Indeed, consider the implication:

$$x \in \mathcal{X} \wedge \neg(x \prec p) \rightarrow \forall y(y \prec x \rightarrow \neg(y \prec p)).$$

This does not hold in general. Suppose that $x \in \mathcal{X}$ and $p \in \mathcal{X}$, such that p is incomparable with x. It suffices to take any task y such that $y \prec x$ and $y \prec p$, and the implication will not be satisfied. Therefore, the difference between two projects in general is not a project.

It holds that $\emptyset = \{u \mid u \neq_p u\}$. Clearly, the formula $\phi(x) : \neg(x =_p x)$ is consistent with the relation $\prec$ on any project $\mathcal{X}$. Thus, it follows that the empty class is a project (an empty project), under the assumption that at least one project $\mathcal{X}$ exists. Therefore, we formulate the following axiom: There exists at least one non-empty project:

$$\exists \mathcal{X} \, \exists x \, (x \in \mathcal{X} \wedge \neg(x =_p \emptyset)). \tag{9}$$

Based on the Axiom of Project Existence, we conclude that the empty project is a subproject of any project. Indeed, if the defining property of project $\mathcal{X}$ is the formula $\psi(x, p)$, then it is consistent with the relation $\prec$ on the empty project $\emptyset$ because:

$$(u \in \emptyset \wedge \psi(x, p)) \rightarrow \forall v(v \prec u \rightarrow \psi(y, p)).$$

Thus, $\emptyset \cap_p \mathcal{X} = \emptyset$ for every project $\mathcal{X}$.

Definition 3.5. *Two projects $\mathcal{X}$ and $\mathcal{Y}$ are called disjoint projects if $\mathcal{X} \cap_p \mathcal{Y} = \emptyset$.*

Theorem 3.6. *Let $\mathcal{P}$ be a given project and x a task within project $\mathcal{P}$, thus $x \in \mathcal{P}$. The class:*

$$\mathrm{ext}_{\mathcal{P}}(x) = \{z \mid z \in \mathcal{P} \wedge z \prec x\} = (\leftarrow, x)$$

is a subproject of project $\mathcal{P}$. It also holds that $\mathrm{ext}_{\mathcal{P}}(x) \cap_p \mathcal{P} = \mathrm{ext}_{\mathcal{P}}(x)$.

Proof. The class $\mathrm{ext}_{\mathcal{P}}(x)$ is defined by the property $\phi(u, \mathcal{P}, x) : u \in \mathcal{P} \wedge u \prec x$, which is consistent with the relation $\prec$ in the project $\mathcal{P}$. Indeed, it holds:

$$(u \in \mathcal{P} \wedge u \prec x) \rightarrow \forall v(v \prec u \rightarrow v \in \mathcal{P} \wedge v \prec x).$$

Therefore, the class $\mathrm{ext}_{\mathcal{P}}(x)$ is a subproject of project $\mathcal{P}$. The defining property $\psi(u, \mathcal{P}) : u \in \mathcal{P}$ of project $\mathcal{P}$ is consistent with the relation $\prec$ on the project $\mathrm{ext}_{\mathcal{P}}(x)$ because it holds:

$$(u \in \mathrm{ext}_{\mathcal{P}}(x) \wedge u \in \mathcal{P}) \rightarrow \forall v(v \prec u \rightarrow v \in \mathcal{P}).$$

Thus, $\mathrm{ext}_{\mathcal{P}}(x) \cap_p \mathcal{P} = \mathrm{ext}_{\mathcal{P}}(x)$. □

Remark 3.7. *As classes in Set Theory are replacements for formulas in the language of Set Theory, just as classes in Project Theory are replacements for formulas in the language of Project Theory, it will be clear from the context what classes are being referred to, so we will not introduce special notations for them.*

Theorem 3.8. *If $\mathcal{P}$ is a project and $x \in \mathcal{P}$ is a task, then the class $\mathrm{int}_{\mathcal{P}}(x) = \{z \in \mathcal{P} \mid x \prec z\}$, the intension of x in $\mathcal{P}$, is not a project in the general case.*

Proof. The task class $\mathrm{int}_{\mathcal{P}}(x)$ is defined by the formula $\phi(u, \mathcal{P}, x) : u \in \mathcal{P} \wedge x \prec u$. The class $\mathrm{int}_{\mathcal{P}}(x)$ is a subclass of the project $\mathcal{P}$. The formula $\phi(u, \mathcal{P}, x)$ is not consistent with the relation $\prec$ on the project $\mathcal{P}$. The implication

$$(u \in \mathcal{P} \wedge x \prec u) \to \forall v(v \prec u \to (v \in \mathcal{P} \wedge x \prec v))$$

does not hold in general. Indeed, let u be a task that is incomparable with x. It is enough to take any task v such that $v \prec u$ and v is incomparable with x, which makes the implication false. Hence, $\mathrm{int}_{\mathcal{P}}(x)$ is not a project. □

It is evident that the following two theorems hold.

Theorem 3.9. *If $x, y \in P$ and $P' \subseteq P$ is a subproject such that $x, y \in P'$, then $x =_p y$ in P implies $x =_p y$ in P'.*

Theorem 3.10. *Let $x, y \in \mathcal{P}$ be tasks in the project. If $x =_p y$, then x is incomparable with y.*

Theorem 3.11. *For any set of tasks $\mathcal{X}$, there exists a project $\mathcal{Y} \subseteq \mathcal{X}$ such that:*

$$\forall u\, (u \in \mathcal{Y} \leftrightarrow \exists z\, (z \in \mathcal{X} \wedge u \prec z)).$$

Proof. Let us define the project $\mathcal{Y}$ as the set of all tasks u for which there exists a task z in $\mathcal{X}$ such that $u \prec z$. Formally, we define $\mathcal{Y}$ as:

$$\mathcal{Y} = \{u \mid (\exists z)(z \in \mathcal{X} \wedge u \prec z)\}.$$

We need to show that $\mathcal{Y} \subseteq \mathcal{X}$ and that $\mathcal{Y}$ satisfies the equivalence:

$$u \in \mathcal{Y} \leftrightarrow \exists z(z \in \mathcal{X} \wedge u \prec z).$$

First, we show that $\mathcal{Y} \subseteq \mathcal{X}$. Suppose $u \in \mathcal{Y}$. By the definition of $\mathcal{Y}$, there exists a task $z \in \mathcal{X}$ such that $u \prec z$. By the properties of the dependency relation $\prec$, if $u \prec z$ and $z \in \mathcal{X}$, then u is also a task in the project $\mathcal{X}$. Hence, $u \in \mathcal{X}$. Therefore, $\mathcal{Y} \subseteq \mathcal{X}$. Now we prove the equivalence. Thus, we show:

$$u \in \mathcal{Y} \leftrightarrow \exists z(z \in \mathcal{X} \wedge u \prec z).$$

Assume $u \in \mathcal{Y}$. By the definition of $\mathcal{Y}$, there exists a task $z \in \mathcal{X}$ such that $u \prec z$. This directly satisfies the right side of the equivalence.

Assume there exists a task $z \in \mathcal{X}$ such that $u \prec z$. Then, by the definition of $\mathcal{Y}$, $u \in \mathcal{Y}$. Since both directions are proven, we have shown that the equivalence holds. Therefore, the theorem is proven. □

Theorem 3.12. *A two-member set of tasks in a project is not a subproject.*

Proof. Let $\mathcal{P}$ be a project in which there exist tasks a and b that are incomparable with respect to the relation $\prec$. The Pairing Axiom in Set Theory is formulated as:

$$(\forall a)(\forall b)(\exists c)(\forall z)(z \in c \leftrightarrow z = a \vee z = b). \tag{10}$$

The Axiom of a two-member project, if its formulation were meaningful, would be stated in the language $\mathcal{L}_p$ as:

$$(\forall a)(\forall b)(\exists c)(\forall z)(z \in c \leftrightarrow z =_p a \vee z =_p b). \tag{11}$$

For a two-member class of tasks to be a project, a subproject of the project $\mathcal{P}$, the defining formula of that class must be consistent with the relation $\prec$ with respect to the project $\mathcal{P}$.

Consider the formula $\phi(z, a, b) : z =_p a \vee z =_p b$. We will prove that it is not consistent with the relation $\prec$ with respect to the given project $\mathcal{P}$; that is, we will prove that the following does not hold:

$$u \in \mathcal{P} \wedge (u =_p a \vee u =_p b) \to \forall v(v \prec u \to v =_p a \vee v =_p b). \tag{12}$$

Assuming $u =_p a \vee u =_p b$, it may be, for instance, that $u =_p a$. If there is a task $v \prec u$, v cannot be equal to a as a project, which is the conclusion of the implication because $v \prec a$. Nor can it be $v =_p b$ because then $b \prec a$, but they are incomparable tasks. Thus, the formula $\phi(u, a, b) : u =_p a \vee u =_p b$ is not consistent with the relation $\prec$ with respect to the given project $\mathcal{P}$ for incomparable tasks a and b, and hence the class c is not a subproject of the project $\mathcal{P}$, and therefore not a project. □

As is well known, in Set Theory, two set constructions are defined: the intersection of a given family of sets and the union of a given family of sets. Here, we will provide analogous definitions for projects.

Definition 3.13. *Let $\{\mathcal{P}_i\}_{i\in I}$ be a family of projects. The intersection of the family of projects is defined as:*

$$\cap_{i\in I}\mathcal{P}_i = \{x \mid (\forall i)(i \in I \to x \in \mathcal{P}_i)\}.$$

Definition 3.14. *Let $\{\mathcal{P}_i\}_{i\in I}$ be a family of projects. The union of the family of projects is defined as:*

$$\cup_{i\in I}\mathcal{P}_i = \{x \mid (\exists i)(i \in I \wedge x \in \mathcal{P}_i)\}.$$

Theorem 3.15. *Let $\{P_i\}_{i\in I}$ be a family of projects. The intersection of this family $\cap_{i\in I}P_i$ is a project and a subproject of every P_i for $i \in I$.*

Proof. The class of tasks $\cap_{i\in I}\mathcal{P}_i$ is a subclass of the project $\mathcal{P}_i$ for every index $i \in I$, including the index $i = k$. To prove that the class $\cap_{i\in I}\mathcal{P}_i$ is a project, it is sufficient to prove that the defining property of this class $\phi(x, \mathcal{P}_i) : (\forall i)(i \in I \to x \in \mathcal{P}_i)\}$ is consistent with the relation $\prec$ with respect to the project $\mathcal{P}_k$, that is, it holds that:

$$(u \in \mathcal{P}_k \wedge (\forall i)(i \in I \to u \in \mathcal{P}_i)) \to \forall v(v \prec u \to (\forall i)(i \in I \to v \in \mathcal{P}_i)).$$

Indeed, if a task $u \in \mathcal{P}_i$ for every index $i \in I$, then every task $v \prec u$ will also be in $\mathcal{P}_i$ due to the transitivity of the relation $\prec$.

Thus, the class of projects $\cap_{i\in I}\mathcal{P}_i$ is a class whose defining property is consistent with the relation $\prec$ with respect to the project $\mathcal{P}_k$, and hence it is a subclass of the project $\mathcal{P}_k$. According to the Separation Schema for projects, it is a project, and a subproject of the project $\mathcal{P}_i$ for every index $i \in I$. □

Theorem 3.16. *Let $\{\mathcal{P}_i\}_{i\in I}$ be a family of projects. If all members of the family $\{\mathcal{P}_i\}_{i\in I}$ are subprojects of a given project $\mathcal{P}$, then $\cup_{i\in I}\mathcal{P}_i$ is also a subproject of that project.*

Proof. Assume now that all members of the family $\{\mathcal{P}_i\}_{i\in I}$ are subprojects of a given project $\mathcal{P}$, and we will prove that $\cup_{i\in I}\mathcal{P}_i$ is a project. Indeed, the class of tasks $\cup_{i\in I}\mathcal{P}_i$ is a subclass of the project $\mathcal{P}$. It is necessary to prove that the defining property of this subclass is consistent with the relation $\prec$ with respect to the project $\mathcal{P}$. This consistency is given by the following implication:

$$(u \in \mathcal{P} \wedge (\exists i)(i \in I \wedge u \in \mathcal{P}_i)) \to \forall v(v \prec u \to (\exists j)(j \in I \wedge v \in \mathcal{P}_i)).$$

Assume that the task $(u \in \mathcal{P}$ is in some subproject $\mathcal{P}_k$ for an index $k \in I$. If $v \prec u$, then it must also hold that $v \in \mathcal{P}_k$ for $j = k$, thereby proving the consistency. Therefore, the subclass $\cup_{i\in I}\mathcal{P}_i$ of the project $\mathcal{P}$ is a subproject of that project. □

4. Determination of Upwards Well-Founded Dependency

For every task in a project, there must exist tasks that directly precede it, whose completion is a prerequisite for the task's completion. The realization of any task cannot depend on an infinite number of its immediate predecessors, as such a situation would render the task unachievable. Ensuring this is crucial, we will address this issue in this section.

Definition 4.1. *An antichain is a subset of a partially ordered set such that any two elements in the subset are incomparable. A maximum antichain is one that has a cardinality at least as large as any other antichain. The width of a partially ordered set is the cardinality of a maximum antichain.*

Definition 4.2. *A binary relation $\prec$ on a project $\mathcal{P}$ is upwards well-founded or Noetherian if every nonempty subset $\mathcal{X} \subset \mathcal{P}$ has a $\prec$-maximal task, that is, an element $a \in \mathcal{X}$ such that there is no $x \in \mathcal{X}$ with $a \prec x$. In this case, we also say that every subproject of the project $\mathcal{P}$ is upwards well-founded.*

4.1. *Schema of Axioms of Upwards Well-Founded Relation $\prec$.*

Let $\mathcal{P}$ be a project. For any subproject $\mathcal{X} \subset_p \mathcal{P}$, the relation $\prec$ is upwards well-founded or Noetherian on $\mathcal{P}$:

$$(\forall \mathcal{X})((\mathcal{X} \neq \emptyset \land \mathcal{X} \subset_p \mathcal{P}) \rightarrow (\exists a)(a \in \mathcal{X} \land (\forall x)(x \in \mathcal{X} \rightarrow \neg(a \prec x)))) \tag{13}$$

For each project $\mathcal{P}$, formula 13 is an Axiom of the upwards well-founded relation $\prec$.

If x is any task in the project $\mathcal{P}$, then $(\leftarrow, x)$ is a subproject of the project $\mathcal{P}$. If the Axiom of Upwards Well-Founded Relation $\prec$ holds, then there exists at least one maximal task a_x in the project $(\leftarrow, x)$. Each of these maximal tasks will be called the *predecessor* of the task x. To complete the task $x \in \mathcal{P}$, it is necessary to complete all its predecessors, which is meaningful only if there are finitely many predecessors. It is therefore natural to formulate the following axiom.

Definition 4.3. *Let $\mathcal{P}$ be a project consisting of a set of tasks and let $\prec$ be a binary relation on the tasks in $\mathcal{P}$. The **transitive closure** of $\mathcal{P}$ under the relation $\prec$, denoted by $\mathcal{P}^+$, is defined as the smallest relation $\prec^+$ on $\mathcal{P}$ such that:*

1. *For all tasks $a, b \in \mathcal{P}$, if $a \prec b$, then $a \prec^+ b$.*
2. *For all tasks $a, b, c \in \mathcal{P}$, if $a \prec^+ b$ and $b \prec^+ c$, then $a \prec^+ c$.*

In other words, $\mathcal{P}^+$ contains all pairs of tasks (a, c) such that there exists a sequence of tasks $a = t_1, t_2, \ldots, t_n = c$ in $\mathcal{P}$ with $t_i \prec t_{i+1}$ for all $1 \leq i < n$.

Theorem 4.4. *Let $(\mathcal{P}, \prec)$ be a project with an upwards well-founded relation $\prec$. The transitive closure $\mathcal{P}^+$ of $\mathcal{P}$ under $\prec$ is well-founded if and only if $\mathcal{P}$ is well-founded.*

Proof. Assume $(\mathcal{P}, \prec)$ is well-founded. If $\mathcal{P}^+$ were not well-founded, there would be an infinite descending chain in $\mathcal{P}^+$. This chain can be decomposed into a finite sequence of $\prec$-relations, contradicting the assumption. Thus, $\mathcal{P}^+$ must be well-founded.

Conversely, assume $\mathcal{P}^+$ is well-founded. If $\mathcal{P}$ were not well-founded, there would be an infinite descending chain in $\mathcal{P}$, which would also be in $\mathcal{P}^+$, contradicting the assumption. Thus, $\mathcal{P}$ must be well-founded. □

Definition 4.5. *A model $\mathcal{M}$ is a pair $(U, \mathcal{R})$ where:*

- *U is a non-empty set, called the domain of $\mathcal{M}$, containing all elements over which the project $\mathcal{P}$ is interpreted.*
- *$\mathcal{R}$ is a set of relations on U that define the properties and constraints between elements in U. These relations can include:*

– *Order relations, such as a partial order $\prec_{\mathcal{M}}$ on U.*
– *Dependency relations, such as $D \subseteq U \times U$, indicating dependencies between elements.*

Definition 4.6. *Let $\mathcal{P} = (X, \prec)$ be a project, where X is a set of tasks and $\prec$ is a transitive, well-founded ordering relation on X. Let $\kappa : X \to \mathcal{M}$ be a function that maps each task $x \in X$ to a set $\kappa(x)$ in some model $\mathcal{M}$.*

We say that κ is transitive expansion of $\mathcal{P}$ if for all $x, y \in X$, the following condition holds:

$$x \prec y \quad \text{in } \mathcal{P} \quad \text{implies} \quad \kappa(x) \subseteq \kappa(y) \quad \text{in } \mathcal{M}.$$

In other words, if x precedes y in the ordering $\prec$, then every element of $\kappa(x)$ is also an element of $\kappa(y)$.

Theorem 4.7. *(Transitivity Preservation): If $\mathcal{P}$ is a project and κ is a transitive expansion of $\mathcal{P}$, then κ preserves the ordering relation:*

$$x \prec y \text{ in } \mathcal{P} \text{ implies } \kappa(x) \prec \kappa(y) \text{ in } \mathcal{M}.$$

Proof. *By the definition of a transitive expansion κ, for any $x \prec y$ in $\mathcal{P}$, $\kappa(x)$ is a subset of $\kappa(y)$. Specifically:*

- *If $x \prec y$, then by the definition of κ, $\kappa(x)$ will be included in $\kappa(y)$, preserving the ordering relation.*
- *For any element $z \in \kappa(x)$, z must be in $\kappa(y)$ since κ preserves the hierarchy of $\mathcal{P}$ in $\mathcal{M}$.*

Thus, κ preserves the ordering relation between tasks in $\mathcal{P}$ and $\mathcal{M}$. □

Example 4.8. *Let $\mathcal{P} = (X, \prec)$ be a project where:*

- *$X = \{A, B, C\}$ is the set of tasks.*
- *The ordering relation $\prec$ is defined as $A \prec B \prec C$, meaning task A must be completed before task B, and task B must be completed before task C.*

Now, let $\mathcal{M} = (U, \mathcal{R})$ be a model defined as follows:

- *The domain U is the set of resources and events:*

$$U = \{r_1, r_2, r_3, e_1, e_2, e_3\}$$

  where r_i represents a resource and e_i represents an event.

- *The set of relations $\mathcal{R}$ represents dependencies among the elements in U:*

$$\mathcal{R} = \{(r_1, r_2), (e_1, e_2)\}$$

  indicating, for example, that r_1 is a prerequisite for r_2 and e_1 is a prerequisite for e_2.

Define the mapping $\kappa : X \to \mathcal{M}$ as:

$$\kappa(A) = \{r_1, e_1\}$$
$$\kappa(B) = \{r_1, r_2, e_1, e_2\}$$
$$\kappa(C) = \{r_1, r_2, r_3, e_1, e_2, e_3\}$$

We can see that:

$$\kappa(A) \subseteq \kappa(B) \subseteq \kappa(C)$$

which preserves the transitivity of the original ordering $A \prec B \prec C$ in the project $\mathcal{P}$.

Thus, the transitivity preservation property is satisfied, as:

$$A \prec B \text{ in } \mathcal{P} \quad \Longrightarrow \quad \kappa(A) \subseteq \kappa(B) \text{ in } \mathcal{M},$$

and

$$B \prec C \text{ in } \mathcal{P} \quad \Longrightarrow \quad \kappa(B) \subseteq \kappa(C) \text{ in } \mathcal{M}.$$

4.2. Axiom of Anti-Chain Finiteness.

Let $\mathcal{P}$ be a project ordered by the relation $\prec$. Then every antichain with respect to the relation $\prec$ in the project $\mathcal{P}$ is finite.

The finiteness of all antichains in a given project $\mathcal{P}$ does not imply that the width of a partial order on a project is finite. The following statement is evident:

Proposition 4.9. *If the Axiom of Upwards Well-Founded Relation $\prec$ and the Axiom of Anti-Chain Finiteness hold, then the set of predecessors for every task in a given project is well-defined, and this set is finite.*

Definition 4.10. *A project $\mathcal{P}$ is said to satisfy the ascending chain condition (ACC) if every strictly ascending sequence $a_2 \prec a_3 \prec \cdots$ eventually terminates. Equivalently, given any sequence $a_1 \preceq a_2 \preceq a_3 \preceq \cdots$, there exists a positive integer n such that*

$$a_n =_p a_{n+1} =_p a_{n+2} =_p \cdots .$$

Theorem 4.11. *A project P satisfies the ascending chain condition (ACC) if and only if every subproject of P has a maximal task.*

Proof. $(i) \to (ii)$.

Assume (ii) is false. Then there is a nonempty subproject $\mathcal{X} \subseteq \mathcal{P}$ with no maximal element. Hence, there exists $x_1 \in \mathcal{X}$ such that x_1 is not a maximal element in the project $\mathcal{X}$. Therefore, in the project $\mathcal{X}$, there exists a task x_2 such that $x_1 \prec x_2$. Since x_2 is not maximal in $\mathcal{X}$, there exists $x_3 \in \mathcal{X}$ such that $x_1 \prec x_2 \prec x_3$, and continuing, the project $\mathcal{X}$ contains a strictly increasing sequence $x_1 \prec x_2 \prec x_3 \prec \cdots$, which contradicts (i).

$(ii) \to (i)$: Assume (ii) holds and $x_1 \prec x_2 \prec x_3 \prec \cdots$ is an increasing sequence in $\mathcal{P}$. Consider the class $\cup_{n\in\mathbb{N}}(\leftarrow, x_n)$, which is a subclass of the project $\mathcal{P}$, hence a subproject. The defining formula for the class $\cup_{n\in\mathbb{N}}(\leftarrow, x_n)$:

$$(u \in \mathcal{P} \wedge \exists n(n \in \mathbb{N} \wedge u \prec x_n) \to \forall v(v \prec u \to \exists m(m \in \mathbb{N} \wedge v \prec x_m))) \tag{14}$$

is consistent with the relation $\prec$ on the project $\mathcal{P}$. Suppose some task $u \in \mathcal{P}$ is in the project x_n for $n = n_0$. Then for every task $v \prec u$, there exists $m \in \mathbb{N}$ such that $v \prec x_m$, with $m = n_0$. Thus, the class $\cup_{n\in\mathbb{N}}(\leftarrow, x_n)$ is a subproject of $\mathcal{P}$ in which the entire sequence is contained. By (ii), there exists a maximal element a in the project $\cup_{n\in\mathbb{N}}(\leftarrow, x_n)$. Consequently, there exists an index $n(a)$ such that $a \in (\leftarrow, x_{n(a)})$, so all members of the sequence with indices $\geq n(a) - 1$ must be equal to $x_{n(a)-1}$, making the sequence stationary from that point. □

Theorem 4.12. *If $\mathcal{P}$ is a project where the Axiom of Upwards Well-Founded Relation $\prec$ and the Axiom of Anti-Chain Finiteness hold, then:*

$$\mathcal{P} = \cup_{x\in M}(\leftarrow, x],$$

where M is the set of all maximal elements in the project $\mathcal{P}$.

Proof. Let $a \in \mathcal{P}$. There are two possibilities: either $a \in M$ or not. If $a \in M$, then $a \in (\leftarrow, a] \subseteq \cup_{x\in M}(\leftarrow, x]$, so the theorem holds. Assume $a \notin M$. Then there exists $a_1 \in \mathcal{P}$ such that $a \prec a_1$. If $a_1 \in M$, then $a \in (\leftarrow, a_1] \subseteq \cup_{x\in M}(\leftarrow, x]$, so the theorem holds. Otherwise, we continue constructing a strictly increasing sequence of tasks in the project $\mathcal{P}$. According to Theorem 3.18, every strictly increasing sequence eventually terminates. Therefore, there exists an index $n(a)$ such that $a_n = a_{n(a)}$ for all indices $n > n(a)$, so $a_{n(a)} \in M$, and $a \in (\leftarrow, a_{n(a)}] \subseteq \cup_{x\in M}(\leftarrow, x]$, which completes the proof. □

4.3. Project Revision

The separation scheme for projects allows us to extract subprojects from a given project, thus generating new projects from the original one. We will show that projects can also be constructed in another way, starting from a given project.

Definition 4.13. *Let $\mathcal{P}$ be a project, and let M be a subclass of its tasks that is saturated with respect to the relation $=_p$, meaning that for each task $x \in M$, the subclass M contains all tasks in $\mathcal{P}$ that are $=_p$ to x. Assume that to the class of tasks M of the project $\mathcal{P}$, we have associated a disjoint collection of projects $\{\mathcal{P}_x\}_{x\in M}$, where each $x \in M$ is associated with a project $\mathcal{P}_x$. Define the class $\mathcal{P}(x) = (\mathcal{P} \setminus \{x\}) \cup \mathcal{P}_x$. Then the class*

$$\mathcal{P}(M) = \bigcup_{x\in M} \mathcal{P}(x) = \bigcup_{x\in M} ((\mathcal{P} \setminus \{x\}) \cup \mathcal{P}_x) = (\mathcal{P} \setminus M) \cup \bigcup_{x\in M} \mathcal{P}_x$$

is called a revision *of the project $\mathcal{P}$ defined by the set of tasks M, provided that for each task $x \in M \subseteq \mathcal{P}$, the ordering $\prec$ from the project $\mathcal{P}$ is extended to $\mathcal{P}(x)$ by the ordering $\prec_x$, defined as follows:*

- *For $u, v \in \mathcal{P} \setminus \{x\}$, we set $u \prec_x v$ if $u \prec v$.*
- *For $u, v \in \mathcal{P}_x$, we set $u \prec_x v$ if $u \prec_{\mathcal{P}_x} v$.*
- *For $u \in \mathcal{P} \setminus \{x\}$ and $v \in \mathcal{P}_x$, we set $u \prec_x v$ if $u \prec x$ in $\mathcal{P}$, and $v \prec_x u$ if $x \prec u$ in $\mathcal{P}$.*

The class of tasks $\mathcal{P}(M)$ is partially ordered by the relation $\prec_M$, which is the extension of $\prec$ to $\mathcal{P}(M)$. In subsequent discussions, if there is no risk of confusion, we will denote the dependency relation of tasks simply by $\prec$, without specifying the class of tasks on which it is defined.

4.4. Axiom of Project Revision

For every project P and any subset M of its tasks, there exists at least one revision $P(M)$ where each task $x \in M$ is replaced by a corresponding subproject P_x.

The class $\mathcal{P}(M)$ is the revision of $\mathcal{P}$ defined by the set of tasks M, where each $x \in M$ is replaced by its subproject $\mathcal{P}_x$.

Remark 4.14. *A task $x \in \mathcal{P}$ may contribute to deficiencies in the project. In such cases, the task x can be analyzed and replaced with a subproject $\mathcal{P}_x$. This leads to the revised project $\mathcal{P}(x) = (\mathcal{P} \setminus \{x\}) \cup \mathcal{P}_x$, which is expected to address the issues associated with x. The decomposition of x into a subproject $\mathcal{P}_x$ is not unique, as different approaches may be taken to define $\mathcal{P}_x$.*

When multiple tasks $M \subseteq \mathcal{P}$ contribute to deficiencies, a similar revision can be applied to all tasks in M, resulting in the revised project $\mathcal{P}(M)$.

4.5. Regressive Hierarchy of Subprojects

In this section, we define a hierarchy of subprojects for a given project $\mathcal{P}$. According to Theorem 3.16 for any subproject $\mathcal{X}$ of $\mathcal{P}$, the union $\bigcup \mathcal{X}$ is also a subproject of $\mathcal{P}$. Additionally, by the Axiom of Upward Well-Founded Relation $\prec$, every subproject $\mathcal{X}$ has at least one maximal task, and this also holds for the projects $\mathcal{P}$ and $\bigcup \mathcal{X}$.

Definition 4.15. *Let $\mathcal{P}$ be a project. A sequence $\{\mathcal{F}_n\}_{n\in\mathbb{N}}$ of subprojects of $\mathcal{P}$ is called a* regressive hierarchy of subprojects *if it is defined as follows:*

$$\mathcal{F}_0 = \mathcal{P}, \quad \mathcal{F}_{n+1} = \bigcup \mathcal{F}_n.$$

The classes $\mathcal{F}_n$, for each index $n \in \mathbb{N}$, are projects and subprojects of $\mathcal{P}$ by Theorem 3.16. Furthermore, each project $\mathcal{F}_n$ is upwardly well-founded, and if $n < m$, then $\mathcal{F}_n \supseteq \mathcal{F}_m$.

Theorem 4.16. *Let $\mathcal{P}$ be a project and $n \in \mathbb{N}$. Let M_n denote the class of all maximal tasks in the project $\mathcal{F}_n$. Then the following properties hold:*

$$M_n = \mathcal{F}_n \setminus \mathcal{F}_{n+1}, \quad M_n \cap M_m = \emptyset \text{ for } m \neq n, \quad \mathcal{P} = \bigcup_{n\in\mathbb{N}} M_n.$$

Proof. To show that $M_n \subseteq \mathcal{F}_n \setminus \mathcal{F}_{n+1}$, we need to prove that $M_n \cap \mathcal{F}_{n+1} = \emptyset$. Suppose, for contradiction, that there exists a task x in this intersection. Since $\mathcal{F}_{n+1} = \bigcup \mathcal{F}_n$, it follows that $x \in \bigcup_{y\in\mathcal{F}_n}(\leftarrow, y)$. Thus, there exists $y \in \mathcal{F}_n$ such that $x \in (\leftarrow, y)$, i.e., $x \prec y$, which contradicts the assumption that x is a maximal task in $\mathcal{F}_n$. Therefore, $M_n \subseteq \mathcal{F}_n \setminus \mathcal{F}_{n+1}$.

If $z \in \mathcal{F}_n \setminus \mathcal{F}_{n+1}$, assume that z is not maximal in $\mathcal{F}_n$. Then there exists $u \in \mathcal{F}_n$ such that $z \prec u$. Consequently, $z \in (\leftarrow, u) \subseteq \mathcal{F}_{n+1}$, which contradicts the assumption that $z \in \mathcal{F}_n \setminus \mathcal{F}_{n+1}$. Hence, z must be maximal.

Next, for $n < m$, we have $\mathcal{F}_m \subseteq \mathcal{F}_n$. If $x \in M_n$, then $x \notin \mathcal{F}_{n+1}$, so $x \notin M_m$, since $M_m \subseteq \mathcal{F}_m$. Therefore, $M_n \cap M_m = \emptyset$ for $n < m$.

Finally, we show that $\{M_n\}_{n\in\mathbb{N}}$ forms a partition of $\mathcal{P}$. Suppose there exists a task $x \notin \bigcup_{n\in\mathbb{N}} M_n$. Then, for every n, there would exist a task $x_n \in \mathcal{F}_n$ such that $x \prec x_n$. This would form an infinite descending chain in $\mathcal{P}$, contradicting the assumption that $\mathcal{P}$ satisfies the ascending chain condition (ACC). Thus, $\bigcup_{n\in\mathbb{N}} M_n = \mathcal{P}$. □

Remark 4.17. *In the proof of the previous theorem, we showed that:*

$$\mathcal{F}_\omega = \bigcap_{n<\omega} \mathcal{F}_n = \emptyset,$$

where ω denotes the first limit ordinal.

4.6. Depth of a Task in a Project

Let $\mathcal{P}$ be a project, and let $\prec$ be an upwards well-founded relation on the set of tasks in $\mathcal{P}$. In this section, we define the concept of the *depth* of a task within a project.

Definition 4.18. *Let P be a project with an upwards well-founded relation $\prec$. The depth of a task $x \in P$, denoted depth(x), is the length of the longest strictly ascending chain of tasks ending at x.*

Theorem 4.19. *For each task $x \in P$, where P is a project with an upwards well-founded relation $\prec$, there exists a natural number depth(x) representing the depth of x in P.*

Proof. Since $\prec$ is upwards well-founded, any strictly ascending chain of tasks in $\mathcal{P}$ is finite. Therefore, for each task $x \in \mathcal{P}$, we can find a maximal length n for which there exists a chain

$$x_0 \prec x_1 \prec \cdots \prec x_n = x.$$

The number n is the depth of x, denoted depth(x). The chain must terminate because $\prec$ is well-founded, ensuring that n is finite. □

To further generalize the notion of depth, we can extend it to ordinal numbers.

Theorem 4.20. *Let $\prec$ be an upwards well-founded relation on a project $\mathcal{P}$. Then there exists a unique function $\delta : \mathcal{P} \to Ord$, where Ord denotes the class of ordinals, such that for all $x \in \mathcal{P}$,*

$$\delta(x) = \sup\{\delta(y) + 1 \mid y \prec x\},$$

where the supremum is taken over all $y \in \mathcal{P}$ such that $y \prec x$. The value $\delta(x)$ is called the ordinal depth *of x.*

Proof. We construct the function δ by transfinite recursion. For each $x \in \mathcal{P}$, define $\delta(x)$ as the least ordinal greater than the depth of any task $y \prec x$. Specifically, we set

$$\delta(x) = \sup\{\delta(y) + 1 \mid y \prec x\}.$$

Since $\prec$ is upwards well-founded, the process of defining $\delta(x)$ terminates at a finite ordinal or a limit ordinal. Uniqueness follows from the well-foundedness of $\prec$, which ensures that no infinite strictly ascending chain exists, and thus every task x is assigned a unique depth. □

Remark 4.21. *The function δ assigns an ordinal depth to each task in $\mathcal{P}$, providing a more refined measure of depth compared to the natural number depth depth(x). In the case where $\mathcal{P}$ has a finite depth structure, $\delta(x)$ coincides with the natural number depth(x). For projects with more complex dependency structures, the ordinal depth may take limit ordinal values.*

Theorem 4.22. *If P is a project with an upwards well-founded relation $\prec$, then every task $x \in P$ has a finite depth.*

Proof. Since $\prec$ is upwards well-founded, there are no infinite strictly ascending chains in $\mathcal{P}$. This implies that for any task $x \in \mathcal{P}$, the depth of x (the length of the longest strictly ascending chain ending at x) must be finite. If the depth were infinite, it would imply the existence of an infinite strictly ascending chain, which contradicts the well-foundedness of $\prec$. Therefore, the depth of each task is finite. □

Theorem 4.23. *Let P be a project consisting of tasks and relations. The task $x \in P$ is maximal if and only if:*

$$\neg(x \prec (\cup P))$$

Additionally, it holds that:

Proof. Assume x is maximal in P. By definition, this means there does not exist any $y \in \cup P$ such that $x \prec y$. Formally, this can be expressed as:

$$\forall y \in \cup P, \neg(x \prec y)$$

Thus, $x \prec (\cup P)$ does not hold, which directly leads to the negation:

$$\neg(x \prec (\cup P))$$

Now, assume $\neg(x \prec (\cup P))$, meaning that x is not less than any task in $\cup P$. Since $\cup P$ includes all tasks in P that are part of a relation, this implies:

$$\forall y \in \cup P, \neg(x \prec y)$$

Thus, x cannot be less than any other task in P, making x maximal. □

Definition 4.24. *Let $(\mathcal{P}, \prec)$ be a project and $(\mathcal{M}, \in)$ a class of sets. We define by induction a function $\kappa : \mathcal{P} \to \mathcal{M}$ as:*

$$\kappa(x) = \{\kappa(z) \mid x \prec z\},$$

for every $x \in \mathcal{P}$. The function κ is called a transitive expansion *of $\mathcal{P}$.*

Theorem 4.25. *(a) The range of κ is a transitive class.*
(b) For all $x, y \in P$, if $x \prec y$, then $\kappa(y) \in \kappa(x)$.

Proof. (a) Let $\mathrm{Im}(\kappa) \subseteq \mathcal{M}$ denote the range of the function κ. We need to show that it is a transitive class. For any $w \in \mathrm{Im}(\kappa)$, there exists $x \in \mathcal{P}$ such that

$$w = \kappa(x) = \{\kappa(z) \mid x \prec z\}.$$

For each z such that $x \prec z$, we have $\kappa(z) \in \mathrm{Im}(\kappa)$. Thus, $w \subseteq \mathrm{Im}(\kappa)$, proving that $\mathrm{Im}(\kappa)$ is a transitive class.

(b) This follows directly from the definition of κ. If $x \prec y$, by the definition of κ, we have $\kappa(y) \in \kappa(x)$.
□

Remark 4.26. *The transitive expansion κ is not necessarily injective. It is injective if the relation $\prec$ satisfies an additional condition called* intentionality.

Definition 4.27. *An upwards well-founded relation $\prec$ on a project $\mathcal{P}$ is called* intentional *if for any two distinct tasks $x, y \in \mathcal{P}$, we have:*

$$int_{\mathcal{P}}(x) \neq int_{\mathcal{P}}(y).$$

According to the definition of task equality in the project $\mathcal{P}$, if for any two tasks x and y we have:

$$x =_p y \implies x = y,$$

then an upwards well-founded relation $\prec$ on a class $\mathcal{M}$ is intentional.

Definition 4.28. *A class M is called intentional if the relation $\prec$ on M is intentional, i.e., for any distinct elements $X, Y \in M$, we have $X \cap M \neq Y \cap M$.*

In formalizing project theory, it is important to ensure that abstract project structures can be represented concretely within a mathematical framework. Mostowski's Collapsing Theorem establishes an isomorphism between a project and a transitive set, providing a rigorous foundation for project hierarchies and ensuring that these structures can be analyzed within standard set theory.

Theorem 4.29 (Mostowski's Collapsing Theorem for Projects). *(i) If $\prec$ is an upwards well-founded relation on a project P, then there exists a unique transitive set M and a unique isomorphism π between $(P, \prec)$ and $(M, \in)$.*
(ii) Every project P is isomorphic to a transitive set M, with a unique isomorphism π.
(iii) If $T \subseteq P$ is transitive, then $\pi(x) = x$ for all $x \in T$.

Proof. We first prove (i), from which (ii) follows when $\prec = \in$.
Since $\prec$ is upwards well-founded, define π by transfinite recursion as:

$$\pi(x) = \{\pi(z) \mid x \prec z\}, \quad \text{for all } x \in \mathcal{P}.$$

In the case $\prec = \in$, this simplifies to:

$$\pi(x) = \{\pi(z) \mid z \in x \text{ and } z \in \mathcal{P}\}.$$

Thus, $\pi(\mathcal{P}) = \mathcal{M}$ is transitive by construction.
Suppose $\pi(x) = \pi(y)$ for distinct $x, y \in \mathcal{P}$. Then:

$$\pi(x) = \{\pi(z) \mid x \prec z\} \quad \text{and} \quad \pi(y) = \{\pi(w) \mid y \prec w\}.$$

Since $x \neq y$, there exists z such that $x \prec z$ but $z \notin \{w \mid y \prec w\}$, contradicting $\pi(x) = \pi(y)$. Therefore, π is injective. Now, let $x \prec y$. Then by definition, $\pi(y) \in \pi(x)$. Conversely, if $\pi(y) \in \pi(x)$, then $x \prec y$ by injectivity of π. Hence, π is an isomorphism between $(\mathcal{P}, \prec)$ and $(\mathcal{M}, \in)$. If π_1 and π_2 are isomorphisms from $\mathcal{P}$ to transitive sets $\mathcal{M}_1$ and $\mathcal{M}_2$, then $\pi_2 \circ \pi_1^{-1}$ is an automorphism of $\mathcal{M}_1$, which must be the identity due to the well-foundedness of $\prec$. Therefore, $\pi_1 = \pi_2$, proving uniqueness. Finally, if $T \subseteq \mathcal{P}$ is transitive, then for all $x \in T$, we have $\pi(x) = \{\pi(z) \mid x \prec z\}$, and since T is transitive, $\pi(x) = x$ for all $x \in T$, which proves (iii). □

5. Hierarchy for Projects

5.1. Upwards Well-Founded Relation

Definition 5.1. *A filter on a nonempty project S is a collection F of subprojects of S such that:*

- $S \in F$,
- *If $X \in F$ and y is a subproject of S, then $X \cap_p y \in F$,*
- *If $X \subseteq_p y$ where $X \in F$ and $y \subseteq_p S$, then $y \in F$,*
- $S \neq_p \emptyset$ *and* $\emptyset \notin F$.

Example 5.2. *A* trivial filter *is given by $\mathcal{F} = \{\mathcal{S}\}$.*

Example 5.3. *A* principal filter *can be constructed as follows. Let $\mathcal{X}_0$ be a nonempty subproject of $\mathcal{S}$. The filter $\mathcal{F} = \{\mathcal{X} \subset_p \mathcal{S} \mid \mathcal{X} \supset_p \mathcal{X}_0\}$ is a principal filter.*

Example 5.4. *The* Fréchet filter *on a project $\mathcal{S}$ consists of the collection of subprojects of $\mathcal{S}$ whose complements are finite. Note that the Fréchet filter is not principal.*

A filter $\mathcal{F}$ on $\mathcal{S}$ is *maximal* if there is no filter $\mathcal{F}'$ on $\mathcal{S}$ such that $\mathcal{F} \subset_p \mathcal{F}'$.

A project is a temporary endeavor designed to produce a unique product, service, or result. Each phase of a given project, as a separate project, delivers specific products or results. The project goal is achieved through the process of project realization, and the realization of each task in the project represents an improvement in the results achieved up to that point. The project result is achieved through the process of project advancement. The process of project management unfolds over time, and the entire project is always situated within some time interval. All tasks in the project or sets of tasks have their place within the project, meaning that all activities in the project have their duration in time and space. At any moment within the interval in which the entire project is situated, the state of the project can be identified, representing the results of project activities up to that point. Thus, while a set of tasks is partially ordered by the relation $\prec$, the set of states is linearly ordered as it can be indexed by time intervals. Each state of the given project $\mathcal{P}$ can be identified with the result of the realization of all tasks, thus it can be viewed as a single task. Similarly, the complete project as a set of tasks can be identified with the result of the entire project, meaning the whole project can be viewed as a single task. The relation between tasks and the project is specified by the following Axiom of Hierarchy.

5.2. Axiom of Hierarchy for Projects.

$$(\forall x)(\exists \mathcal{P})(\exists z)(z \in_p \mathcal{P} \wedge z = x).$$

This axiom expresses the idea that every result, every achievement is material for new achievements. With the completion of each project, the possibility for creating and realizing new projects is established.

To advance the theoretical framework, we propose the following theorem:

Theorem 5.5. *Let $(P, \prec)$ be a project with an upwards well-founded relation $\prec$. For any task $x \in P$, there exists a minimal transitive closure $P_{min} \subseteq P$ such that:*

$$P_{min} = \{y \in P \mid x \prec y \text{ and for all } z \in P, z \prec y \Rightarrow z \in P_{min}\}.$$

Proof. We first observe that the relation $\prec$ is upwards well-founded. Therefore, by definition, $\mathcal{P}_{\min}$ must be a minimal transitive closure. To prove this, consider the set:

$$\mathcal{P}_{\min} = \{y \in \mathcal{P} \mid \exists x \in \mathcal{P} \text{ such that } x \prec y \text{ and for all } z \in \mathcal{P} \text{ if } z \prec y \text{ then } z \in \mathcal{P}_{\min}\}.$$

Since $\mathcal{P}_{\min}$ is defined as containing all tasks y where each y is related to some $x \in \mathcal{P}$ and retains transitivity, it follows that $\mathcal{P}_{\min}$ is indeed a minimal transitive closure. Any larger set would violate the minimality condition or fail to retain transitivity.

Thus, $\mathcal{P}_{\min}$ uniquely represents the minimal transitive closure of the project $\mathcal{P}$. □

The proposed theorem contributes to the ongoing study of project hierarchies by providing a formal method for determining minimal transitive closures within project structures. This extension will enhance our ability to analyze and manage complex project environments more effectively.

5.3. *Compatibility*

There are simple tasks, or activities, that are considered fundamental and cannot be decomposed further into simpler activities. In the context of a project, some tasks can be treated as separate subprojects, i.e., as strictly ordered sets of tasks, while others cannot. When a task within a project can be viewed as a separate subproject, the ordering of the task must be consistent with the ordering of the entire project. This consistency is formalized by the following axiom:

5.4. *Axiom of Compatibility of Projects.*

(a) If $x \prec y$ and $z \in_p y$, then $x \prec z$.

(b) If $x \prec y$ and $w \in_p x$, then $w \prec y$.

Theorem 5.6. *If P is a project and $\prec$ satisfies the Axiom of Compatibility, then for any task $x \in P$ that is also a project, the ordering of tasks within x is consistent with the ordering in P.*

Proof. The Axiom of Compatibility of Projects states:

(a) $(\forall x)(\forall y)(\forall z)(x \prec y \land z \in_p y \to x \prec z)$,

(b) $(\forall x)(\forall y)(\forall w)(x \prec y \land w \in_p x \to w \prec y)$.

This axiom ensures that if $x \prec y$ in $\mathcal{P}$ and w is a task in a subproject x, then $w \prec y$. Specifically:

- If $x \prec y$ and $w \in_p x$, then by (b), $w \prec y$.
- If $w \in_p y$, and $x \prec y$, then by (a), $x \prec w$.

Thus, the ordering of tasks within a subproject respects the overall project ordering, maintaining consistency between the subproject and the parent project. □

5.5. *Revision of Projects*

The process of constructing new projects from an existing project is a natural consideration. The following axiom, known as the Axiom of Separation of Projects, formalizes this concept.

5.6. *Regularity*

In a project, each task is realized within a specific time interval and has a defined position relative to other tasks, including its predecessors and successors. Consequently, we regard a project as a set of distinct tasks, and each task as a distinct activity. Thus, we assert that a project does not contain duplicate tasks. This leads to the following axiom:

5.7. Axiom of Regularity for Projects.

In any well-structured project, task dependencies must follow a logical order. If tasks were allowed to depend on each other in an infinite loop or if there were no minimal tasks to start execution, project management would become impossible. The Axiom of Regularity ensures that there are no such infinite regressions or cycles in the task hierarchy, making the project well-founded.

Every non-empty project has an $\in_p$-minimal task:

$$\forall S(S \neq \emptyset \to \exists x \in S(S \cap x = \emptyset)).$$

Thus, the Axiom of Regularity for Projects implies that projects with certain undesirable properties, such as infinite regressions or cycles, do not exist. This restriction is consistent with the other axioms.

We recall that the project T is said to be transitive if for every $x \in_p T$, it holds that $x \subset T$.

6. Examples and Practical Applications

In this section, we provide some examples and illustrations to clarify the theoretical concepts discussed in previous chapters.

Example 6.1 (Complex Project Structure). *Consider a project P with tasks A, B, C, D, and E, where the relations are defined as follows:*

$$A \prec B \prec C, \quad A \prec D \prec E.$$

The depth of C is 2*, as the longest chain leading to C is $A \prec B \prec C$. Similarly, the depth of E is* 2*, while the depths of B and D are both* 1*. This example demonstrates how projects can have independent chains with equal depths.*

Next example involves the Axiom of Compatibility which ensures that task hierarchies remain consistent across subprojects. To illustrate this, consider the following example of a hierarchical project where subprojects maintain internal structures:

Example 6.2 (Hierarchical Project with Subprojects). *Consider a project P consisting of tasks A, B, C, and D, where:*

$$A \prec B \prec C, \quad A \prec D \prec C.$$

Now, let task D be a subproject consisting of tasks D_1 and D_2, where $D_1 \prec D_2$. In this case, the depth of C remains 2*, while the depth of D_2 is* 1*. This example illustrates how a subproject can fit within the larger project hierarchy while maintaining its own internal structure.*

The Axiom of Regularity prevents cyclic dependencies in project hierarchies, ensuring that task structures are well-founded. This axiom plays a key role in managing software development projects, where tasks follow strict orderings without circular dependencies. Consider the following example that highlights this concept:

Example 6.3 (Transitive Expansion in Software Development). *Let P represent a software development project where tasks are:*

$$A \prec B \prec C,$$

with A representing database setup, B representing API development, and C representing frontend design. Define the transitive expansion κ such that:

$$\kappa(A) = \{r_1, e_1\}, \quad \kappa(B) = \{r_1, r_2, e_1, e_2\}, \quad \kappa(C) = \{r_1, r_2, r_3, e_1, e_2, e_3\}.$$

Here, the transitive expansion κ ensures that all dependencies between components are properly maintained and that no component is used without its prerequisites.

Theorem 5.2 ensures that for any task x within a project, there exists a minimal transitive closure that maintains all dependencies. This principle is crucial in software engineering, where task dependencies form the foundation of build systems. The following example demonstrates how a dependency graph can be structured using minimal transitive closures:

Example 6.4 (Dependency Graph in Build Systems). *Consider a build system where tasks represent different stages of software compilation. Let P be a project with tasks A, B, C, and D where:*

$$A \prec B \prec D, \quad A \prec C \prec D.$$

Here, A represents code preparation, B is module compilation, C is resource linking, and D is the final build. The depth of D is 2*, and tasks B and C can be executed in parallel, showing how parallel tasks contribute to the overall project structure while maintaining dependencies.*

Example 6.5 (Application in Operations Research - Workflow Optimization). *In operations research, projects can be viewed as workflows composed of tasks that need optimization. By analyzing the hierarchical structure of tasks and their dependencies, one can: optimize workflows to minimize completion time, efficiently allocate resources to improve overall productivity and enhance process efficiency by identifying and eliminating redundant or unnecessary tasks.*

This chapter illustrated practical illustrations and applications of the theoretical concepts discussed, demonstrating their relevance and utility across various fields. By incorporating these examples, we aim to enhance the reader's understanding and highlight the practical impact of the theoretical results.

7. Conclusion

In this paper, we have thoroughly investigated the structural and foundational aspects of projects within the context of upwards well-founded relations. Our work has led to several key results and definitions that contribute significantly to the understanding of project hierarchies and their formal properties.

We introduced the concept of *depth* for tasks within a project, defined through a unique function δ that maps each task to an ordinal number. This depth function provides a robust measure of the hierarchical level of each task, revealing the intricate layering and progression inherent in project structures.

Our exploration of transitive expansions resulted in Theorem 1, which establishes that for any upwards well-founded relation $\prec$ on a project $\mathcal{P}$, there exists a unique function δ that assigns depth to each task in $\mathcal{P}$. This result underscores the inherent structure within projects and offers a foundational tool for analyzing hierarchical relationships.

We discussed the Axiom of Hierarchy, asserting that every outcome or achievement within a project can serve as a basis for further projects, illustrating the dynamic and iterative nature of project development.

The Axiom of Compatibility was introduced to ensure that the partial ordering of tasks within a project is preserved when considering subprojects. This compatibility guarantees that the hierarchical structure remains consistent across different levels of the project.

The Axiom of Regularity was established to prevent cycles and infinite descending chains within projects, thereby enforcing a well-structured and non-redundant task organization, crucial for managing complex projects effectively.

Applying Mostowski's Collapsing Theorem, we demonstrated that every upwards well-founded project is isomorphic to a transitive set. This result provides a solid theoretical foundation for understanding project structures and aligns with practical needs by simplifying the conceptual model.

This work extended existing project management methodologies by introducing a formal axiomatic foundation for managing task dependencies and hierarchies. The project theory presented in this paper can be directly applied to task scheduling in build systems, where tasks must be executed in a specific order while maintaining dependencies.

We proved our theoretical results by providing concrete examples and applications. For instance, we illustrated the application of the depth function in project management by analyzing task hierarchies in software development and construction projects. The depth function helped in understanding the complex dependencies and progressions in these projects, leading to more effective planning and resource allocation.

We also explored the implications of the Axiom of Compatibility in computer science, where maintaining hierarchical consistency across different levels of system design can be critical for ensuring system integrity and functionality.

In operations research, the Axiom of Regularity proved valuable in structuring complex logistics and supply chain management tasks, ensuring that task organization remains clear and free of redundant cycles.

Based on our findings, several directions for future research are proposed: exploration of additional properties of project hierarchies, such as the impact of varying depth functions on project efficiency and effectiveness, study specific classes of projects (such as an agile software development or large-scale construction projects, in order to understand how the theoretical results apply in these context), and employment of the theoretical results to interdisciplinary fields, such as organizational behavior or strategic planning, in order to explore how hierarchical structures influence decision-making and project outcomes.

This work advances the theoretical framework for understanding and managing projects by formalizing key concepts and proving significant results. The findings offer valuable insights into project organization and hierarchy and highlight the practical applications in various fields. Future research could build on these insights to develop advanced methodologies and tools, contributing to more effective project management and organizational planning.

References

[1] I. Bagaria, *Axioms: The Pillars of Mathematical Reasoning Explored*, Journal of Applied and Computational Mathematics, **12** (2023), 1–5.

[2] British Standards Institute (BSI), *Project Management-Principles and Guidance for the Management of Projects*, British Standards Institute, (2019).

[3] J. Heagney, *Fundamentals of Project Management*, 5th ed., AMACOM., (2016).

[4] T. Jech, *Set Theory: The Third Millennium Edition, Revised and Expanded*, Springer Science and Business Media, (2003).

[5] H. Kerzner, *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*, 6th ed., New York: Wiley, (1998).

[6] W. Park, *On Abducing the Axioms of Mathematics*, by J. R. Shook and S. Paavola (Eds.), Abduction in Cognition and Action: Logical Reasoning, Scientific Inquiry, and Social Practice, Springer Verlag., (2021), 161–175.

[7] PMI (Project Management Institute), *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*, 5th ed., (2012).

[8] R. Turner, *Gower Handbook of Project Management*, 5th ed., Routledge, (2014).