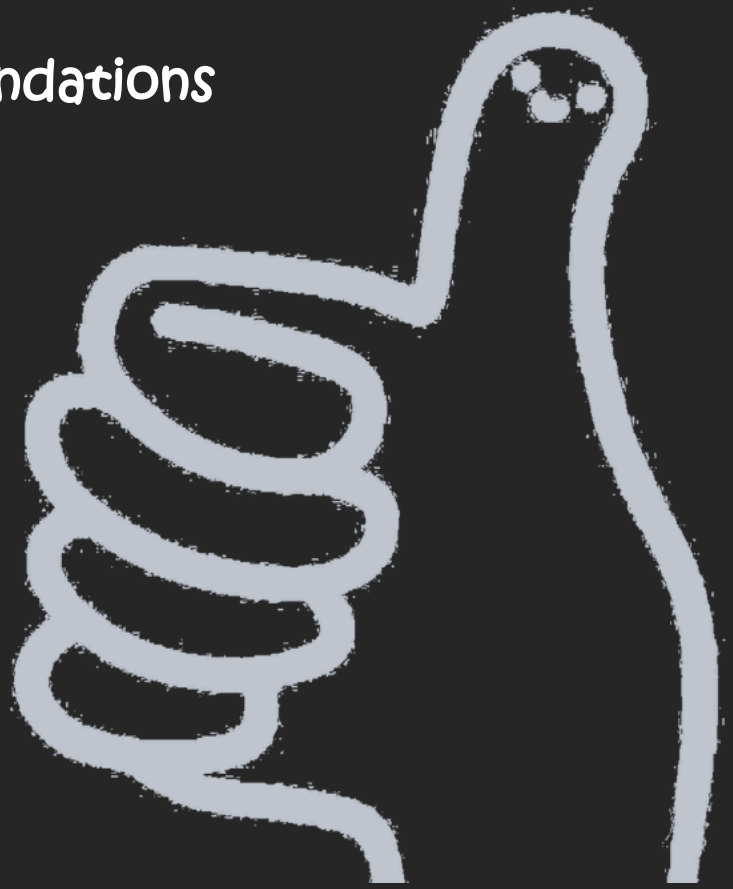
Formal Methods in Requirements Engineering
需求工程中的形式化方法

Fahad Sabah

CONTENTS



01

逻辑奠基与需求歧义消除

Logical foundation and demand ambiguity elimination

Why do we need a formal approach?
为何需要形式化方法？

01

Formal method definition
形式化方法定义

The formal method accurately expresses requirements through mathematical logic, replacing vague natural language and ensuring the uniqueness and verifiability of requirements.

形式化方法通过数学逻辑精确表达需求，替代模糊的自然语言，确保需求的唯一性和可验证性。

02

Disambiguation
消除歧义

In requirements engineering, formal methods can effectively eliminate ambiguity caused by natural language, provide clear guidance for subsequent development, and avoid errors caused by different understandings.

在需求工程中，形式化方法能有效消除自然语言带来的歧义，为后续开发提供清晰的指导，避免因理解不同而导致的错误。

03

Safety-critical applications
安全关键领域应用

Formal methods are widely used in safety-critical fields such as aviation, healthcare, and finance to ensure that systems behave as expected and avoid catastrophic consequences through rigorous verification and validation processes.

形式化方法广泛应用于航空、医疗、金融等安全关键领域，通过严格的验证和确认流程，确保系统行为符合预期，避免灾难性后果。

Levels of Specification
规范层级

Informal
非正式

Informal requirements specifications rely on natural language, which is prone to ambiguity.

非正式需求规格依赖自然语言，易产生歧义

Semi Formal
半形式化

Semi-formal such as UML use case diagrams, although structured, still contain ambiguity, and are suitable for preliminary requirements analysis.

半形式化如UML用例图，虽有结构但仍含模糊性，适用于初步需求分析。

Fully formalized
完全形式化

Fully formal methods such as Z language and Alloy achieve precise expression and automatic verification of requirements through strict syntax and semantic rules, and are suitable for the requirements specifications of complex systems.

完全形式化方法如Z语言和Alloy，通过严格的语法和语义规则，实现需求的精确表达和自动验证，适用于复杂系统的需求规格。

Example: UML vs Z vs Alloy

```
class User {
  - userId: String
  - name: String
}

User " * " -- " * " User : friends

-- "Users cannot be friends with themselves"
-- "Friendship is mutual"
```

```
[USER]

SocialNetwork
  users: ℙ USER
  friends: USER ↔ USER

  -- No self-friendship
  ∀ u: users • (u ↦ u) ∉ friends
  -- Friendship is symmetric
  friends = ~friends
```

```
sig User {
    friends: set User
}

fact FriendshipRules {
    -- No self-friendship
    no u: User | u in u.friends
    -- Friendship is mutual
    friends = ~friends
}
pred showFriends {
    #User >= 3
    some friends
}

run showFriends for 5
```

Propositional logic operators at a glance
命题逻辑运算符速览

The logical basis of the proposition
命题逻辑基础

Propositional logic is the basis of formal methods, which express the logical relationship of demand through logical operators such as ¬ (not), ∧ (and), ∨ (or), and → (implied).

命题逻辑是形式化方法的基础，通过逻辑运算符如¬（非）、∧（与）、∨（或）、→（蕴含）表达需求逻辑关系。

truth table
真值表

The truth table can clarify the truth or falsehood of each logical expression, ensure the correctness of logical relationships, and provide a basis for requirements verification.

通过真值表可以明确每个逻辑表达式的真假情况，确保逻辑关系的正确性，为需求验证提供基础。

Logical expression construction
逻辑表达式构建

To convert natural language requirements into logical expressions, it is necessary to accurately identify conditions and results, use logical operators reasonably, and ensure that the expressions are complete and unambiguous.

将自然语言需求转化为逻辑表达式，需准确识别条件与结果，合理使用逻辑运算符，确保表达式完整且无歧义。

Operator example
运算符示例

For example, the bank transaction rule 'Reject transaction if balance is less than zero' can be formalized as:
Balance < 0 → RejectTransaction.

例如，银行交易规则‘如果余额小于零，则拒绝交易’可形式化为：
Balance < 0 → RejectTransaction。

Step method of natural language to logic
自然语言转逻辑步法

A method for transforming natural language requirements into logical expressions: identifying entities, determining predicates, adding quantifiers, and constructing implications to ensure the systematization and accuracy of the requirements formalization process.

将自然语言需求转化为逻辑表达式的步法：识别实体、确定谓词、添加量词、构建蕴含，确保需求形式化过程的系统性和准确性。

Classroom practice: Requirements translation drill
课堂实战：需求翻译演练

1.**Identifying Entities:** The key objects, actors, or concepts in the requirement.
2.**Determining Predicates:** The properties of entities or the relationships between them.
3.**Adding Quantifiers:** Specifying the scope (e.g., ∀ for "all", ∃ for "there exists").
4.**Constructing Implications/Formulas:** Combining everything into a formal logical statement to capture the requirement's meaning.

Basic Logic Tables

Before working with the formalization exercises, let's review the fundamental logical operators and their truth tables. These are the building blocks for all logical expressions.

Basic Logical Operators

1. Negation (NOT) ¬

The opposite of the original statement.

P	¬P
T	F
F	T

Example: If P = "It is raining", then ¬P = "It is not raining"

2. Conjunction (AND) - ∧

True only when both statements are true.

P	Q	P ∧ Q
T	T	T
T	F	F
F	T	F
F	F	F

Example: "I will go to the park AND it is sunny" - I only go if both conditions are true.

3. Disjunction (OR) - ∨

True when at least one statement is true.

P	Q	P ∨ Q
T	T	T
T	F	T
F	T	T
F	F	F

Example: "I will have coffee OR tea" - I'm happy with either or both.

4. Implication (IF...THEN) - →

False only when the antecedent is true but the conclusion is false.

P	Q	P → Q
T	T	T
T	F	F
F	T	T
F	F	T

Example: "IF it rains, THEN I will bring an umbrella"

Rain + Umbrella = Logical (T)

Rain + No Umbrella = Illogical (F)

No Rain + Umbrella = Logical (T) - I might be cautious

No Rain + No Umbrella = Logical (T)

5. Equivalence (IF AND ONLY IF) - ⇔

True when both statements have the same truth value.

P	Q	P ⇔ Q
T	T	T
T	F	F
F	T	F
F	F	T

Example: "I will go to the party IF AND ONLY IF you go" - We both go or we both don't go.

Quantifiers

Universal Quantifier (∀ - "For all")
∀x P(x) means "For every x, P(x) is true"

Existential Quantifier (∃ - "There exists")
∃x P(x) means "There exists at least one x such that P(x) is true"

Combined Operations

De Morgan's Laws
$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$
"It's not true that both P and Q are true" = "Either P is false or Q is false"

$\neg(P \vee Q) \equiv \neg P \wedge \neg Q$
"It's not true that either P or Q is true" = "Both P is false and Q is false"

Let's verify De Morgan's first law with a truth table:

P	Q	$P \wedge Q$	$\neg(P \wedge Q)$	$\neg P$	$\neg Q$	$\neg P \vee \neg Q$
T	T	T	F	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	T	T	T	T

As we can see, the columns for $\neg(P \wedge Q)$ and $\neg P \vee \neg Q$ are identical, proving they are logically equivalent.

Exercise 1: Basic Access Control

Requirement: "A user must be logged in to post a message."

Solution:???

Let P = "User is logged in"

Let Q = "User posts a message"

Logical Expression: $Q \rightarrow P$

Q (Posts Message)	P (Logged In)	$Q \rightarrow P$ (Requirement)	Explanation
T	T	T	The user is logged in and posts a message. This satisfies the requirement.
T	F	F	The user is not logged in but posts a message. This violates the requirement.
F	T	T	The user is logged in but does not post a message. The requirement is not violated because no posting occurred.
F	F	T	The user is not logged in and does not post a message. The requirement is not triggered, so it is satisfied.

Exercise 2: Dual Condition

Requirement: "To approve a loan, the applicant must have good credit AND sufficient income."

Solution:

Let P = "Applicant has good credit"

Let Q = "Applicant has sufficient income"

Let R = "Loan is approved"

Logical Expression: $R \rightarrow (P \wedge Q)$

P	Q	R	$P \wedge Q$	$R \rightarrow (P \wedge Q)$
T	T	T	T	T
T	T	F	T	T
T	F	T	F	F (Violation: Approved without sufficient income)
T	F	F	F	T
F	T	T	F	F (Violation: Approved without good credit)
F	T	F	F	T
F	F	T	F	F (Violation: Approved with neither condition)
F	F	F	F	T

Exercise 3: Conditional Alerting

Natural Language Requirement: "If the CPU temperature exceeds 80°C, then an alert must be triggered."

Identify Entities:

This is a "state-of-affairs" requirement. The entities are the CPU temperature (a value) and the alert (an event/state). We can model these as propositions.

Determine Predicates:

HighTemp: The proposition that "CPU temperature > 80°C".

AlertTriggered: The proposition that "an alert is triggered".

Add Quantifiers:

This is a single, global condition. No quantifiers are needed in propositional logic.

Construct Implication:

The requirement is a direct conditional.

Logical Expression:

$$HighTemp \rightarrow AlertTriggered$$

In English: "If the temperature is high, then an alert is triggered."

HighTemp	AlertTriggered	HighTemp → AlertTriggered
T	T	T
T	F	F (Violation: High temp but no alert)
F	T	T
F	F	T

Exercise 4: User Authentication

Natural Language Requirement: "All users must be authenticated before they can access the secure server."

Identify Entities:

- user
- secure server
- access (an action/relationship)
- authenticated (a state/property)

Determine Predicates:

- User(u): u is a user.
- Server(s): s is the secure server.
- CanAccess(u, s): User u can access server s.
- IsAuthenticated(u): User u is authenticated.

Add Quantifiers:

"All users" → Universal quantifier (∀) over users.

We are talking about a specific server, so we can use a constant (e.g., server1) or existentially quantify it. Since it's a specific system component, a constant is better.

Construct Implication:

The requirement states that being able to access the server implies that the user must be authenticated. It does not say that being authenticated grants access (there could be other conditions).

Logical Expression:

$\forall u \, ((User(u) \land CanAccess(u, server1)) \rightarrow IsAuthenticated(u))$ Truth Table???

In English: "For every user, if the user is a user and can access the server, then that user must be authenticated."

Example 1: User Authentication

Expression: $\forall u$ ((User(u) $\wedge$ CanAccess(u, server1)) $\rightarrow$ IsAuthenticated(u))

Note: For truth tables, we consider specific instances since universal quantifiers apply to all instances.

User(u)	CanAccess(u, server1)	IsAuthenticated(u)	(User(u) $\wedge$ CanAccess(u, server1)) $\rightarrow$ IsAuthenticated(u)
T	T	T	T
T	T	F	F (Violation: Access without authentication)
T	F	T	T
T	F	F	T
F	T	T	T
F	T	F	T
F	F	T	T
F	F	F	T

Exercise 3: Resource Allocation Policy

Natural Language Requirement: "A process that holds a resource cannot request another resource."

Identify Entities:

- process
- resource

Determine Predicates:

- Process(p): p is a process.
- Resource(r): r is a resource.
- Holds(p, r): Process p holds resource r.
- Requests(p, r): Process p requests resource r.

Add Quantifiers:

We are talking about any process and any two (distinct) resources. We need to quantify over all processes and all pairs of resources.

Construct Implication:

The requirement is a constraint: If a process holds some resource r1, then it must not request any other resource r2. We need to ensure r1 and r2 are different.

Logical Expression:

$\forall p \, \forall r1 \, \forall r2 \, ((Process(p) \land Resource(r1) \land Resource(r2) \land (r1 \neq r2) \land Holds(p, r1)) \rightarrow \neg Requests(p, r2))$

In English: "For every process p and for every pair of distinct resources r1 and r2, if p holds r1, then p does not request r2."

Case	Holds(p1,r1)	Requests(p1,r2)	(Holds(p1,r1) → ¬Requests(p1,r2))
1	T	T	T → F = F
2	T	F	T → T = T
3	F	T	F → F = T
4	F	F	F → T = T

Process(p1) = True
Resource(r1) = True, Resource(r2) = True, Resource(r3) = True

Truth Table:

Case	Holds(p1,r1)	Requests(p1,r2)	(Holds(p1,r1) → ¬Requests(p1,r2))	Result	Notes
1	T	T	T → F = F	F	VIOLATION: Holds r1 AND requests r2
2	T	F	T → T = T	T	Holds r1 but doesn't request r2
3	F	T	F → F = T	T	Doesn't hold r1 (can request r2)
4	F	F	F → T = T	T	Doesn't hold r1 and doesn't request r2

Extended for multiple resources:

Holds(p1,r1)	Requests(p1,r2)	Requests(p1,r3)	Result r1→¬r2	Result r1→¬r3	Overall
T	T	T	F	F	F
T	T	F	F	T	F
T	F	T	T	F	F
T	F	F	T	T	T
F	T	T	T	T	T
F	T	F	T	T	T
F	F	T	T	T	T
F	F	F	T	T	T

Key Insight: The expression is FALSE only when a process holds one resource AND requests a different resource.

02

Logic-Based Specifications (Z Notation)

基于逻辑的规范（Z表示法）

The value of the Z Notation
Z符号的价值

Definition of Z language
Z语言的定义

Z is a formal language based on patterns and set theory, developed by the University of Oxford and became an ISO standard. It describes system requirements through precise mathematical notation, effectively eliminating ambiguity.

Z是一种基于模式和集合论的形式化语言，由牛津大学开发并成为ISO标准。它通过精确的数学符号来描述系统需求，有效消除歧义。

Z language application
Z语言的应用

The Z language is widely used in avionics and financial systems, and the system defect rate is significantly reduced through a formal approach. For example, after a flight control module uses Z language, the unit test cases are reduced by 30%.

Z语言广泛应用于航空电子和金融系统，通过形式化方法显著降低了系统缺陷率。例如，某飞控模块使用Z语言后，单元测试用例减少了30 。

The necessity of the Z language
Z语言的必要性

In the requirements stage, Z language can provide non-ambiguous benchmarks and provide accurate specifications for subsequent design, coding and testing, thereby reducing the cost of debugging and regression in the later stage.

在需求阶段，Z语言能够提供无二义性的基准，为后续设计、编码与测试提供精确的规格说明，从而减少后期的调试和回归成本。

Z核心概念速览

01

集合、关系与函数

The Z language uses three building blocks: **sets, relationships, and functions** to define system states. Collections are used to describe data structures, and relationships and functions are used to define mapping relationships between data.

Z语言使用集合、关系和函数三大构件来定义系统状态。集合用于描述数据结构，关系和函数用于定义数据之间的映射关系。

02

状态变化符号

The Z language introduces the Δ and Ξ symbols, which represent variable and unchanging state patterns, respectively. The declaration part of the schema box defines the variable signature, and the predicate part imposes business rules.

Z语言引入Δ和Ξ符号，分别表示可变和不变的状态模式。模式框的声明部定义变量签名，谓词部施加业务规则。

Z Notation: Sets, Relations, and Functions - Simple Examples
1. Sets in Z Notation

Basic Set Declaration

-- Declare basic types (sets of all possible values)
[PERSON, BOOK, DEPARTMENT]

-- Define specific sets
Company
  employees: ℙ PERSON -- Set of employees
  managers: ℙ PERSON -- Set of managers
  departments: ℙ DEPARTMENT -- Set of departments

  managers ⊆ employees -- All managers are employees

Set Operations Examples

-- Given sets
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}
C = {5, 6, 7}

-- Set operations
A ∪ B = {1, 2, 3, 4, 5, 6} -- Union: elements in A OR B
A ∩ B = {3, 4} -- Intersection: elements in A AND B
A \ B = {1, 2} -- Difference: elements in A but NOT in B
2 ∈ A = true -- Membership: 2 is in A
9 ∉ A = true -- Non-membership: 9 is not in A

Practical Example

[STUDENT, COURSE]

University
  students: ℙ STUDENT
  courses: ℙ COURSE
  enrolled: STUDENT → ℙ COURSE -- Each student maps to set of courses

  -- Every enrolled course must be a real course
  ∀ s: students • enrolled(s) ⊆ courses

  -- No student can be enrolled in more than 5 courses
  ∀ s: students • #enrolled(s) ≤ 5

Simple Relation Examples

-- Define some people and books
People = {"Alice", "Bob", "Charlie"}
Books = {"Book1", "Book2", "Book3"}

-- Borrowing relation (who borrowed which book)
borrowed = {"Alice" ↦ "Book1", "Bob" ↦ "Book2", "Alice" ↦ "Book3"}

-- This means:
-- Alice borrowed Book1 and Book3
-- Bob borrowed Book2
-- Charlie borrowed nothing

Function Examples

[STUDENT, GRADE, COURSE]

```
GradingSystem
  students: ℙ STUDENT
  courses: ℙ COURSE
  grades: STUDENT → COURSE → GRADE  -- Nested function

  -- Example data:
  -- grades = {
  --   "Alice" ↦ {"Math" ↦ "A", "Science" ↦ "B"},
  --   "Bob" ↦ {"Math" ↦ "C", "History" ↦ "A"}
  -- }
```

Exercise 1: Basic Set Operations

Given:

$A = \{1, 2, 3, 4, 5\}$

$B = \{3, 4, 5, 6, 7\}$

$C = \{5, 6, 7, 8, 9\}$

Compute:

- ❑ $A \cup B$
- ❑ $A \cap C$
- ❑ $B \setminus A$
- ❑ $(A \cup B) \cap C$
- ❑ $A \cap B \cap C$

$A \cup B$ (Union - elements in A OR B)

$\{1, 2, 3, 4, 5, 6, 7\}$

$A \cap C$ (Intersection - elements in A AND C)

$\{5\}$

$B \setminus A$ (Set Difference - elements in B but NOT in A)

$\{6, 7\}$

$(A \cup B) \cap C$ (Union then Intersection)

First: $A \cup B = \{1, 2, 3, 4, 5, 6, 7\}$
Then: $\{1, 2, 3, 4, 5, 6, 7\} \cap \{5, 6, 7, 8, 9\} = \{5, 6, 7\}$
$A \cap B \cap C$ (Triple Intersection)

$A \cap B = \{3, 4, 5\}$
$\{3, 4, 5\} \cap C = \{5\}$

Exercise 2: Library Invariants

Complete the invariants:

```
Library
  books: ℙ BOOK
  available: ℙ BOOK
  borrowed: BOOK → USER

  -- TODO: Add invariants:
  -- 1. Available books are a subset of all books
  -- 2. Borrowed books are a subset of all books
  -- 3. No book can be both available and borrowed
  -- 4. Every book is either available or borrowed
```

```
available ⊆ books
dom borrowed ⊆ books
available ∩ dom borrowed = ∅
available ∪ dom borrowed = books
```

Exercise 3: Social Network Rules

[USER]

SocialNetwork
  users: $\mathbb{P}$ USER
  friends: USER → $\mathbb{P}$ USER
  blocks: USER → $\mathbb{P}$ USER

-- Users can only friend/block registered users
  ∀ u: USER • friends(u) ⊆ users
  ∀ u: USER • blocks(u) ⊆ users

  -- No self-friending
  ∀ u: USER • u ∉ friends(u)

  -- Friendship is symmetric
  ∀ u1, u2: USER • u1 ∈ friends(u2) ⇔ u2 ∈ friends(u1)

  -- Can't friend someone you block
  ∀ u: USER • friends(u) ∩ blocks(u) = ∅

Basic Building Blocks in Z

1. Sets

Sets represent collections of objects in the system state.

Example: Library Management System

[BOOK, USER, COPY]

LibraryState
  books: ℙ BOOK
  users: ℙ USER
  copies: ℙ COPY
  available_copies: ℙ COPY
  borrowed_copies: ℙ COPY

  available_copies ∩ borrowed_copies = ∅
  available_copies ∪ borrowed_copies = copies

Explanation:
[BOOK, USER, COPY] - Basic type declarations
ℙ denotes power set (set of all subsets)
Sets represent collections: all books, all users, all copies
Constraints ensure available and borrowed copies are disjoint but cover all copies

3. Functions

Functions define specific mappings where each input maps to exactly one output.

Example: Library Functions

LibraryFunctions
  copy_to_book: COPY → BOOK
  user_borrow_limit: USER → ℕ
  book_author: BOOK → NAME

  ∀ u: USER • user_borrow_limit(u) ≤ 5

Explanation:

→ denotes a total function (defined for all inputs in domain)
copy_to_book maps each copy to its book
user_borrow_limit maps each user to their maximum borrow limit
Constraint ensures no user can borrow more than 5 books