Regelwerk fuer codebasierte Lean-Dokumentationen

Externe Dokumentation statt Code-Kommentierung

Revision 1.8 mit Lesbarkeitsnotationen fuer alle dokumentierten Versionsstaende und erweitertem Geltungsbereich fuer Code-Lesbarkeit ab v > 0.0605

Normativer Leitsatz

In der betrachteten Arbeitsweise ist der maschinengepruefte Lean-Code die einzige primaere Wahrheitsquelle. Jede Dokumentation ist davon abgeleitete, auditierbare Zweitdarstellung. Zwischenebenen in Form semantischer Code-Kommentare entfallen vollstaendig.

Dokumenttyp	Normatives Regelwerk
Zweck	Erstellung kuenftiger Lean-Dokumentationen ohne semantische Kommentare im Code
Geltungsbereich	Vollstaendige Projektdokumentationen, Pfaddokumentationen, Handoff-Spezifikationen, Datei-/Kettendokumentationen, Theorem-/Beweisdokumentationen, Moduluebersichten
Sprachordnung	Mathematik first, Erlaeuterung second, Interpretation third
Ausgabemedien	`.tex` als Primarformat, daraus kompiliertes PDF als Review-Artefakt
Version	1.8
Datum	28. Maerz 2026
Aenderungsstand	Zusaetzlich zu Revision 1.8 eingearbeitet: chirurgische Korrekturen zur Notationsschicht, insbesondere Konsistenz des Kommentarverbots auch fuer Beispielmodule, Erweiterung der Glossar-Pflichtfelder auf Notationsformen, Rueckverweis von der mathematischen Notation auf projektinterne Lesbarkeitsnotationen gemaess §4.6 sowie explizite Pflichtangaben zu `open scoped`-aktivierten projektinternen Notationen

Editorischer Status: zur unmittelbaren Anwendung in kuenftigen Dokumentationslaeufen bestimmt; nun auch mit Lesbarkeitsnotationen fuer alle dokumentierten Versionsstaende sowie prospektiven Code-Regeln fuer Benennung, Register und Lesbarkeit ab v > 0.0605

Inhaltsverzeichnis

1 Ziel, Status und Begriffsordnung

Dieses Dokument legt verbindliche Regeln fuer Dokumentationen fest, die sich auf eine formal verifizierte Lean-Codebasis stuetzen. Bis einschliesslich Version 0.0605 regelt es ausschliesslich die Ableitung externer Dokumente aus bereits vorliegendem, maschinengeprueftem Code. Fuer Versionen *groesser* als 0.0605 erweitert es seinen Geltungsbereich gezielt auf diejenigen Teile der Code-Gestaltung, die fuer Auditierbarkeit und reviewerfaehige Lesbarkeit unverzichtbar sind: Benennung, Namens- und Notationsregister, Lesbarkeitsnotationen, kommentarfreie Strukturierung und Kohaerenz zwischen Code und Dokumentation.

Hierarchie der Aussageebenen

1. **Ebene 1:** der maschinengepruefte Lean-Code, inklusive importierter Definitionen, Theoreme, Instanzen, Notationen und expliziter Abhaengigkeiten;
2. **Ebene 2:** die externe Dokumentation als formalsprachliche Rekonstruktion von Ebene 1;
3. **Ebene 3:** optionale minimale Erlaeuterung;
4. **Ebene 4:** optionale Interpretation.

Jede hoehere Ebene ist nur zulaessig, wenn sie durch die darunterliegende Ebene gedeckt ist.

1.1 Normative Begriffe

In diesem Regelwerk werden die Begriffe **MUSS**, **DARF NICHT**, **SOLL** und **DARF** streng normativ verwendet.

1.2 Grundsatz der auditierbaren Trennung

Die Dokumentation **MUSS** so geschrieben sein, dass ihre Aussagen gegen den Code, die importierten Bibliotheken und die zitierten primaeren Quellen gegengeprueft werden koennen. Sie **DARF NICHT** eine semantische Mischform aus Kommentar, Intuitionstext und teilweise formaler Aussage darstellen.

2 Primaere Quellenbasis

2.1 Zulaessige Quellenhierarchie

Jede Dokumentation **MUSS** ihre Aussagen auf folgende Quellenhierarchie stuetzen:

1. den konkret betrachteten, erfolgreich gepruefteten Lean-Code der Zielversion;
2. die in diesem Code importierten oder verwendeten Bestandteile von Lean und mathlib;
3. offizielle Dokumentation von Lean und mathlib, soweit sie Begriffe, Deklarationsformen, Namenskonventionen, Section-Verhalten, `omit`/`include`, Dokumentationsstile oder Validierungspraxis praezisiert;
4. fachliche Primaerliteratur *nur* dort, wo das Dokument ausdruecklich den Anschluss an Standardmathematik oder Standardphysik belegt.

2.2 Verwendung offizieller Lean-/mathlib-Quellen

Wo immer Begriffe bereits in Lean oder mathlib definiert, dokumentiert oder stilistisch normiert sind, **MUSS** die Dokumentation diese Begriffe moeglichst uebernehmen, statt eigene Paralleltterminologie zu erzeugen. Das betrifft insbesondere:

- Deklarationsarten wie `def`, `theorem`, `lemma`, `structure`, `class`, `instance`;
- die Rolle von Namespaces, Sections, Imports und hierarchischen Namen;
- Namenskonventionen fuer Typen, Funktionen und Theoreme;
- offizielle oder community-getragene Dokumentationskonventionen.

Folgerung

Die Dokumentation **DARF NICHT** einen dritten Sprachlayer erfinden, wenn ein bereits eingefuehrter Lean-/mathlib-Begriff denselben formalen Gehalt traegt.

3 Axiomatische Arbeitsprinzipien

3.1 Prinzip A: Code zuerst

Jede einzelne prosehafte Aussage **MUSS** aus dem tatsaechlichen Code abgeleitet sein. Kommentare im Code spielen fuer die inhaltliche Gueltigkeit keine Rolle.

3.2 Prinzip B: Mathematik vor Erklaerung

Jede nichttriviale Passage ist in der Reihenfolge

$$\text{Mathematik} \rightarrow \text{minimale Erlaeuterung} \rightarrow \text{optionale Interpretation}$$

zu dokumentieren.

3.3 Prinzip C: Explizite Typ- und Bereichsbindung

Jede mathematische Groesse **MUSS** vor ihrer Verwendung mit ihrem formalen Bereich, Typ oder Definitionskontext eingefuehrt werden. Ein Reviewer **MUSS** insbesondere nicht raten muessen,

- was ein Symbol bezeichnet,
- in welcher Menge, welchem Typ oder welcher Algebra es lebt,
- welche Voraussetzungen global oder lokal gelten,
- ob eine Aussage definitorisch, abgeleitet oder lediglich heuristisch ist.

3.4 Prinzip D: Keine semantische Aufladung ohne Beleg

Formulierungen wie „beschreibt“, „entspricht physikalisch“, „repraesentiert“, „kodiert“, „fasst zusammen“ oder „liefert“ **DARF NICHT** verwendet werden, sofern nicht klar angegeben ist, ob es sich um

1. eine Definition,
2. ein bereits formales Korollar,
3. eine minimale Erlaeuterung oder
4. eine ausdruecklich als Interpretation markierte Passage

handelt.

3.5 Prinzip E: Beweis und Beweisziel sind getrennt zu pruefen

Dass ein Lean-Beweis erfolgreich geprueft wurde, bedeutet nicht automatisch, dass eine informelle Formulierung seinen Gehalt exakt trifft. Deshalb **MUSS** jede Beweisdokumentation am Ende explizit feststellen, ob das in Schritt 1 formulierte Ziel vom tatsaechlichen Theorem wirklich erreicht wurde.

3.6 Prinzip F: Skalierung ist Teil der Korrektheit

Eine Dokumentation gilt nur dann als praezise, wenn ihre Granularitaet an die reale Struktur des Codes angepasst ist. Weder Ueber-Zerlegung in atomare Eintraege noch ungesteuerte Zusammenfassung grosser Ketten sind zulaessig. Deshalb **MUSS** jede Dokumentation explizit angeben, auf welcher Skalenebene sie jeweils arbeitet.

4 Code-Lesbarkeit, Benennung, Lesbarkeitsnotationen und Register

4.1 Prospektiver Geltungsbereich

Dieser Abschnitt **MUSS** fuer Lean-Code mit Zielversionen *groesser* als 0.0605 angewendet werden. Fuer Version 0.0605 und aeltere Staende wirken sie nicht rueckwirkend als Fehlerkatalog; dort **MUSS** die Dokumentation verbleibende Opazitaet durch Glossar, Definitionskatalog, Variablenregister und explizite Aufloesung kompensieren.

Praezisierung des erweiterten Geltungsbereichs

Ab Revision 1.8 betrifft das Regelwerk nicht die gesamte Stilfrage des Codes, wohl aber alle codeinternen Benennungs- und Strukturentscheidungen, die unmittelbar darueber bestimmen, ob der Code als primaere Wahrheitsquelle ohne semantische Hilfskommentare auditierbar bleibt.

4.2 Kommentarverbot bleibt bestehen

Auch fuer Versionen *groesser* als 0.0605 gilt: semantische Kommentare im Lean-Code sind nicht zulaessig. Das betrifft insbesondere --Kommentare, Blockkommentare mit inhaltlicher Prosa und Docstrings, die fachliche Bedeutung transportieren sollen. Zulaessig bleiben allein technisch unvermeidliche, nicht-semantische Artefakte ausserhalb des Beweisgehalts, etwa editor- oder toolbedingte Dateikopfdaten, sofern sie keinen mathematischen oder interpretativen Inhalt einfuehren.

4.3 Regel fuer Variablenbenennung

Variablennamen **MUSS** entweder

1. einem etablierten mathematischen Kurzstandard folgen und im Variablen-, Namens- und Notationsregister als solcher ausgewiesen sein, oder
2. unmittelbar menschenlesbar benannt sein.

Opaque Einbuchstabennamen **DARF NICHT** ohne Registereintrag verwendet werden. Dies betrifft insbesondere globale oder wiederkehrende Parameter, deren Bedeutung nicht bereits universell-standardisiert ist.

Zulaessige Kurzformen

Kurzformen wie i, j, k fuer Laufindizes, m, n fuer ganze Zahlparameter, x, y, z fuer Elemente, sowie etablierte Symbole wie Ω, β oder φ **DARF** beibehalten werden, wenn ihre Bedeutung im jeweiligen mathematischen Kontext standardnah ist und im Register dokumentiert wird. Dagegen **MUSS** projektspezifische oder mehrdeutige Namen wie `b`, `p`, `s`, `B`, `I` oder `A` ausserhalb klarer Standardfaelle lesbar ausgeschrieben oder explizit registriert werden.

4.4 Regel fuer Abkuerzungen und Suffixe

Abkuerzungen, Akronyme und Suffixmuster **MUSS** selbsterklaerend sein oder durch ein Register im selben Projektkontext aufgeloest werden koennen. Ein kuerzelartiger Suffix wie `_ToC` **DARF** als Instanz- oder Herkunftsmarker stehenbleiben, aber nur dann, wenn der Stammname bereits lesbar ist. Namen vom Typ `N_ToC` oder `SN_ToC` **SOLL** fuer neue Versionen durch Formen ersetzt werden, deren Stamm selbst erklaerend ist, etwa `localNet_treeOfCliques` oder `stateNet_treeOfCliques`.

4.5 Regel fuer Deklarations- und Theoremnamen

Deklarationsnamen **SOLL** sich am mathlib-Schema orientieren: Subjekt, Eigenschaft, Modifikator. Projektspezifische Praefixe sind zulaessig, **MUSS** aber im Namens- und Notationsregister erklaert werden, wenn sie nicht selbsterklaerend sind. Gestapelte Namen mit mehreren undurchsichtigen Abkuerzungen **MUSS** fuer neue Versionen vermieden oder in lesbare Teilwoerter aufgeloest werden.

4.6 Pflicht zu Lesbarkeitsnotationen

Diese Pflicht gilt fuer *alle* dokumentierten Versionsstaende, auch fuer Versionen *kleiner oder gleich* 0.0605, weil sie additiv, semantisch neutral und mit geringem Aufwand umsetzbar ist. Wiederkehrende mathematische Ausdruecke, deren ausgeschriebene Lean-Form den formalen Gehalt nicht aendert, aber die menschliche Lesbarkeit deutlich verschlechtert, **MUSS** durch eine explizit dokumentierte Lesbarkeitsnotation begleitet werden, sobald sie fuer einen dokumentierten Pfad tragend sind.

Typische Kandidaten sind insbesondere:

- quadratische oder bilineare Formen,
- Spur-, Adjunktions- oder Konjugationstermen,
- Matrixexponentiale oder projektweit wiederkehrende Transportausdruecke,
- andere wiederkehrende Ausdruecke, deren mathematische Gestalt fuer einen Reviewer sofort erkennbar sein soll.

Semantische Schranke der Notation

Eine Lesbarkeitsnotation **DARF NICHT** neuen mathematischen Inhalt erzeugen. Sie **MUSS** auf bereits definierte Terme, Definitionen oder wohldefinierte Kompositionen expandieren und ihren Gehalt vollstaendig aus dem vorhandenen Code beziehen. Die Notation dient der Sichtbarkeit der bereits bestehenden Mathematik, nicht ihrer stillschweigenden Veraenderung.

Die bevorzugte Realisierung erfolgt **SOLL** innerhalb von Lean selbst, insbesondere ueber `notation` oder `scoped notation` in einem eigenen Notationsmodul oder Namespace. `scoped notation` **SOLL** verwendet werden, wenn Namenskonflikte, lokale mathematische Dialekte oder projektinterne Symbolschichten kontrolliert werden muessen. Externe Python-Skripte, Pretty-Printer oder rein dokumentarische Umschriften **DARF** die Darstellung unterstuetzen, **DARF NICHT** aber als Erfuellung dieser Pflicht gelten, wenn die entsprechende Lesbarkeit nicht auch als Lean-interne Notation oder als formal dokumentierte Expansionsregel abgesichert ist.

4.7 Pflicht zum Variablen-, Namens- und Notationsregister

Jeder dokumentierte Versionsstand **MUSS** ein strukturiertes Variablen-, Namens- und Notationsregister besitzen. Fuer Zielversionen *groesser* als 0.0605 **MUSS** dieses Register als eigenes, versionsgebundenes Artefakt maschinenlesbar oder zumindest streng auswertbar vorliegen, etwa als eigene Lean-Datei, als tabellarischer Anhang oder als anderweitig versionsgebundenes Register. Fuer aeltere Zielversionen **MUSS** mindestens ein dokumentgebundenes, explizites Register mit denselben Kernfeldern vorliegen.

Pro Eintrag **MUSS** mindestens angegeben werden:

1. Lean-Name oder Notationsform;
2. mathematische oder fachliche Bedeutung;
3. Standardbeleg oder Begruendung der Benennung beziehungsweise Notation;
4. Scope: global, pillar-spezifisch, datei-lokal, theorem-lokal oder notationslokal;
5. definierende oder primaer bindende Stelle;
6. explizite Expansion auf den zugrunde liegenden Lean-Term, falls es sich um eine Notation handelt;
7. Verweis auf den korrespondierenden Glossar-Eintrag, sofern vorhanden.

4.8 Kohaerenzpflicht zwischen Code und Dokumentation

Code und Dokumentation **MUSS** dieselben tragenden Namen fuer dieselben tragenden Objekte verwenden. Verwendet der Code einen menschenlesbaren Namen, so **DARF NICHT** die Dokumentation dafuer einen neuen informellen Alternativnamen erfinden. Verwendet der Code aus historischen oder technischen Gruenden einen Kurznamen, so **MUSS** die Dokumentation diesen bei der ersten Verwendung explizit aufloesen und anschliessend konsistent fuehren. Das Glossar und das Variablen-, Namens- und Notationsregister **MUSS** gegenseitig koharent sein; jeder dauerhaft relevante Registereintrag **SOLL** einen Glossargegenpart haben, und jeder projektspezifische Glossarbegriff **SOLL** auf seinen Lean-Namen oder seine definierte Notationsform zurueckfuehren.

4.9 Regel fuer dateiinterne Struktur

Jede `.lean`-Datei **SOLL** ab Versionen *groesser* als 0.0605 einer top-down lesbaren Struktur folgen. Dies bedeutet in der Regel:

1. Imports und Scope-Deklarationen;
2. Namespace-, Section- und Variablenblock;
3. Definitionen und Strukturen;
4. lokale Hilfssaetze;
5. Hauptsaetze oder Endobjekte.

Abweichungen sind zulaessig, **MUSS** aber durch die logische Abhaengigkeit des Inhalts motiviert sein. Willkuerliche Durchmischung definitorischer und abschliessender Teile **DARF NICHT** zum Normalfall werden.

5 Verbindliche Dokumentationsebenen

5.1 Pflicht-Ebenen

Fuer groessere Lean-Codebasen ist zwischen folgenden Dokumentationsebenen zu unterscheiden:

1. **Projekt- oder Pfeiler-Ebene:** gesamter formaler Bereich einer Version;
2. **Pfad-Ebene:** strikter Ableitungspfad von Startobjekten zu Endobjekten;
3. **Pillar-Handoff-Ebene:** explizite Uebergabe zwischen zwei Pfeilern oder semantischen Registern;
4. **Datei-/Ketten-Ebene:** logische Kette innerhalb eines einzelnen Moduls;
5. **Theorem-Gruppen-Ebene:** zusammenhaengende Gruppen von Deklarationen mit gemeinsamer Funktion;
6. **Einzeltheorem-Ebene:** atomare Beweis- oder Definitionseinheit;
7. **Meta-Audit-Ebene:** maschinenpruefbare Marker-Endpunkte zur globalen Verifikation.

5.2 Pflicht zur expliziten Ebenenwahl

Jeder Abschnitt **MUSS** zu Beginn angeben, auf welcher dieser Ebenen er operiert. Ein Dokument **DARF NICHT** zwischen den Ebenen springen, ohne den Ebenenwechsel kenntlich zu machen.

5.3 Skalenregel

Je groesser die Zahl zusammenhaengender Deklarationen, desto staerker **MUSS** die Darstellung zunaechst aggregieren und erst danach in atomare Beweise hineinzoomen. Die Korrektheit einer Dokumentation haengt somit auch davon ab, dass sie einen zusammenhaengenden Beweisfaden als solchen sichtbar macht.

6 Pflicht-Metadaten jedes Dokuments

Jedes kuenftige Dokument **MUSS** auf den ersten Seiten mindestens folgende Metadaten enthalten:

Feld	Inhalt
Dokumenttitel	Praeziser Gegenstand, z. B. „Vollstaendige Dokumentation von Pillar A auf Basis des Codes von v0.0605“
Codebasis	Exakter Projektname, Version, Archivname oder Commit-Bezug
Pruefstatus	build-Status, Warnungsstatus, Umgang mit `sorry`, Axiomen, privaten Hilfssaetzen und `omit`-Stellen
Geltungsbereich	Welche Module, Pfade, Theoreme und Ausschluesse dokumentiert werden
Zielpublikum	z. B. Reviewer ohne laufenden Codezugriff
Sprachstufe	Mathematik first / Erlaeuterung second / Interpretation third
Quellenbasis	Liste der verwendeten primaeren Quellen: Zielcode, mathlib, Lean-Referenz, evtl. Fachliteratur
Audit-Konvention	Wie Aussagen auf Deklarationen zurueckgefuehrt werden (FQNs, Modulpfade, Marker, Anhaenge, Graphen, Tabellen)
Geltigkeitsbindung	Ob das Dokument an einen exakten Commit, ein Archiv oder einen Versionsbereich gebunden ist; bei Fortschreibungen auf neue Versionen: Regel fuer den Delta-Abgleich zur Vorgaengerversion
Reifegrad-Konvention	Welche Reifegradklassifikation das Dokument fuer Module und Pfade verwendet

6.1 Geltigkeitsbindung und Fortschreibung

Jedes Dokument **MUSS** explizit angeben, ob seine Aussagen an

- einen exakten Commit,
- ein bestimmtes Archiv oder
- einen klar begrenzten Versionsbereich

gebunden sind.

Wenn eine spaetere Codeversion strukturelle Aenderungen an dokumentierten Modulen, Pfaden, Handoffs oder Endpunkten einfuehrt, **SOLL** die Fortschreibung des Dokuments einen kompakten Delta-Abgleich zur Vorgaengerversion enthalten. Dieser Delta-Abgleich **SOLL** mindestens benennen:

1. welche Dateien oder Pfade hinzugekommen, entfallen oder umgebaut worden sind;
2. welche Definitionen, Handoffs oder Marker ihren Status geaendert haben;
3. ob bestehende Dokumentteile unveraendert weiter gelten oder revidiert werden mussten.

7 Dokumentarchitektur fuer Voll-Dokumentationen

7.1 Verbindliche Hauptstruktur

Eine vollstaendige Dokumentation eines Moduls, eines Pfeilers oder eines strikten Ableitungspfades **MUSS** mindestens die folgende Grobstruktur besitzen:

1. Ziel und Geltungsbereich;
2. Klassifikation des Dokumentationstyps und der verwendeten Skalenebenen;
3. Definitionskatalog;
4. Projektglossar mit Verweisstruktur;
5. Voraussetzungen und globale Kontexte;
6. Abhaengigkeitsgraph oder Strukturdiagramm des dokumentierten Pfades;
7. strikte Ableitungskette oder Strukturkarte;
8. Datei-/Kettendokumentationen fuer groessere Module;
9. Theoreme, Propositionen, Lemmata, Korollare in logischer Reihenfolge;
10. Beweisdokumentationen fuer nichttriviale Aussagen;
11. Meta-Audit-Marker und Audit-Anhang.

7.2 Pflicht vor jeder Ableitungskette

Bevor eine Ableitungskette startet, **MUSS** das Dokument explizit angeben:

- was der Startpunkt ist (z. B. eine Struktur, ein exportiertes Objekt, ein Namespace oder ein Theorem);
- welche Zwischenobjekte tatsaechlich im strikten Pfad vorkommen;
- welches Endobjekt oder welche Endobjekte erreicht werden;
- welche alternativen, historischen oder nicht-strikten Pfade *nicht* Gegenstand der aktuellen Darstellung sind.

7.3 Pflicht zur Symbolerklaerung

Vor dem ersten Auftreten eines Symbols in einer zentralen Aussage **MUSS** das Symbol mit mindestens folgenden Informationen eingefuehrt werden:

1. Name des Symbols;
2. formaler Typ oder mathematischer Bereich;
3. Herkunft im Code (Definition, Parameter, exportiertes Objekt, Instanz, Notation);
4. bei Notationen: explizite Expansion auf den zugrunde liegenden Lean-Term;
5. Rolle im aktuellen Abschnitt.

8 Datei-/Kettendokumentationen als Pflicht-Zwischenebene

8.1 Wann diese Ebene zwingend ist

Wenn ein Modul mehr als zehn logisch zusammenhaengende Deklarationen enthaelt oder wenn mehrere Deklarationen gemeinsam einen einzigen formalen Bogen bilden, dann **MUSS** vor der Einzeltheorem-Dokumentation eine Datei-/Kettendokumentation stehen.

8.2 Pflichtstruktur einer Datei-/Kettendokumentation

Die Datei-/Kettendokumentation **MUSS** das Schema 1–6 auf Kettenebene verwenden:

1. **Ziel der Kette:** Was die gesamte Datei oder Teilkette als Ganzes erreicht.
2. **Definitionen der Kette:** Zentrale in der Datei eingefuehrte Objekte.
3. **Voraussetzungen der Kette:** Globale Kontexte, Instanzen und importierte Resultate.
4. **Behauptung der Kette:** Welche logische Endaussage oder welches Endobjekt die Kette bereitstellt.
5. **Beweis der Kette:** Gruppierter Verlauf, gegliedert in Theorem-Gruppen statt nur in atomare Sätze.
6. **Zielabgleich der Kette:** Ob die Datei tatsaechlich den behaupteten Gesamtbogen erreicht.

8.3 Theorem-Gruppen

Innerhalb einer Datei-/Kettendokumentation **MUSS** die Deklarationen in funktionale Gruppen gegliedert werden, etwa „Definitionen“, „Invarianz-Lemmata“, „Transportschritte“, „Hauptsatz“. Nur die logisch tragenden oder nichttrivialen Schluesselstufen erhalten danach ein volles Einzeltheorem-Schema 1–6.

9 Verbindliches Schema fuer alle nichttrivialen Beweise

Jeder nichttriviale Beweis **MUSS** nach exakt folgendem Schema dokumentiert werden:

1. Ziel des Beweises
2. Definitionen
3. Voraussetzungen
4. die Behauptung
5. der Beweis
6. Ueberpruefung, ob das in 1 genannte Ziel auch erreicht wurde

Strikter Status dieses Schemas

Dieses Schema ist nicht optional. Es gilt fuer einzelne Theoreme ebenso wie fuer ueber mehrere Module verzweigte Beweisketten, sofern keine hoeherstufige Datei-/Kettendokumentation vorgeschaltet ist.

9.1 Inhaltliche Minimalanforderungen an Schritt 6

Unter „Ziel des Beweises“ **MUSS** angegeben werden:

- welches Erkenntnisziel die Beweisdokumentation verfolgt;
- welche formale Luecke fuer den Reviewer geschlossen werden soll;
- ob das Ziel eine einzelne Aussage, eine Transportkette, eine Strukturerhaltung, eine Typidentifikation oder eine mehrstufige Ableitung betrifft.

9.2 Inhaltliche Minimalanforderungen an Schritt 6

Unter „Definitionen“ **MUSS** stehen:

- alle im Satz vorkommenden Objekte, sofern sie nicht bereits in einem unmittelbar vorangestellten Definitionskatalog formal eingefuehrt wurden;
- saemtliche Notationen, falls deren Bedeutung nicht offensichtlich aus Standardnotation oder mathlib folgt;
- bei importierten Begriffen die Lean-/mathlib-Bezeichnung, wenn diese fuer die Nachpruefbarkeit wichtig ist.

9.3 Inhaltliche Minimalanforderungen an Schritt 6

Unter „Voraussetzungen“ **MUSS** zwischen folgenden Ebenen unterschieden werden:

1. globale Voraussetzungen des Dokuments;
2. abschnittsspezifische Voraussetzungen;
3. theorem-lokale Annahmen;
4. importierte Standardvoraussetzungen aus Typklassen, Instanzen oder bereits vorausgesetzten Strukturen;
5. durch `omit` oder `include` veraenderte Section-Kontexte.

9.4 Inhaltliche Minimalanforderungen an Schritt 6

Unter „die Behauptung“ **MUSS** die eigentliche Aussage moeglichst nah an der Lean-Form wiedergegeben werden. Wo sinnvoll, **SOLL** zusaetzlich eine mathematische Schreibweise angegeben werden, die aber semantisch dieselbe Aussage repraesentiert.

9.5 Inhaltliche Minimalanforderungen an Schritt 6

Unter „der Beweis“ **MUSS** die Beweisstruktur explizit nach den wirklich verwendeten Schritten rekonstruiert werden. Dabei gilt:

- Ein Beweis **DARF NICHT** als blosses Narrativ beschrieben werden.
- Der Uebergang zwischen den Schritten **MUSS** logisch und nicht psychologisch formuliert sein.

- Moduluebergreifende Abhaengigkeiten **MUSS** sichtbar gemacht werden.
- Verwendete Hilfssaetze **SOLL** mit Namen oder mindestens mit ihrer eindeutigen Funktion genannt werden.
- Falls der Code nur durch Transport ueber Isomorphismen, Restriktionen, Instanzen oder Reindexierungen funktioniert, **MUSS** dieser Transport explizit gemacht werden.

9.6 Inhaltliche Minimalanforderungen an Schritt 6

Schritt 6 **MUSS** explizit pruefen,

- ob der in Schritt 1 genannte Zielgehalt erreicht wurde,
- ob nur die formale Behauptung gezeigt wurde oder auch die beabsichtigte Lesart,
- ob verbleibende Restfragen rein interpretativ sind,
- ob das Resultat enger oder weiter ist als die informelle Erwartung.

10 Operationales Trivialitaetskriterium

10.1 Zweck

Damit Autoren nicht ad hoc entscheiden, welche Saetze voll aufgedroeselt werden, gilt das folgende operationale Kriterium.

10.2 Trivial

Ein Beweis **DARF** als trivial behandelt werden, wenn beide Bedingungen erfuellt sind:

1. Die Lean-Taktikkette besteht aus hoechstens einer unmittelbar definitorischen Form wie `rfl`, `simp`, `exact`, `infer_instance` oder einer offensichtlichen Kombination aus `constructor` plus direktem `simp`;
2. die Aussage folgt ohne Zwischenkonstruktion unmittelbar aus Definitionen oder bereits explizit dokumentierten Standardinstanzen.

10.3 Nichttrivial

Ein Beweis ist als nichttrivial zu behandeln, sobald mindestens eines der folgenden Merkmale vorliegt:

- `calc`-Ketten oder mehrere `have`-Zwischenschritte;
- moduluebergreifende `exact`-Referenzen oder Transport ueber Hilfssaetze;
- veraenderter Section-Kontext, etwa ueber `omit`;
- mehr als fuenf nichttriviale Taktikzeilen;
- mehrere logisch gekoppelte Teilziele, deren Zusammenhang fuer den Reviewer nicht sofort definitorisch sichtbar ist.

Folgerung

Grenzfaelle sind zugunsten der expliziten Dokumentation zu entscheiden. Ueber-Dokumentation ist in Zweifelsfaellen der Unter-Dokumentation vorzuziehen, aber nur innerhalb der korrekten Skalenebene.

11 Regeln fuer Definitionskataloge

11.1 Vollstaendigkeit vor Oekonomie

Ein Definitionskatalog **MUSS** so vollstaendig sein, dass ein Reviewer den formalen Gehalt der nachfolgenden Aussagen ohne Ratespiel lesen kann. Es **DARF NICHT** vorausgesetzt werden duerfen stillschweigende Kontexte, die nur aus dem Code-Layout ersichtlich waeren.

11.2 Pflichtbestandteile pro vollem Kerneintrag

Jeder *zentrale* Eintrag in einem Definitionskatalog **MUSS** enthalten:

1. Symbol bzw. Lean-Name oder Notationsform;
2. mathematische Schreibweise;
3. Typ/Sort/Menge/Algebra;
4. formale Definition oder definierende Herkunft;
5. eventuelle Abhaengigkeit von Parametern;
6. kurze Erlaeuterung *erst nach* der formalen Angabe.

11.3 Hierarchische Schichtung des Katalogs

Definitionskataloge **MUSS** geschichtet werden:

- **Kernstrukturen und Kernobjekte:** voller Sechs-Felder-Eintrag;
- **abgeleitete Schluesselobjekte:** verkuerzter Eintrag mit Name, Typ, definierender Satz/Datei und Rolle;
- **technische Hilfsstrukturen:** tabellarische Liste mit Name, Typ und Dateiverweis.

11.4 Reihenfolge der Darstellung

Fuer jeden Definitionseintrag gilt die Reihenfolge

$$\text{Name} \to \text{Typ/Bereich} \to \text{Definition} \to \text{Erlaeuterung}.$$

Interpretation ist hiervon getrennt zu halten und spaeter auszuweisen.

12 Projektglossar und Verlinkungsregeln

12.1 Pflicht zum Projektglossar

Jede Voll-Dokumentation einer groesseren projektinternen Lean-Codebasis **MUSS** ein eigenes Projektglossar enthalten. Dies gilt insbesondere dann, wenn projektspezifische Begriffe verwendet werden, die

- kein 1:1-Aequivalent in mathlib besitzen,
- bekannte mathlib- oder Standardbegriffe nur angenaehert treffen,
- in mehreren Pillars mit unterschiedlicher semantischer Last auftreten oder
- fuer den Reviewer nicht bereits aus der Standardterminologie von Lean, mathlib oder der Fachliteratur eindeutig lesbar sind.

12.2 Pflichtfelder pro Glossareintrag

Jeder Glossareintrag **MUSS** mindestens enthalten:

1. Lean-Name oder Notationsform;
2. mathematische Schreibweise oder explizite Kennzeichnung „keine eigene mathematische Schreibweise“;
3. falls der Eintrag eine Lesbarkeitsnotation beschreibt: explizite Expansion auf den zugrunde liegenden Lean-Term;
4. Kurzdefinition in hoechstens zwei Saetzen;
5. definierende Datei oder definierender Modulpfad;
6. Status „mathlib-Aequivalent vorhanden“ oder „kein mathlib-Aequivalent“;
7. falls ein Aequivalent vorhanden ist: Angabe des naechsten mathlib-Begriffs und der relevanten Differenz;
8. falls der Begriff einen bekannten Lean-/mathlib-Begriff ueberschattet oder daran erinnert: explizite Markierung dieser Ueberschneidung.

12.3 Schichtung des Glossars

Das Projektglossar **MUSS** mindestens in zwei Schichten gegliedert werden:

- **Kernbegriffe des Projekts:** voller Glossareintrag;
- **abgeleitete oder lokale Fachbegriffe:** verkuerzter Eintrag mit Lean-Name, Kurzdefinition und Dateiverweis.

Ein Glossar **DARF NICHT** durch Gleichbehandlung aller Hilfsbegriffe unlesbar werden.

12.4 Verlinkungsregel als Wiki-Ersatz in LaTeX

Wenn das Primaerformat `.tex` ist, **SOLL** das Glossar ueber `\label` und `\hyperref` als zentrale Navigationsschicht nutzbar gemacht werden. Bei der ersten Verwendung eines projektspezifischen Kernbegriffs im laufenden Dokument **SOLL** ein Hyperlink auf den Glossareintrag gesetzt werden.

12.5 Reviewer-Zweck

Das Glossar ersetzt keine formalen Definitionen im Haupttext. Es stellt aber sicher, dass ein Reviewer bei projektspezifischen Begriffen nicht fortlaufend in den Code springen muss, um die minimale semantische Grundlast des Dokuments zu rekonstruieren.

12.6 Abgrenzung zu Definitionskatalogen

Das Projektglossar ergaenzt die Definitionskataloge als dokumentuebergreifendes Nachschlagewerk. Dabei gilt:

- Begriffe, die in einem Definitionskatalog bereits voll behandelt sind, **DARF** im Glossar mit Kurzeintrag und Verweis auf den Katalog erscheinen;
- Begriffe, die in keinem lokalen Definitionskatalog stehen, aber projekt- oder pfaduebergreifend relevant sind, **MUSS** im Glossar eigenstaendig dokumentiert werden;
- der Definitionskatalog traegt die volle formale Last des aktuellen Abschnitts, waehrend das Glossar die wiederkehrende semantische Grundorientierung des Gesamtprojekts sichert.

13 Pillar-Handoff-Spezifikation als eigener Dokumentationstyp

13.1 Anwendungsfall

Sobald zwischen zwei Pfeilern, Subsystemen oder semantisch verschiedenartigen Formalismen Daten uebergeben werden, **MUSS** eine eigene Handoff-Spezifikation erstellt werden. Ein blosses Erwaehnen der beteiligten Module reicht nicht aus.

13.2 Pflichtfelder einer Handoff-Spezifikation

Eine Handoff-Spezifikation **MUSS** mindestens folgende Felder enthalten:

Feld	Inhalt
Quell-Pillar / Quellbereich	Modul- oder Pfadbereich, aus dem Daten exportiert werden
Ziel-Pillar / Zielbereich	Modul- oder Pfadbereich, der die exportierten Daten weiterverarbeitet
Uebergebene Daten	explizite Liste der uebergebenen Objekte mit Lean-Namen und Typen
Mitgegebene Beweise / Instanzen	Welche Eigenschaften bereits auf Quellseite vorliegen und auf Zielseite weitergenutzt werden
Nicht uebergebene Struktur	Welche Strukturen *nicht* uebergeben, sondern im Ziel neu konstruiert werden

Rechtsstatus des Handoffs	definitorischer Export, Satz-Transport, Repackaging, Adapter, oder gemischter Status
Audit-Punkte	Welche Deklarationen die Handoff-Spezifikation pruefbar machen

13.3 Zweck

Die Handoff-Spezifikation trennt streng zwischen *uebergebenen Daten*, *uebergebenen Eigenschaften* und *neu auf Zielseite konstruierten Strukturen*. Dadurch wird sichtbar, ob ein Paradigmenwechsel nur rhetorisch behauptet oder formal sauber dokumentiert wird.

14 Regeln fuer moduluebergreifende Beweisketten

Nichttriviale Lean-Entwicklungen verlaufen oft ueber mehrere Module, Reexports und Hilfssaetze. Deshalb **MUSS** jede aus mehreren Modulen zusammengesetzte Doku mindestens eine explizite Beweiskettentabelle und einen Abhaengigkeitsgraphen enthalten.

14.1 Beweiskettentabelle

Schritt	Modul	Deklaration	Rolle im Gesamtbeweis
S1	Zielmodul oder Exportmodul	exakt benannter Satz / Definition	Startpunkt oder Importvoraussetzung
S2	Zwischenmodul	Hilfssatz / Transport / Restriktion	logischer Uebergang
S3	weiteres Modul	Isomorphie / Instanz / Kompatibilitaet	Fortsetzung der Ableitung
S4	Zielmodul	Schlusssatz / Endobjekt	Endresultat

14.2 Abhaengigkeitsgraph als Pflichtbestandteil

Fuer jeden dokumentierten strikten Pfad **MUSS** ein Diagramm beigefuegt werden, das mindestens die beteiligten Module, ihre Import- oder Abhaengigkeitsrelationen und die Position des dokumentierten Pfades innerhalb dieser Struktur zeigt.

14.3 Pflichtangaben bei privaten oder unsichtbaren Hilfssaetzen

Falls der Code private Hilfssaetze oder lokal nicht exportierte Konstruktionen benutzt, **MUSS** die Dokumentation deren Rolle entweder

- ueber oeffentlich sichtbare Ersatzformulierungen rekonstruieren oder
- transparent angeben, dass ein interner Zwischenschritt existiert, dessen Funktion beschrieben, dessen Name aber nicht als oeffentliche Referenz nutzbar ist.

Ein Reviewer **MUSS** wissen, ob eine referenzierte Stufe extern verifizierbar ist oder nur indirekt ueber oeffentliche Konsequenzen.

15 Instanzen, `include`/`omit` und reale Voraussetzungskontexte

15.1 Grundsatz

Die Dokumentation **MUSS** den *tatsaechlichen* Voraussetzungskontext eines Satzes angeben, nicht bloss den umgebenden `variable`- oder `section`-Kontext.

15.2 Pflicht bei `omit`

Wenn ein Theorem oder eine Definition durch `omit` Instanzen oder Variablen aus dem globalen Section-Kontext ausblendet, dann **MUSS** das Dokument explizit angeben:

1. welche Variablen oder Instanzen im Abschnitt global vorhanden sind;
2. welche davon fuer die konkrete Deklaration per `omit` nicht in den Satzkopf aufgenommen werden;
3. welchen effektiven Satzkontext die Deklaration dadurch tatsaechlich besitzt.

15.3 Pflicht bei Instanz-Synthese

Wenn zentrale Typklassen-Instanzen per `infer_instance` aufgeloest werden, **MUSS** die Dokumentation mindestens die Aufloesungskette bis zur fuer den Reviewer sachlich relevanten Quelle angeben. Dabei sind mindestens drei Ebenen zu unterscheiden:

- unmittelbare Instanznutzung im Zielmodul,
- Zwischenquelle der Instanzbereitstellung,
- sachlich relevante Ursprungskonstruktion.

15.4 Instanz-Kettenblatt

Fuer zentrale Pfade **SOLL** ein eigenes Instanz-Kettenblatt beigefuegt werden, das in Tabellenform Name, Typklassenkopf, Aufloesungsquelle und Dokumentrolle auflistet.

16 Regeln fuer mathematische Schriftform

16.1 Lean-nahe Formalitaet

Die mathematische Schrift **SOLL** moeglichst nahe an der Lean-Struktur bleiben. Das bedeutet insbesondere:

- explizite Bindung freier Variablen;
- klare Trennung zwischen Daten, Klassenannahmen und zu zeigenden Aussagen;
- keine implizite Einfuehrung neuer Objekte in laufendem Text;
- keine Auslassung zentraler Bereichsinformation.

16.2 Notation

Nicht-standardisierte Notation **MUSS** in einem eigenen Unterabschnitt angegeben werden. Dies umfasst auch projektinterne Lesbarkeitsnotationen gemaess §4.6, sofern sie im dokumentierten Pfad tragend sind oder wiederkehrende Kernausdruecke des Codes lesbar machen sollen. Standardnotation **DARF** ohne Zusatz verwendet werden, sofern ihre Rolle aus dem mathematischen Kontext und dem verwendeten Lean-/mathlib-Objekt eindeutig hervorgeht.

16.3 Gleichungen und definitorische Zeichen

Es **SOLL** deutlich unterschieden werden zwischen

- definitorischer Einfuehrung,
- Gleichheit als Satzinhalt,
- kanonischer Identifikation,
- isomorpher oder aequivalenter Uebersetzung.

Wenn mehrere dieser Ebenen im selben Argument vorkommen, **MUSS** die Dokumentation sie explizit markieren.

17 Formaler Reifegrad von Modulen und Pfaden

17.1 Pflichtklassifikation

Jedes dokumentierte Modul und jeder dokumentierte Pfad **MUSS** mit einem formalen Reifegrad versehen werden. Zulaessig sind mindestens die folgenden Stufen:

1. **vollstaendig bewiesen:** tragende Eigenschaften sind formal als Theoreme hergeleitet;
2. **strukturell vollstaendig, Eigenschaften als Prop-Felder:** Interface-Struktur ist vorhanden, aber zentrale Eigenschaften liegen als Felder oder offene Verpflichtungen vor;
3. **reiner Interface-Hook:** Konstruktion dient nur als Anschluss- oder Platzhalterstruktur.

17.2 Pflichtwirkung

Der Reifegrad **MUSS** in jeder Uebersichtstabelle und in jedem Kapitelkopf sichtbar sein. Eine Dokumentation **DARF NICHT** einen Interface-Hook sprachlich wie einen voll bewiesenen Pfad praesentieren.

18 Regeln fuer Erlaeuterung und Interpretation

18.1 Minimale Erlaeuterung

Erlaeuterung ist zulaessig, aber nur nach der formalen Angabe. Sie **SOLL** vor allem

- den Zweck einer Definition knapp angeben,
- die Rolle eines Zwischenschritts in der Beweiskette sichtbar machen,
- den Unterschied zwischen formaler Aussage und beabsichtigter Lesart klaeren.

18.2 Interpretation als eigener Status

Interpretative Passagen **MUSS** eindeutig als Interpretation gekennzeichnet werden. Sie **DARF NICHT** mit den Definitionen oder den Beweisschritten verschmolzen werden.

Zulaessige Markierung

Geeignet sind etwa getrennte Unterabschnitte mit Titeln wie „Minimale Erlaeuterung“ oder „Interpretative Bemerkung“. Ungeeignet ist es, interpretative Saetze in eine Definition oder in einen Beweisschritt einzuschmuggeln.

18.3 Verbotene Verkuerzungen

Die Dokumentation **DARF NICHT**

- aus einem formal gezeigten Existenzsatz unmittelbar eine physikalische Interpretation machen,
- eine Konstruktion als „das System“ oder „die Dynamik“ bezeichnen, wenn sie formal nur ein Kandidat, eine Darstellung oder ein transportiertes Objekt ist,
- Standardbegriffe aus der Literatur beanspruchen, ohne die entsprechenden formalen Voraussetzungen explizit gemacht zu haben.

19 Meta-Audit-Marker als primaeres Verifikationsinstrument

19.1 Pflicht

Wenn eine Codebasis einen eigenen Meta-Layer mit Marker-Theoremen oder Audit-Endpunkten besitzt, dann **MUSS** dieser Meta-Layer in der Dokumentation als primaeres Verifikationsinstrument verwendet werden.

19.2 Marker-Tabelle

Fuer jeden dokumentierten Pfad **MUSS** eine Tabelle existieren, die mindestens enthaelt:

Marker	Modul	Status	Wofuer der Marker auditierbar steht
Marker-Theorem	Meta-Modul	bewiesen / offen	Endpunkt oder Audit-Siegel fuer eine globale Eigenschaft

19.3 Wirkung

Meta-Audit-Marker ersetzen nicht die innere Beweisdokumentation, aber sie verankern fuer den Reviewer, welche globalen Eigenschaften am Ende des Pfades formal als erreicht gelten.

20 Regeln fuer Auditierbarkeit

20.1 Nachverfolgbarkeit jeder tragenden Aussage

Jede tragende Aussage im Dokument **MUSS** rueckfuehrbar sein auf

- einen exakten Lean-Namen,
- einen genauen Modulpfad,
- eine offizielle Lean-/mathlib-Quelle oder
- eine explizit genannte fachliche Primaerquelle.

20.2 Pflicht zum Namens- und Pfadverzeichnis

Am Ende des Dokuments **MUSS** ein Audit-Anhang stehen, der mindestens enthaelt:

1. Modulpfad;
2. vollqualifizierter oder hinreichend eindeutiger Deklarationsname;
3. Rolle im Dokument;
4. Verweis auf die Stelle, an der die Deklaration verwendet wird;
5. Reifegrad-Einordnung;
6. falls relevant: Markerbezug, Handoff-Bezug oder Instanzkettenbezug.

20.3 Pruefprotokoll

Fuer Dokumentationen von hohem Anspruch **SOLL** zusaetzlich ein Pruefprotokoll angegeben werden, etwa:

- erfolgreicher Build der Zielversion;
- keine `sorry`-Abhaengigkeit im betrachteten Pfad;
- keine nicht deklarierten zusaetzlichen Axiome;
- gegebenenfalls Ueberpruefung per `#print axioms` oder weitergehender Validierung;
- Sicht auf die fuer die Globalaussagen relevanten Meta-Audit-Marker.

21 Regeln fuer mathlib-Bezugsnahmen und Lean-Kontextabgleich

21.1 Uebernahme vor Neuformulierung

Wenn eine Definition, ein Klassenname, eine Strukturbedeutung oder eine Namenskonvention bereits in mathlib dokumentiert ist, **SOLL** diese Form bevorzugt uebernommen werden. Eine Neuformulierung ist nur dann zulaessig, wenn sie fuer den konkreten Dokumentzweck noetig ist und den formalen Gehalt nicht veraendert.

21.2 Pflicht zum mathlib-Abgleich fuer Kernstrukturen

Fuer jede zentrale projektinterne Struktur oder Klasse, die einem bekannten Lean-/mathlib-Konzept verwandt ist, **MUSS** die Dokumentation einen expliziten Abgleich enthalten. Dieser Abgleich **MUSS** mindestens angeben:

1. welches mathlib-Konzept das naechste Aequivalent waere;
2. worin die projektinterne Struktur formal davon abweicht;
3. ob ein expliziter Brueckensatz, Transport oder keine formale Bruecke vorliegt;
4. ob die Abweichung nur dokumentarisch, nur implementatorisch oder mathematisch wesentlich ist.

21.3 Pflicht zur Kennzeichnung abweichender Terminologie

Jede projektinterne Terminologie, die ueber Lean-/mathlib-Standards hinausgeht, **MUSS** als projektspezifisch markiert werden. Dabei **MUSS** klar angegeben werden, auf welchen formalen Lean-Objekten diese Terminologie aufsetzt.

21.4 Keine verdeckte Bedeutungsverschiebung

Ein mathlib-Begriff **DARF NICHT** im Dokument stillschweigend in einem engeren, weiteren oder physikalisch aufgeladenen Sinn verwendet werden, ohne dass diese Verschiebung explizit thematisiert wird.

21.5 Adaption der mathlib-Dokumentationskonventionen

Die mathlib-Dokumentationskonventionen dienen fuer externe Voll-Dokumentationen als stilistisches Leitbild, auch wenn im Code selbst bewusst keine Docstrings verwendet werden. Das bedeutet insbesondere:

- Abschnittstitel und Gliederung **SOLL** sich an mathlib-nahen Funktionen orientieren, etwa „Main definitions“, „Main statements“, „Notation“, „Implementation notes“, „References“;
- bei strukturellen Abweichungen wegen des externen Formats **MUSS** die Abweichung kenntlich gemacht werden;
- die externe Dokumentation ersetzt die Docstrings nicht dadurch, dass sie sie imitiert, sondern dadurch, dass sie ihre dokumentarische Funktion auf Dokumentebene systematisch uebernimmt;
- wo ein Lean-Theorem oder eine Struktur erkennbar an mathlib-Namens- oder Dokumentationsmuster anschliesst, **SOLL** dies als Querverweis oder Stilhinweis sichtbar gemacht werden;
- wo die externe Voll-Dokumentation ueber mathlib hinausgeht, etwa durch Kettenebenen, Handoff-Spezifikationen oder Meta-Audit-Marker, **MUSS** diese Mehrstruktur als projektspezifische Ergaenzung ausgewiesen werden.

21.6 Namespace-, Scope- und Namenskonvention

Jedes Dokument **MUSS** zu Beginn angeben,

- welche Namespaces als geoeffnet vorausgesetzt werden,
- welche `open scoped`-Aktivierungen oder Notationsscopes benutzt werden,
- welche Notations-Namespaces geoeffnet sind und welche projektinternen Notationen dadurch aktiv werden,
- ob im Haupttext vollqualifizierte Namen oder geoeffnete Namen verwendet werden.

Die gewaehlte Konvention **MUSS** danach konsistent durchgehalten werden. Bei der ersten Verwendung einer tragenden Deklaration **SOLL** entweder der vollqualifizierte Name oder eine eindeutig rueckverfolgbare Namensform angegeben werden.

21.7 Pflicht bei `@[simp]`-Lemmata

Wenn `@[simp]`-Lemmata fuer die Lesbarkeit oder den Erfolg einer Beweiskette wesentlich sind, **MUSS** die Dokumentation diese als Simplifier-Lemmata kennzeichnen. Es **SOLL** knapp angegeben werden, ob ein Lemma vor allem

- definitorisches Entfalten,
- kanonische Normalform,
- algebraische Standardvereinfachung oder
- einen projektspezifisch wichtigen Umschreibeschritt

liefert.

21.8 Pflicht bei Universen, Sorten und `noncomputable`

Wenn der formale Gehalt eines Pfades oder einer Struktur von Universen, Sorten oder dem `noncomputable`-Status sachlich betroffen ist, **MUSS** die Dokumentation dies explizit angeben. Insbesondere **SOLL** festgehalten werden:

- ob eine Konstruktion universenpolymorph oder auf einen festen Sort-/Type-Kontext beschraenkt ist;
- ob `noncomputable` nur die Ausfuehrbarkeit betrifft oder fuer die mathematische Lesart weitere Konsequenzen im Dokument hat;
- ob aus dem `noncomputable`-Status fuer den Reifegrad, die Auditierung oder die praktische Nachvollziehbarkeit ein eigener Hinweis noetig ist.

22 Spezialregel fuer Anforderungen der Form „vollstaendige Dokumentation“

Wenn eine Anforderung lautet „Erstelle eine vollstaendige Dokumentation von X auf Basis des Codes von Version Y“, dann **MUSS** das resultierende Dokument mindestens Folgendes leisten:

1. den Geltungsbereich explizit abgrenzen;
2. alle zentralen Groessen formal definieren;

3. den strikten Ableitungspfad oder die strikten Ableitungspfade vollstaendig ausweisen;
4. fuer groessere Module Datei-/Kettendokumentationen einschieben;
5. alle tragenden Theoreme und Hilfssaetze im Zusammenhang anordnen;
6. nichttriviale Beweise nach dem Schema 1–6 dokumentieren;
7. Handoffs zwischen Pfeilern oder Registern gesondert und formal typisiert dokumentieren;
8. historische, alternative oder nicht-strikte Pfade gesondert kennzeichnen;
9. ein Projektglossar und die mathlib-Abgleichsstellen so beifuegen, dass projektspezifische Begriffe und begriffliche Abweichungen ohne Ratespiel lesbar werden;
10. Meta-Audit-Marker, Abhaengigkeitsgraphen und Audit-Anhang so beifuegen, dass ein Reviewer nicht fortlaufend in den Code springen muss, um die Dokumentstruktur zu verstehen.

Reviewer-Kriterium

Ein Dokument erfuellt den Anspruch „vollstaendig“ nur dann, wenn ein fachlich kompetenter Reviewer den formalen Aufbau, die logische Lastverteilung und den Status der globalen Endpunkte ohne Ratespiel rekonstruieren kann.

23 Templates

23.1 Pflichttemplate fuer Einzeltheoreme

Einzeltheorem

1 Ziel des Beweises
Formuliere das konkrete Erkenntnisziel, nicht nur den Titel des Satzes.

2 Definitionen
Liste alle benoetigten Objekte, Typen, Notationen und importierten Standardbegriffe.

3 Voraussetzungen
Trenne globale, abschnittslokale, theorem-lokale und durch `omit`/`include` modifizierte Kontexte.

4 die Behauptung
Gib die Aussage moeglichst Lean-nah und zusaetzlich in mathematischer Schrift an, falls dies zur Lesbarkeit beitraegt.

5 der Beweis
Rekonstruiere die wirklichen Beweisschritte, inklusive moduluebergreifender Transporte, Hilfssaetze und Schlussschritte.

6 Ueberpruefung, ob das in 1 genannte Ziel auch erreicht wurde
Pruefe explizit die Deckung von informellem Ziel und formaler Aussage.

23.2 Pflichttemplate fuer Datei-/Kettendokumentation

Datei oder Kette

1 Ziel der Kette
Was als Gesamtbogen der Datei oder Theorem-Gruppe erreicht wird.

2 Definitionen der Kette
Zentrale Objekte, die in der Datei eingefuehrt oder fortgeschrieben werden.

3 Voraussetzungen der Kette
Globale Kontexte, Imports, relevante Instanzen, wirksamer Section-Kontext.

4 Behauptung der Kette
Formale Endaussage oder Endobjekt der Kette.

5 Beweis der Kette
Gruppierter Verlauf nach funktionalen Teilbloecken und Schluesselstufen.

6 Zielabgleich der Kette
Ob die Gesamtdatei den behaupteten Bogen tatsaechlich schliesst.

23.3 Pflichttemplate fuer Handoff-Spezifikation

Pillar-Handoff

Quelle und Ziel
Nenne Quell- und Zielmodul beziehungsweise Quell- und Zielpfeiler.

Uebergebene Daten
Liste Lean-Namen und Typen aller exportierten Groessen.

Mitgegebene Eigenschaften
Welche Saetze, Instanzen oder Strukturannahmen mitgehen.

Nicht uebergebene Struktur
Welche Zielstrukturen neu konstruiert werden.

Auditpunkte
Welche Deklarationen den Handoff pruefbar machen.

24 Freigabekriterien vor Publikation

Ein Dokument ist erst dann freigabefaehig, wenn alle folgenden Punkte bejaht werden koennen:

1. Ist die Zielversion der Codebasis exakt benannt?
2. Ist der dokumentierte Geltungsbereich explizit?
3. Sind alle zentralen Symbole formal eingefuehrt?
4. Ist jede tragende Aussage rueckverfolgbar?
5. Sind nichttriviale Beweise im Schema 1–6 dokumentiert?
6. Sind fuer groessere Module Datei-/Kettendokumentationen vorhanden?
7. Sind Handoffs gesondert spezifiziert?

8. Ist ein Projektglossar vorhanden und an den Erstverwendungen sinnvoll verlinkt?
9. Sind mathlib-Abgleich und projektspezifische Differenzen fuer Kernstrukturen dokumentiert?
10. Sind `omit`-Stellen, Scope-Kontexte und Instanz-Ketten sichtbar gemacht?
11. Sind Namespace-, Notations- und gegebenenfalls `open scoped`-Kontexte explizit?
12. Sind Reifegrad, Marker und Abhaengigkeitsgraphen angegeben?
13. Sind `@[simp]`-Lemmata, Universen und `noncomputable`-Aspekte dort markiert, wo sie sachlich relevant sind?
14. Ist die Geltigkeitsbindung des Dokuments explizit und sind Fortschreibungen gegenueber der Vorgaengerversion benannt, sofern eine Revision vorliegt?
15. Existiert fuer den dokumentierten Versionsstand ein Variablen-, Namens- und Notationsregister?
16. Sind tragende Lesbarkeitsnotationen fuer wiederkehrende, schwer lesbare Kernausdruecke vorhanden oder ist ihre Abwesenheit begruendet?
17. Sind Notationsform, Expansion und Scope dort sichtbar gemacht, wo sie fuer das Verstaendnis des Pfades relevant sind?
18. Falls die Zielversion > 0.0605 ist: liegt das Variablen-, Namens- und Notationsregister als eigenes, versionsgebundenes Artefakt vor?
19. Falls die Zielversion > 0.0605 ist: sind opake Kurzformen entweder standardbelegt oder im Register explizit aufgeloest?
20. Falls die Zielversion > 0.0605 ist: stimmen tragende Code-Namen und tragende Dokumentationsnamen sichtbar ueberein?
21. Falls die Zielversion > 0.0605 ist: ist die dateiinterne Struktur der dokumentierten Module top-down lesbar oder begruendet abweichend?
22. Bleibt der Code kommentar- und docstringfrei, soweit das Regelwerk dies fordert?
23. Sind Erlaeuterung und Interpretation sauber getrennt?
24. Sind mathlib-/Lean-Begriffe dort uebernommen, wo sie bereits existieren?
25. Ist sichtbar, welche Teile strikt formal und welche Teile nur interpretativ sind?
26. Ist ein Reviewer ohne dauerndes Quellcode-Raetseln arbeitsfaehig?

25 Schlussformel

Das vorliegende Regelwerk erzwingt eine harte Trennung zwischen formalem Wahrheitskern und externer Darstellung. Genau darin liegt sein Nutzen: Der Code bleibt kommentar- und interpretationsfrei; die Dokumentation wird dafuer umso strenger, expliziter und auditierbarer. Revision 1.8 erweitert diese Disziplin in zwei Richtungen: Einerseits bleiben fuer Versionen *groesser* als 0.0605 die prospektiven Code-Regeln fuer Benennung, Register und Struktur bestehen. Andererseits werden fuer *alle* dokumentierten Versionsstaende Lesbarkeitsnotationen als eigene Pflichtschicht eingefuehrt, sobald wiederkehrende Kernausdruecke den mathematischen Gehalt in ausgeschriebener Lean-Form verdecken. Damit gilt nun eine dreifache Kohaerenzpflicht: Der externe Text

MUSS den Code praezise rekonstruieren, der kommentarfreie Code **MUSS** fuer neue Versionen lesbar benannt sein, und die verwendeten Notationen **MUSS** ihre Expansion und ihren Scope so offenlegen, dass ein Reviewer ohne semantisches Raten von der Symbolschicht zum Kernterm zurueckgehen kann.

A Empfohlene externe Referenzbasis

Die folgenden offiziellen Quellen sind fuer kuenftige Dokumentationen besonders relevant:

1. **Lean Language Reference:** Kapitel zu Source Files and Modules, Namespaces and Sections, Definitions, Theorems, Attributes, Type Classes, Universes, Inductive Types / Structures, Notations and Macros (insbesondere Notations und Custom Operators) sowie Validating a Lean Proof.
2. **Lean Community / mathlib:** Library Style Guidelines, Documentation Style, Naming Conventions, Style-Linter und das Lean-Glossar der Community-Seiten.
3. **Theorem Proving in Lean 4:** insbesondere die Kapitel zu Structures and Records sowie Type Classes, wenn Instanzdesign, Instanzsynthese und die operative Seite von Typklassen fuer die Dokumentation relevant sind.

B Mikrobeispiel fuer die Anwendung des Regelwerks

Dieser Anhang ist ein didaktisches Musterbeispiel. Er demonstriert die Form einer kuenftigen Dokumentation, ohne den Anspruch zu erheben, bereits die endgueltige inhaltliche Ausfuellung eines konkreten Zielsatzes zu ersetzen. Bei realer Verwendung **MUSS** die hier gezeigte Struktur gegen die tatsaechliche Lean-Deklaration der Zielversion validiert werden.

B.1 Beispiel 1: Schematische Einzeltheorem-Dokumentation

Muster fuer ein nichttriviales Einzeltheorem

1 Ziel des Beweises
Es soll dokumentiert werden, dass ein Theorem mit einem Namen der Form `gate_LAPLACIAN_ADMISSIBLE` tatsaechlich die fuer den weiteren Pfad benoetigte Zulassungs- oder Admissibilitaetsaussage liefert, statt nur eine Teilkomponente davon zu zeigen.

2 Definitionen

- `AdmissibleOp`: projektspezifischer Begriff; der genaue formale Gehalt ist aus der definierenden Datei zu uebernehmen und im Definitionskatalog oder Glossar zu verankern.
- `Weight`: projektspezifischer Begriff; ebenfalls mit Lean-Name, Typ, definierender Herkunft und gegebenenfalls mathlib-Abgleich zu dokumentieren.
- alle im Theorem auftretenden Parameter, Typen, Indizes und Notationen sind vor der Behauptung formal einzufuehren.

3 Voraussetzungen
Hier werden der wirksame Section-Kontext, globale Annahmen, relevante Instanzen sowie eventuelle `omit`-Abweichungen aufgelistet. Falls `AdmissibleOp` mehrere Teilbedingungen buendelt, **MUSS** explizit angegeben werden, welche davon bereits vorausgesetzt und welche im Theorem neu gezeigt werden.

4 die Behauptung
Die Lean-Aussage ist moeglichst nah an der Originalform wiederzugeben. Danach folgt eine mathematische Umschrift, die denselben Gehalt hat, aber fuer den Reviewer lesbarer ist.

5 der Beweis
Der Beweis wird nach wirklichen Schritten rekonstruiert, etwa: Entfaltung der relevanten Definitionen, Separierung der Teilziele, Verweis auf bereits bewiesene Komponenten, Zusammenbau zur Endaussage. Falls der Code mehrere Teilbedingungen in einem einzigen `constructor`-Block oder ueber gebuendelte Felder zeigt, **MUSS** die Dokumentation diese Zerlegung sichtbar machen.

6 Ueberpruefung, ob das in 1 genannte Ziel auch erreicht wurde
Es ist explizit festzuhalten, ob der Satz die volle Admissibilitaet, nur eine Teilmenge der Admissibilitaetsbedingungen oder eine Hilfsaussage ueber den beteiligten Operator beweist.

B.2 Beispiel 2: Muster fuer Glossar- und mathlib-Abgleich

Element	Musterhafte Ausfuellung
Glossareintrag `AdmissibleOp`	Lean-Name: `AdmissibleOp`. Mathematische Schreibweise: falls keine etablierte Kurznotation existiert, explizit als „keine eigene mathematische Schreibweise“ markieren. Kurzdefinition: knappe projektinterne Beschreibung in hoechstens zwei Saetzen. Dateiverweis: definierende Datei. mathlib-Status: naechstes Aequivalent oder „kein mathlib-Aequivalent“. Differenz: in welchem Punkt die Projektstruktur enger, weiter oder anders zugeschnitten ist.
Glossareintrag `Weight`	Lean-Name: `Weight`. Typ/Bereich und definierende Datei angeben. Falls `Weight` kein mathlib-Aequivalent hat, ist dies explizit zu markieren; falls doch, ist die Differenz zum naechsten mathlib-Begriff kurz anzugeben.

mathlib-Abgleich	Fuer jede Kernstruktur ist festzuhalten: naechstes mathlib-Aequivalent, formale Differenz, vorhandene oder fehlende Bruecke sowie der Status „nur implementatorisch abweichend“ oder „mathematisch wesentlich abweichend“.

B.3 Beispiel 3: Muster fuer Lesbarkeitsnotation

Muster fuer eine semantisch neutrale Lesbarkeitsnotation

Ausgangspunkt ist ein bereits definierter Lean-Term, etwa eine bilineare oder quadratische Form. Wenn derselbe Ausdruck projektweit wiederkehrt und in ausgeschriebener Summenform schwer lesbar wird, **SOLL** eine Lean-interne Notation eingefuehrt werden, zum Beispiel als scoped notation in einem eigenen Namespace.
Die Dokumentation **MUSS** dann mindestens festhalten. Ein begleitendes Lean-Beispielmodul, das diese Notation als Artefakt ausliefert, **MUSS** selbst ebenfalls dem Kommentarverbot aus §4.2 genuegen; fachliche Prosa dazu gehoert in das PDF, nicht in die Lean-Datei.

- die Notationsform, etwa $\langle\langle f, L, g \rangle\rangle$;
- die mathematische Lesart, etwa „bilineare Form von f und g relativ zu L“;
- die Lean-Expansion, etwa auf einen bereits definierten Term der Form `bilinForm L f g`;
- den Scope der Notation, etwa ueber einen eigenen Namespace mit `open` oder `open scoped`;
- die Aussage, dass die Notation keinen neuen mathematischen Gehalt einfuehrt, sondern nur den vorhandenen Term lesbarer macht.

Eine rein externe Python-Darstellung ohne zugehoerige Lean-Notation oder dokumentierte Expansionsregel genuegt diesem Regelwerk nicht.

B.4 Nutzen des Mikrobeispiels

Der Zweck dieses Anhangs ist nicht inhaltliche Vollstaendigkeit, sondern operative Eindeutigkeit. Er zeigt Autoren, wie Schema 1–6, Glossar, Definitionskatalog und mathlib-Abgleich in einer realen Dokumentation zusammenspielen sollen.

C Quellenverzeichnis

1. Lean Language Reference, *Source Files and Modules*. https://lean-lang.org/doc/reference/latest/Source-Files-and-Modules/
2. Lean Language Reference, *Namespaces and Sections*. https://lean-lang.org/doc/reference/latest/Namespaces-and-Sections/
3. Lean Language Reference, *Definitions*. https://lean-lang.org/doc/reference/latest/Definitions/Definitions/
4. Lean Language Reference, *Theorems*. https://lean-lang.org/doc/reference/latest/Definitions/Theorems/
5. Lean Language Reference, *Attributes*. https://lean-lang.org/doc/reference/latest/Attributes/

6. Lean Language Reference, *Type Classes*. https://lean-lang.org/doc/reference/latest/Type-Classes/

7. Lean Language Reference, *Universes*. https://lean-lang.org/doc/reference/latest/The-Type-System/Universes/

8. Lean Language Reference, *Inductive Types* (einschliesslich Structures). https://lean-lang.org/doc/reference/latest/The-Type-System/Inductive-Types/

9. Lean Language Reference, *Notations*. https://lean-lang.org/doc/reference/latest/Notations-and-Macros/Notations/

10. Lean Language Reference, *Custom Operators*. https://lean-lang.org/doc/reference/latest/Notations-and-Macros/Custom-Operators/

11. Lean Language Reference, *Validating a Lean Proof*. https://lean-lang.org/doc/reference/latest/ValidatingProofs/

12. Theorem Proving in Lean 4, *Structures and Records*. https://lean-lang.org/theorem_proving_in_lean4/Structures-and-Records/

13. Theorem Proving in Lean 4, *Type Classes*. https://lean-lang.org/theorem_proving_in_lean4/Type-Classes/

14. Lean Community, *Library Style Guidelines*. https://leanprover-community.github.io/contribute/style.html

15. Lean Community, *Documentation style*. https://leanprover-community.github.io/contribute/doc.html

16. Lean Community, *Mathlib naming conventions*. https://leanprover-community.github.io/contribute/naming.html

17. Mathlib Documentation, *Mathlib.Tactic.Linter.Style*. https://leanprover-community.github.io/mathlib4_docs/Mathlib/Tactic/Linter/Style.html