符號作為算子：從靜態字元到可組合計算閉包

Symbols as Operators: From Static Characters to Composable Computational Closure

v0.1 / 2026-07-27 / Neo.K with Aletheia / 初版完成

https://thisoneisneok.com/html/papers/pu-2-02.html

SECTION

Symbols as Operators: From Static Characters to Composable Computational Closure

論文編號： PU-2-02

系列：「程式宇宙書系」第 2 冊《程式語言的本質》

作者： Neo.K with Aletheia

機構： EveMissLab／一言諾科技有限公司

版本： v0.1

日期： 2026 年 7 月 27 日

SECTION

摘要

前篇指出程式語言不等於文字，文字只是程式結構的一種線性序列化投影。本篇進一步追問：若語言不以文字為本體，那麼構成語言的「符號」究竟是什麼？

傳統語言學與程式語言實作常將符號理解為字元、token、關鍵字、操作符或語法節點。然而，這種理解只描述符號的表面辨識方式，沒有完整描述其在程式世界中的可執行角色。本文提出：

> 【Symbol
> ≠
> Character
> ≠
> Token】

並將可執行符號定義為：

> 帶有輸入域、輸出域、前置條件、後置條件、效果、權限、證據與組合規則的語意算子。
> 形式上：

> 【ω
> =
> <
> I,
> O,
> P,
> Q,
> E,
> A,
> V,
> C
> >】

其中：

- I：Input Domain，輸入域；
- O：Output Domain，輸出域；
- P：Precondition，前置條件；
- Q：Postcondition，後置條件；
- E：Effect，效果；

- A：Authorization，授權；
- V：Verification，驗證；
- C：Composition，組合規則。

本文核心命題是：

> 【Executable Symbol
> =
> Operator with Governed Meaning】

一個符號不因被寫成 +、send、approve、圖形節點或自然語言片語而自動具有完整算子地位。真正決定其身份的是：

- 它作用於什麼；
- 它產生什麼；
- 它要求哪些世界條件；
- 它可能造成哪些狀態與外部效果；
- 誰有權使用；
- 如何驗證；
- 能否與其他算子合法組合。

本文進一步建立三層符號觀：

- 表面符號：字元、圖示、詞彙或節點外觀；
- 語意符號：連接問題世界概念與關係；
- 算子符號：可對值、狀態、事件、權限或世界執行受約束變換。

形式上：

> σ_(surface)
> xrightarrowbind
> σ_(semantic)
> xrightarrowoperationalize

同一算子可以擁有多種表面符號：

> π_(text)(ω),
>
> π_(visual)(ω),
>
> π_(speech)(ω),
>
> π_(machine)(ω)

因此，表面表示不同，不代表算子不同。反之，兩個外觀相同的符號若具有不同輸入域、效果、權限或Runtime，也可能是不同算子。

本文提出「算子身份」：

> Identity(ω)
> =
> (
> SemanticRole,
> InputType,
> OutputType,
> EffectSignature,
> Invariant,
> Version
>)

而非單純依靠名稱或字元。

本文區分六類基礎算子：

- 值算子：對值進行轉換；
- 結構算子：建立、連接、分解或重組結構；
- 狀態算子：改變權威狀態；
- 事件算子：形成或轉譯已發生事實；

- 效果算子：作用於外部數位或物理世界；
- 治理算子：授權、撤銷、批准、申訴、暫停或改變規則。

  本文指出，傳統語言往往高度形式化值算子與結構算子，卻把狀態、事件、效果與治理算子隱藏在函式命名、框架、API、middleware 與人工流程中。這使語言能精確描述「怎麼算」，卻難以直接描述「誰能改變什麼世界、如何證明、失敗如何恢復」。

  本文進一步建立算子組合：

> ω□
> ∘
> ω□

只有在輸出—輸入相容、前後條件可接合、效果不衝突、權限合法、證據可傳遞時，組合才成立。

本文將組合合法性形式化為：

> Composable
> (
> ω□,ω□
>)
> ⇔
> O□⊆ I□
> ∧
> Q□|= P□
> ∧
> Compatible(E□,E□)
> ∧
> Authorized(A□,A□)
> ∧
> Traceable(V□,V□)

這表示型別相容只是組合的第一層。若兩個函式型別吻合，但一個會產生不可逆付款效果，另一個假設可回滾，則語意上仍不可安全組合。

本文提出「計算閉包」概念。給定一組基礎算子 Ω□ 與組合規則 mathcal C，可生成：

> 【Closure
> (
> Ω⊠,
> mathcal C
>)】

閉包中的每一複合算子都應仍具有可推導的：

- 輸入與輸出；
- 前後條件；
- 效果；
- 權限；
- 失敗；
- 證據；
- 可恢復性。

  如果組合後這些資訊遺失，就只形成語法閉包，而非可治理計算閉包。

  本文提出：

> 【Syntactic Closure
> ≠
> Semantic Closure
> ≠
> Operational Closure
> ≠
> Governed Closure】

其中：

- 語法閉包：可形成合法結構；
- 語意閉包：組合後仍有可理解意義；

- 操作閉包：可被 Runtime 執行；
- 治理閉包：權限、責任、證據與恢復仍完整。

  本文進一步分析算子的純度、效果、可逆性、冪等性、交換性、結合性、分配性與時序依賴。傳統代數律在涉及狀態、外部效果與權限後，不能無條件沿用。例如：

> ω_a ∘ ω_b
> ≠
> ω_b ∘ ω_a

寄信後再撤銷權限，與先撤銷權限再嘗試寄信，顯然不同。

本文使用算術、資料轉換、狀態更新、工作流、AI Agent 工具、醫療批准與物理裝置控制等案例，說明如何將靜態符號提升為帶有世界責任的算子。本文最後提出可證偽研究綱領，包括表面符號—算子身份錯配、組合合法性檢查、效果簽章完整度、閉包保持、AI 算子選擇、可逆性推導、治理算子建模、跨投影算子同一性與算子圖對事故預防的效果。

本文為下一篇〈語法—語意—效果〉建立算子地基：語法將被重新理解為算子形成規則，語意是算子所指向的世界關係，效果則是算子對數位與物理世界產生的差分。

關鍵詞： 符號、算子、計算閉包、組合、效果簽章、治理算子、程式語言本體論、後文本語言

SECTION

Abstract

This paper argues that executable symbols are not merely characters or tokens. They are semantic operators with governed meaning.

An operator is modeled as:

> ω
> =
> <
> I,
> O,
> P,
> Q,
> E,

The paper distinguishes surface symbols, semantic symbols, and operator symbols; defines operator identity; develops six classes of operators; and formalizes composability beyond type compatibility.

It distinguishes syntactic, semantic, operational, and governed closure, and argues that a mature programming language should preserve effects, authorization, evidence, failure, and recovery through composition.

Keywords: symbols, operators, computational closure, composition, effect signatures, governed computation

字元 + 本身不會加法。

單字 send 本身不會寄信。

圖形中的箭頭本身不會建立流程。

只有當符號被連接到：

- 輸入；
- 輸出；
- 世界語意；
- Runtime；
- 效果；
- 權限；
- 驗證；

  它才成為可執行算子。

  所以：

> 【Visible Symbol
> not⇒
> Executable Operator】

語言設計不能只問符號看起來像什麼，而必須問符號在世界中能做什麼。

本文定義：

> 【ω
> =
> <
> I,
> O,
> P,
> Q,
> E,
> A,
> V,
> C
> >】

SECTION

2.1 輸入域 I

算子能接受什麼值、狀態、事件或世界實體。

SECTION

2.2 輸出域 O

算子會產生什麼值、狀態、事件或證據。

SECTION

2.3 前置條件 P

執行前世界必須滿足什麼。

SECTION

2.4 後置條件 Q

成功後應保證什麼。

SECTION

2.5 效果 E

算子可能：

- 讀取；
- 寫入；
- 發布；
- 呼叫外部系統；
- 修改權限；
- 改變物理世界。

SECTION

2.6 授權 A

誰可以在何種情境使用此算子。

SECTION

2.7 驗證 V

如何證明輸入、執行與結果合法。

SECTION

2.8 組合規則 C

算子可以與哪些算子、以何種順序與條件組合。

SECTION

3.1 表面符號

可以是：

- 字元；
- 關鍵字；
- 圖示；
- 按鈕；
- 節點；
- 語音；
- 自然語言片語。

SECTION

3.2 語意符號

指向：

- 問題世界概念；
- 實體；
- 關係；
- 狀態；
- 規則；
- 角色。

SECTION

3.3 算子符號

能對世界或計算結構執行受規則約束的變換。

SECTION

3.4 形成鏈

> σ_(surface)
> xrightarrowbind
> σ_(semantic)
> xrightarrowoperationalize
> ω

SECTION

3.5 多表面投影

同一算子可投影為：

- +；
- add(a,b)；
- 視覺加法節點；
- 語音「將兩數相加」；
- AST Add(lhs,rhs)。

  表面不同，不代表算子不同。

  本文提出：

> Identity(ω)
> =
> (
> SemanticRole,
> InputType,
> OutputType,
> EffectSignature,

SECTION

4.1 名稱不足

兩個算子都叫 save，可能分別表示：

- 儲存草稿；
- 提交正式狀態；
- 寫入外部資料庫；
- 發布不可逆內容。

名稱相同不代表身份相同。

SECTION

4.2 字元不足

符號 + 在不同語言中可能表示：

- 數值加法；
- 字串連接；
- 集合聯集；
- 型別組合；
- 路徑拼接。

SECTION

4.3 語意角色

算子必須綁定其問題世界角色，例如：

```
Money.Add
String.Concat
Set.Union
```

Permission.Grant

SECTION

4.4 效果簽章

算子身份還應包含：

pure
reads-state
writes-state
emits-event
external-effect
physical-effect
governance-effect

SECTION

4.5 版本

算子語意改變時，即使名稱不變，也可能成為新版本算子。

SECTION

5.1 值算子

對值進行轉換：

> ω_V
> :
> X
> $\to$
> Y

例如：

- 加法；
- 比較；
- 格式轉換；

- 編碼；
- 聚合。

SECTION

5.2 結構算子

建立、連接、分解或重組結構：

- 建立節點；
- 連接關係；
- 合併集合；
- 展開樹；
- 建立模組；
- 形成工作流。

SECTION

5.3 狀態算子

改變權威狀態：

> ω_s
> :
> S⊠
> →
> S_(t+1)

例如：

- 啟用帳號；
- 完成任務；

- 更新訂單；
- 撤銷權限。

SECTION

5.4 事件算子

形成、驗證、轉譯或路由事件：

- 建立事件；
- 去重；
- 排序；
- 聚合；
- 轉換事件版本；
- 將外部主張提升為領域事件。

SECTION

5.5 效果算子

作用於外部數位或物理世界：

- 寄信；
- 付款；
- 發布；
- 刪除；
- 控制設備；
- 呼叫外部服務。

SECTION

5.6 治理算子

改變誰能做什麼，以及規則如何生效：

- 授權；
- 撤銷；
- 批准；
- 申訴；
- 暫停；
- 改規則；
- 轉移責任；
- 啟動人工覆核。

SECTION

6.1 值算子高度成熟

傳統語言擅長：

- 算術；
- 邏輯；
- 集合；
- 字串；
- 型別；
- 資料結構。

SECTION

6.2 狀態算子部分顯式

狀態改變通常以：

- 賦值；
- 方法呼叫；
- 資料庫寫入；

  表達，但其世界語意常被隱藏。

SECTION

6.3 效果算子外掛化

寄信、付款、發布等效果通常只是一般函式呼叫，語言本身不知道其不可逆性與風險。

SECTION

6.4 治理算子隱形

權限、批准與申訴常散落於：

- middleware；
- policy；
- UI；
- 人工流程；
- 組織制度。

SECTION

6.5 算子語言的擴展

未來語言應把狀態、事件、效果與治理算子提升為一等結構，而非只靠命名慣例。

傳統函式簽章：

> f
> :
> X
> →
> Y

只描述輸入與輸出。

本文提出擴展算子簽章：

> 【ω
> :
> I
> xrightarrow[
> A,V
>]P,E,Q
> O】

其中：

- P：前置條件；
- E：效果；
- Q：後置條件；
- A：授權；
- V：驗證。

SECTION

7.1 失敗簽章

還可加入：

> F_ω
> =
> {
> rejected,
> retryable,
> unknown,
> partial,
> irreversible
> }

SECTION

7.2 時間簽章

> T_ω
> =
> <
> deadline,
> timeout,
> effectiveTime,
> latency
> >

SECTION

7.3 可恢復性簽章

reversible
compensatable
idempotent
reconstructable
irreversible

給定：

> ω☒
> :
> I☒
> →
> O☒

以及：

> ω☒
> :
> I☒
> →
> O☒

若 O☒ 可作為 I☒，則可考慮：

> ω☒
> ∘
> ω☒

但型別相容只是最低條件。

SECTION

8.1 完整組合判準

> Composable
> (
> ω☒,ω☒
>)
> ⇔
> O☒⊆ I☒
> ∧
> Q☒|= P☒

SECTION

8.2 輸出—輸入相容

前一算子產物必須能被下一算子理解。

SECTION

8.3 後置—前置接合

前一算子的保證，必須足以滿足下一算子的要求。

SECTION

8.4 效果相容

效果不能破壞彼此假設。

SECTION

8.5 授權可傳遞性

有權執行第一步，不代表有權執行第二步。

SECTION

8.6 證據可追蹤

組合後應保留每一步的身份、事件與證據鏈。

例如：

```
chargeCard : PaymentRequest -> Receipt
sendReceipt : Receipt -> Message
```

型別上可組合。

但仍需知道：

- Receipt 是否代表授權或最終結算；

- 是否有權向此收件人寄送；
- 是否含敏感資料；
- 付款結果未知時是否可寄信；
- 重試是否會重複扣款或寄信。

所以：

> 【TypeCompatible
> not⇒
> WorldCompatible】

完整語言需檢查世界語意與效果。

SECTION

10.1 純算子

純算子在相同輸入下產生相同輸出，且不修改外部狀態：

> $\omega_p(x)=y$

SECTION

10.2 效果算子

效果算子依賴或改變世界：

> $\omega_e(x,S\boxtimes)=(y,S_{t+1},E)$

SECTION

10.3 純度的價值

純算子較容易：

- 測試；
- 重排；
- 快取；
- 並行；
- 推理；
- 重現。

SECTION

10.4 效果不可消失

效果算子不能被偽裝成純值函式。

SECTION

10.5 效果邊界

可將純計算集中於內部，把外部效果集中於明示邊界，使責任與恢復更清楚。

SECTION

11.1 結合律

若：

> (ω□ ∘ ω□) ∘ ω□
> =
> ω□ ∘ (ω□ ∘ ω□)

則組合順序的括號可改變而不改變結果。

但涉及：

- 交易邊界；
- 超時；
- 資源；
- 補償；
- 權限；

  時，不一定成立。

SECTION

11.2 交換律

> ω_a ∘ ω_b
> =
> ω_b ∘ ω_a

對許多效果算子不成立。

例如先撤銷權限再寄信，與先寄信再撤銷權限不同。

SECTION

11.3 冪等律

> ω ∘ ω
> =
> ω

對讀取、設定狀態等算子可能成立；對付款、寄信與累加通常不成立。

SECTION

11.4 分配律

效果、錯誤與時序可能使分配律失效。

SECTION

11.5 語言責任

語言與型別系統應標示哪些律可安全使用，而不是把純數學直覺無條件帶入世界效果。

SECTION

12.1 可逆算子

若存在 ω^(-1)：

> ω^(-1)
> o
> ω
> =
> id

則稱其在特定域內可逆。

SECTION

12.2 數位可逆

值轉換、局部狀態或未發布草稿可能可逆。

SECTION

12.3 補償可逆

某些世界效果不能真正逆轉，只能透過補償接近原狀。

SECTION

12.4 不可逆

例如：

- 已閱讀訊息；
- 已公開資料；
- 已造成物理傷害；
- 已影響人類決策；
- 已洩漏秘密。

SECTION

12.5 可逆性簽章

算子應明示：

```
exactly-reversible
state-reversible
compensatable
explainable-only
irreversible
```

SECTION

12.6 組合後可逆性

即使每個局部算子可逆，複合後也可能因外部觀測與時間而不可逆。

SECTION

13.1 冪等算子

> ω(ω(x))
> =
> ω(x)

SECTION

13.2 世界冪等

世界冪等關心重複行動是否產生重複世界效果。

SECTION

13.3 身份

冪等通常需要行動身份，而不只是相同資料。

SECTION

13.4 條件冪等

算子可能只在：

- 相同目標；
- 相同行動 ID；
- 相同版本；
- 時限內；

  冪等。

SECTION

13.5 冪等性推導

複合算子是否冪等，不能只看單一算子，還需考慮順序與外部效果。

SECTION

14.1 時間是算子條件

某些算子只在期限、窗口或狀態階段中合法。

SECTION

14.2 等待算子

wait-until
wait-for-event
wait-for-human
wait-with-timeout
等待不是沒有執行，而是正式時間狀態。

SECTION

14.3 超時算子

超時可產生：

- 取消；
- 結果未知；
- 升級；
- 補償；
- 降級。

SECTION

14.4 排程算子

排程需保存：

- 執行時間；

- 時區；
- 重複；
- 過期；
- 權限；
- 任務身份。

SECTION

14.5 時序組合

先後、並行、競爭、選擇與匯合應成為語言的一等組合方式。

SECTION

15.1 失敗不是無結果

算子可能產生正式失敗：

> ω(x)
> →
> Failure(f)

SECTION

15.2 失敗處理算子

例如：

- retry；
- catch；
- compensate；

- escalate；
- fallback；
- request-human-review。

SECTION

15.3 錯誤值與世界失敗

回傳 Error 是資料表示；真正世界失敗還包含：

- 影響；
- 已完成效果；
- 所有者；
- 可恢復性；
- 證據。

SECTION

15.4 組合中的失敗傳播

語言應明示：

- 向上傳播；
- 局部吸收；
- 轉換；
- 補償；
- 合併；
- 終止。

SECTION

15.5 失敗閉包

複合算子的失敗集合應可從子算子推導，而不是在 Runtime 才意外出現。

SECTION

16.1 授權

grant(actor, capability, scope, time)

SECTION

16.2 撤銷

revoke(actor, capability)

SECTION

16.3 批准

把待定行動轉入可執行狀態。

SECTION

16.4 申訴

對分類、拒絕或結果提出正式異議。

SECTION

16.5 暫停

停止某主體或流程繼續產生效果。

SECTION

16.6 修改規則

改變其他算子的合法性與效果。

SECTION

16.7 高階性

治理算子是作用於算子、權限與規則之上的高階算子。

> $$\omega_g : \Omega \to \Omega'$$

因此需更高層的版本、審核與證據。

SECTION

17.1 算子接受算子

> $$H : \Omega \to \Omega$$

例如：

- map；
- filter；
- retry；
- authorize；
- audit；

- parallelize。

SECTION

17.2 高階效果

高階算子可能改變：

- 執行次數；
- 並行方式；
- 錯誤傳播；
- 權限；
- 可觀測性；
- 恢復策略。

SECTION

17.3 retry 作為高階算子

> Retry⊠(ω)

不能只重複呼叫，還需知道 ω 是否安全重試。

SECTION

17.4 authorize 作為高階算子

> Authorize_A(ω)

將授權條件附加於算子。

SECTION

17.5 audit 作為高階算子

> Audit(ω)

應保存行動、事件、效果與證據，而不是只列印文字日誌。

SECTION

18.1 定義

算子圖：

> G_Ω
> =
> (
> Ω,
> E_C,
> E_D,
> E_F,
> E_A
>)

其中：

- E_C：組合邊；
- E_D：資料與狀態依賴；
- E_F：效果與失敗傳播；
- E_A：授權與治理關係。

SECTION

18.2 不只是呼叫圖

呼叫圖只表示誰呼叫誰。

算子圖還表示：

- 為何可組合；
- 誰有權；
- 效果如何累積；
- 哪些失敗可傳播；
- 哪些證據可連接。

SECTION

18.3 算子圖驗證

可檢查：

- 無擁有者效果；
- 不可逆效果未批准；
- 無安全重試；
- 權限提升；
- 不可滿足前置條件；
- 證據鏈中斷。

SECTION

18.4 多尺度

高階算子可封裝子圖，並保留摘要簽章。

SECTION

19.1 基礎算子集合

給定：

> Ω□
> =
> {
> ω□,ω□,⋯,ω□
> }

與組合規則 mathcal C。

SECTION

19.2 閉包

> 【Closure
> (
> Ω□,
> mathcal C
>)】

表示所有可由基礎算子合法組成的複合算子。

SECTION

19.3 語法閉包

只保證組合結果仍是合法語法結構。

SECTION

19.4 語意閉包

組合後仍具有可理解的世界語意。

SECTION

19.5 操作閉包

組合後能被 Runtime 執行。

SECTION

19.6 治理閉包

組合後仍能推導：

- 權限；
- 責任；
- 效果；
- 失敗；
- 證據；
- 恢復。

SECTION

19.7 四層不等價

> 【Syntactic Closure
> ≠
> Semantic Closure
> ≠
> Operational Closure
> ≠
> Governed Closure】

能編譯不代表能合法、安全、可追責地改變世界。

複合算子 Ω^ast 應保持：

SECTION

20.1 型別可推導

輸入與輸出仍可被判定。

SECTION

20.2 條件可推導

整體前置與後置條件可由子算子合成。

SECTION

20.3 效果可推導

能知道整體：

- 讀取；
- 寫入；
- 外部呼叫；
- 不可逆效果；
- 世界範圍。

SECTION

20.4 失敗可推導

失敗集合與傳播方式可見。

SECTION

20.5 權限可推導

整體所需權限不應低於子算子要求。

SECTION

20.6 證據可推導

能從複合結果追蹤回每個重要子效果。

SECTION

20.7 恢復可推導

能知道：

- 哪些步驟可重試；
- 哪些可回滾；
- 哪些需補償；
- 哪些不可逆。

SECTION

21.1 封裝複合算子

一個複合子圖可以被封裝為高階算子。

SECTION

21.2 摘要簽章

封裝後仍需對外揭露：

- 輸入；
- 輸出；
- 前置條件；
- 效果；
- 失敗；
- 權限；
- 證據；
- 版本。

SECTION

21.3 隱藏實作，不隱藏責任

抽象可以隱藏內部步驟，但不能隱藏外部效果與失敗責任。

SECTION

21.4 洩漏抽象

若使用者必須知道大量內部細節才能正確使用算子，摘要簽章不完整。

SECTION

21.5 危險抽象

把付款、刪除與發布包裝成普通 execute()，是把世界風險隱藏在名稱之下。

SECTION

22.1 加法

> +
> :
> mathbb R×mathbb R
> →
> mathbb R

SECTION

22.2 前置條件

在有限機器中仍需考慮：

- 溢位；
- 精度；
- 單位；
- 空值；
- 數值域。

SECTION

22.3 單位算子

3 公尺 + 2 秒 在一般物理語意下不可組合。

型別系統可把量綱納入算子簽章。

SECTION

22.4 金額

金額加法還需要相同幣別、精度與會計規則。

因此，即使最基本的 + 也不是脫離語意域的普遍靜態字元。

SECTION

23.1 解析算子

ParseDate : Text -> Date

SECTION

23.2 隱藏條件

仍需：

- 地區格式；
- 時區；
- 曆法；
- 無效日期；
- 模糊日期。

SECTION

23.3 組合

Text
→ ParseDate
→ ConvertTimezone
→ ScheduleEvent
最後一步產生世界效果，不能因前兩步純計算而把整條鏈視為純算子。

SECTION

23.4 證據

排程成功還需保存：

- 原始文字；

- 解析結果；
- 時區；
- 使用者確認；
- 事件 ID。

  算子：

  ```
  CompleteTask
  ```

  其完整簽章應包含：

- 輸入：任務、行動者、證據；
- 前置：任務仍可完成；
- 權限：行動者有權主張完成；
- 效果：任務進入完成待核或已完成；
- 事件：TaskCompletionClaimed；
- 失敗：已取消、版本衝突、證據不足；
- 恢復：撤回或爭議程序。

  它不是單純：

  ```
  task.completed = True
  ```

  工作流：

  ```
  ApproveRelease
  → DeployRelease
  → VerifyRelease
  → ShiftTraffic
  ```

SECTION

25.1 組合條件

- 批准對應相同版本；
- 部署環境正確；

- 驗證證據滿足流量切換前置；
- 切換權限存在。

SECTION

25.2 失敗

若部署成功但驗證失敗，需要：

- 保留部署狀態；
- 禁止流量切換；
- 回滾或人工檢查。

SECTION

25.3 閉包

整體 PublishRelease 算子必須暴露所有重要效果與失敗，而不能只回傳布林值。

SECTION

26.1 工具不是名稱集合

Agent 看見 send_email，還需知道：

- 收件人作用域；
- 是否不可逆；
- 是否冪等；
- 是否需要批准；
- 成功證據；
- 結果未知處理。

SECTION

26.2 算子選擇

Agent 應根據算子簽章與世界狀態選擇工具，而非只依自然語言名稱相似度。

SECTION

26.3 組合計畫

計畫必須在執行前檢查：

- 效果總和；
- 權限總和；
- 不可逆節點；
- 失敗與補償；
- 證據鏈。

SECTION

26.4 高階治理

可使用：

RequireApproval(SendEmail)
Audit(DeleteFile)
RetrySafe(ReadWebPage)
將治理與恢復策略附加於工具算子。

算子：

ApproveTreatment
不是普通布林更新。

它包含：

- 合格專業者；

- 病人身份；
- 資料新鮮度；
- 治療風險；
- 同意；
- 時效；
- 替代方案；
- 撤回；
- 稽核證據。

這是治理算子與狀態算子的組合。

算子：

StartMotor
應具有：

- 裝置身份；
- 感測狀態；
- 安全區域；
- 操作者權限；
- 轉速限制；
- 緊急停止；
- 命令回執；
- 物理效果確認。

若感測器無法確認，結果應是：

command-sent
physical-outcome-unverified

而非直接宣稱馬達已啟動。

- 字元即算子： 只憑外觀判定符號身份。
- 名稱即語意： 同名函式被假定具有相同世界角色。
- 型別即完整簽章： 只有輸入輸出，缺少條件、效果與權限。
- 型別相容即世界相容： 忽略後置—前置、效果與證據。
- 純度偽裝： 外部效果被隱藏在普通函式中。
- 效果外掛化： 語言無法直接表示付款、發布、權限與物理作用。
- 治理缺席： 批准、申訴與規則修改只存在於人工流程。
- 代數律濫用： 對效果算子錯誤假定交換、結合或冪等。
- 可逆性誤判： 數位狀態可回復便宣稱世界效果可逆。
- 補償等同逆算子： 忽略殘差、時間與人類影響。
- 冪等只看 payload： 不保存行動身份與作用域。
- 等待非狀態化： 長時等待只由執行緒或臨時記憶維持。
- 超時即失敗： 結果未知被誤判並盲目重試。
- 失敗集合隱藏： 複合算子可能失敗方式無法推導。
- 高階算子改變語意： retry、parallel 等包裝器未反映效果變化。
- 呼叫圖即算子圖： 缺少權限、效果、失敗與證據邊。
- 語法閉包冒充治理閉包： 能組合與編譯便宣稱可安全執行。
- 抽象隱藏責任： 封裝同時隱藏不可逆效果與失敗所有者。
- AI 依名稱選工具： 不檢查算子簽章與當前世界狀態。
- 物理命令僭位： 命令送出被當成物理效果成立。
- 算子版本無身份： 語意改變卻沿用舊名稱與契約。

- 多投影身份分裂： 文字、圖形與機器節點無法確認是否為同一算子。

SECTION

30.1 表面符號—算子身份錯配

統計同名、同字元或同圖示在不同上下文中實際對應不同算子的比例。

SECTION

30.2 算子簽章完整度

對八元算子結構評分：

> C_ω
> =
> (|declared operator dimensions|)/(8)

研究其與事故率、重構成本、AI 工具誤用率的關係。

SECTION

30.3 組合合法性檢查

比較只做型別檢查，以及同時檢查條件、效果、權限與證據的系統，對錯誤組合的檢出能力。

SECTION

30.4 效果簽章完整率

統計高風險函式中，有多少能明示：

- 外部效果；
- 不可逆性；
- 權限；

- 冪等；
- 補償；
- 證據。

SECTION

30.5 代數律有效域

對實際算子測量結合、交換、冪等與分配性成立的條件與失效範圍。

SECTION

30.6 可逆性推導

比較人工判斷與結構化算子簽章，對複合工作流可逆、可補償與不可逆的判定準確率。

SECTION

30.7 閉包保持率

檢查複合算子是否仍能推導：

- 型別；
- 條件；
- 效果；
- 失敗；
- 權限；
- 證據；
- 恢復。

SECTION

30.8 語法閉包—治理閉包差異

統計可編譯工作流中，有多少缺少合法授權、責任或恢復結構。

SECTION

30.9 算子圖事故預防

比較呼叫圖與完整算子圖，在發現越權、不可逆效果、失敗中斷與無擁有者責任上的能力。

SECTION

30.10 AI 算子選擇

比較 AI 只依名稱與描述選工具，以及依完整算子簽章與世界狀態選擇工具的正確率。

SECTION

30.11 高階算子安全性

測量 retry、parallelize、cache、authorize、audit 等高階算子是否正確保持原算子的效果與失敗語意。

SECTION

30.12 治理算子可見性

統計批准、撤銷、申訴、暫停與規則修改在系統中有多少被正式建模，而非隱藏於人工作業。

SECTION

30.13 跨投影算子同一性

將同一算子投影為文字、圖形、語音與機器節點，測量身份、語意與版本保持。

SECTION

30.14 物理算子驗證

比較只記錄命令送出與同時記錄感測確認、安全不變量及物理證據的控制系統。

- 符號不等於字元或 token。
- 可執行符號是具有受治理語意的算子。
- 算子至少包含輸入、輸出、前置、後置、效果、授權、驗證與組合。
- 同一算子可以具有文字、圖形、語音與機器等多種表面投影。
- 表面符號相同不保證算子身份相同。
- 算子身份應由語意角色、型別、效果、不變量與版本共同決定。
- 程式語言需要值、結構、狀態、事件、效果與治理等多類算子。
- 傳統語言高度形式化值算子，卻常隱藏效果與治理算子。
- 型別簽章不足以描述世界算子。
- 算子組合需要輸出—輸入、前後條件、效果、授權與證據同時相容。

11.

> TypeCompatible
> not⇒
> WorldCompatible

- 純算子與效果算子應在語言中明確區分。
- 交換、結合、冪等與分配等代數律只在明示條件域內成立。
- 數位可逆、可補償與世界不可逆是不同性質。
- 時間、等待、超時與結果未知應成為正式算子結構。
- 失敗集合與失敗傳播應能由複合算子推導。

- 治理算子是作用於權限、規則與其他算子的高階算子。
- 算子圖比呼叫圖多保存效果、權限、失敗與證據。
- 語法閉包、語意閉包、操作閉包與治理閉包互不等同。
- 抽象可以隱藏實作，但不能隱藏外部責任與不可逆效果。
- AI 應根據算子簽章與世界狀態選擇工具，而非只依名稱相似度。

22.

> 【程式語言中的符號，】

> 【不是靜止地代表某個意思，】

> 【而是在明示條件與責任下，】

> 【對可計算世界施加可組合變換。】

SECTION

32.1 承接 PU-2-01

PU-2-01 將文字降為程式結構的一種投影。

本篇進一步把表面符號與權威算子分開，說明同一算子可以擁有多種投影。

SECTION

32.2 承接第 1 冊

第 1 冊建立了狀態、事件、責任、契約、失敗與恢復。

本篇把這些結構納入算子簽章，使語言中的操作不再只是值轉換，而能承載世界責任。

SECTION

32.3 銜接 PU-2-03

下一篇將建立：

> 【Syntax
> →
> Semantics
> →
> Effects】

並回答：

- 語法如何形成合法算子結構；
- 語意如何把結構綁定到問題世界；
- 效果如何描述實際世界差分；
- 型別與效果系統如何共同驗證；
- 相同語法為何可能具有不同語意與效果。

  人類看到符號時，容易先注意它的外觀。

  機器處理符號時，容易先辨認它的 token 與語法位置。

  但對真正的程式語言而言，符號的核心不在於它長什麼樣，而在於它能對什麼施加何種合法變換。

  因此：

> 【Symbol Identity
> ≠
> Visual Identity】

符號需要被提升為算子，並保存：

- 輸入與輸出；

- 前置與後置；
- 世界效果；
- 權限；
- 失敗；
- 證據；
- 組合規則；
- 版本與責任。

  這會使程式語言從「操作文字」進一步轉化為「組合世界變換」。

  本文將算子語言收束為：

> 【Operator Language
> =
> Typed Transformation
> +
> World Preconditions
> +
> Effect Signature
> +
> Authorization
> +
> Evidence
> +
> Composition Law】

當多個算子被組合時，真正需要保持的不只是型別，也包括：

- 語意；
- 效果；
- 權限；

- 失敗；
- 恢復；
- 證據。

因此，計算閉包也不應只表示「還能形成更多合法式子」，而應表示：

> 由一組基礎算子生成的所有複合結構，仍可被理解、執行、驗證、追責與恢復。

本文最終命題是：

> 【語言的力量，】
>
> 【不只來自它擁有多少符號，】
>
> 【而來自它能否讓符號在組合後，】
>
> 【仍保存世界語意與責任。】

```
operator:
 operator_id: "send-message-v2"
 semantic_role: "external-message-delivery"
 input:
  - "recipient"
  - "message-content"
 output:
  - "delivery-attempt"
 preconditions:
  - "recipient is resolved"
  - "content policy passed"
 postconditions:
  - "delivery attempt is recorded"
 effects:
  external:
   - "message may become visible to recipient"
  irreversible: true
 authorization:
  required:
   - "message-send-capability"
 verification:
```

```
    evidence:
     - "provider receipt"
   composition:
    retry:
     safe: "only-with-same-action-id"
    compensation:
     available: false
composition_check:
 first: "AuthorizePayment"
 second: "MarkOrderPaid"
 compatibility:
  output_input: true
  post_precondition: true
  effects: true
  authorization: true
  evidence_trace: true
 composed_effects:
  - "external-bank-authorization"
  - "order-state-write"
 failure_union:
  - "PaymentDeclined"
  - "PaymentOutcomeUnknown"
  - "OrderVersionConflict"
 reversible: false
 compensatable: true
closure:
 base_operators:
  - "ValidateOrder"
  - "AuthorizePayment"
  - "RecordPayment"
  - "NotifyCustomer"
 levels:
  syntactic:
   closed: true
  semantic:
   closed: true
  operational:
   closed: true
  governed:
   closed: false
   missing:
    - "human approval for high-value payment"
    - "compensation owner for duplicate charge"
operator_node:
 id: "DeployRelease"
```

```
class: "effect-operator"
signature:
  input: "ApprovedRelease"
  output: "DeploymentState"
effects:
  - "writes deployment state"
  - "calls infrastructure"
  - "may shift physical compute resources"
failures:
  - "DeploymentRejected"
  - "DeploymentPartial"
  - "OutcomeUnknown"
governance:
  authorization: "production-deploy"
  approval_required: true
recovery:
  rollback: "conditional"
  compensation: "traffic-shift-back"
```

- PU-2-01　程式語言不等於文字：結構、語意與執行的基本分離
- PU-2-02　符號作為算子：從靜態字元到可組合計算閉包
- PU-2-03　語法—語意—效果：程式語言的三層存在結構
- PU-2-04　意圖中介表示：從自然意圖到多重可執行投影
- PU-2-05　可編譯世界：程式執行作為世界狀態差分
- PU-2-06　後文本程式語言：意圖、結構、驗證與物理耦合的統一框架

SECTION

Neo.K／EveMissLab 相關理論

- Neo.K with Aletheia，《程式語言不等於文字：結構、語意與執行的基本分離》，2026。
- Neo.K with Aletheia，《程式不等於程式碼：可執行問題模型的基本定義》，2026。
- Neo.K with Aletheia，《資料—狀態—事件—行動：程式系統的四元動力結構》，2026。
- Neo.K with Aletheia，《責任、模組與契約：從檔案分類到系統邊界》，2026。

- Neo.K with Aletheia，《失敗也是程式：驗證、可觀測、恢復與長期維護》，2026。
- Neo.K with Aletheia，《分域結構論》，2026。
- Neo.K with Aletheia，《算子本體論》，2026。

SECTION

一般理論背景

- Church, A., “An Unsolvable Problem of Elementary Number Theory,” 1936.
- Turing, A. M., “On Computable Numbers, with an Application to the Entscheidungsproblem,” 1936.
- Curry, H. B. and Feys, R., Combinatory Logic, 1958.
- Mac Lane, S., Categories for the Working Mathematician, 1971.
- Scott, D. and Strachey, C., “Toward a Mathematical Semantics for Computer Languages,” 1971.
- Moggi, E., “Notions of Computation and Monads,” 1991.
- Wadler, P., “The Essence of Functional Programming,” 1992.
- Plotkin, G. D. and Power, J., “Algebraic Operations and Generic Effects,” 2003.
- Pierce, B. C., Types and Programming Languages, 2002.
- Hoare, C. A. R., “An Axiomatic Basis for Computer Programming,” 1969.
- Milner, R., Communicating and Mobile Systems: The Pi-Calculus, 1999.
- Reynolds, J. C., Theories of Programming Languages, 1998.
- Backus, J., “Can Programming Be Liberated from the von Neumann Style?”, 1978.

SECTION

v0.1 — 2026-07-27

- 完成十八篇地基論文第十四篇。
- 將可執行符號定義為具有受治理語意的算子。
- 建立算子八元模型。
- 區分表面符號、語意符號與算子符號。
- 建立算子身份與多表面投影。
- 建立值、結構、狀態、事件、效果與治理六類算子。
- 擴展輸入輸出簽章為前置、後置、效果、授權與驗證簽章。
- 形式化完整算子組合判準。
- 區分型別相容與世界相容。
- 分析純度、代數律、可逆性、冪等性與時序。
- 建立失敗算子、治理算子與高階算子。
- 建立算子圖及多尺度摘要。
- 區分語法、語意、操作與治理四層計算閉包。
- 建立閉包保持條件與責任保留抽象。
- 加入算術、資料轉換、狀態、工作流、AI Agent、醫療與物理裝置案例。
- 提出二十二類失敗模式與十四項可證偽研究方向。
- 完成與 PU-2-03《語法—語意—效果》的銜接。