後文本程式語言：意圖、結構、驗證與物理耦合的統一框架

Post-Textual Programming Languages: A Unified Framework for Intent, Structure, Verification, and Physical Coupling

v0.1 / 2026-07-27 / Neo.K with Aletheia / 初版完成／十八篇地基系列封頂

https://thisoneisneok.com/html/papers/pu-2-06.html

SECTION

Post-Textual Programming Languages: A Unified Framework for Intent, Structure, Verification, and Physical Coupling

論文編號： PU-2-06

系列：「程式宇宙書系」第 2 冊《程式語言的本質》

系列位置： 第十八篇／十八篇地基論文封頂篇

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

本篇是第 2 冊《程式語言的本質》的封頂篇，也是「程式宇宙書系」十八篇地基論文的總封頂篇。

前十七篇依序建立了三重宇宙、表示落差、數位存在域、生成與限制、計算目的、可執行問題模型、問題世界、資料—狀態—事件—行動、責任模組、外部結構記憶、失敗與恢復、非文本語言本體、符號算子、語法—語意—效果、意圖中介表示與可編譯世界。

本篇的任務不是再提出一個孤立概念，而是把前述結構統合成一種新的程式語言定義。

本文提出：

> 【Post-Textual
> ≠
> Textless】

後文本程式語言並不是取消文字，而是取消文字對程式本體的壟斷。

文字仍可以是高效率、精確、可搜尋、可版本控制的編輯與交換介面；但程式的權威存在不再被限制為單一字元序列，而是由可識別、可組合、可驗證、可投影、可執行並可追蹤至世界效果的語意結構構成。

本文將後文本程式語言定義為：

> 【mathbb L_(PT)
> =
> <
> I,
> U,
> W,
> Ω,
> S,
> M,
> E,
> P,
> V,
> Π,
> R,
> H
> >】

其中：

- I：Intent，原始意圖、目的與成功條件；
- U：Uncertainty，歧義、未知、爭議與不可代理選擇；

- W：World，問題世界、實體、狀態、規則與邊界；
- Ω：Operators，具有輸入、輸出、前後條件、效果與治理的算子；
- S：Structure，可形成、可組合並具有身份的語意結構；
- M：Meaning，世界綁定、規範位置與語意映射；
- E：Effects，數位、制度、外部與物理效果；
- P：Permissions，權限、委任、批准與不可代理決策；
- V：Verification，型別、不變量、證據、完成與接受條件；
- Π：Projections，文字、視覺、表格、UI、工作流、契約、測試與文件投影；
- R：Runtime，世界差分提交、觀測、恢復與核對；
- H：History，來源意圖、版本、決策、執行、事故與殘差歷史。

本文核心命題是：

> 【Programming Language
> =
> Intent-Bearing World Structure
> +
> Governed Executable Transformation】

程式語言不只是一組語法規則，也不只是從來源碼到機器碼的轉譯系統。它應能保存：

- 人類希望達成什麼；
- 哪些內容仍未決；
- 程式指向哪個問題世界；
- 哪些算子能改變哪些狀態；
- 誰有權要求與批准；
- 哪些效果可逆、可補償或不可逆；

- 什麼證據代表完成；
- Runtime 真正觀測到什麼；
- 預測與實際世界差分之間仍有哪些殘差。

本文提出後文本語言的七層統一架構：

> 【Intent
> →
> SemanticIR
> →
> OperatorGraph
> →
> EffectPlan
> →
> VerifiedProjections
> →
> WorldRuntime
> →
> ObservedEvidence】

並形成閉環：

> ObservedEvidence
> →
> Reconciliation
> →
> SemanticIR'
> →
> Intent'

因此，後文本編譯不是單向翻譯，而是一個人類意圖、AI 結構化、編譯驗證、Runtime 執行與世界證據共同參與的持續循環。

本文提出「語言核心與投影分離」：

> 【CoreProgram
> ≠
> TextProjection】

核心程式是一個具有穩定節點身份、世界語意、算子簽章、權限、效果與歷史的圖結構；文字、視覺圖、UI、API、工作流與測試只是面向不同角色與任務的投影。

同一核心程式可生成：

> Π(CoreProgram)
> =
> {
> Text,
> Visual,
> UI,
> Workflow,
> Contract,
> Test,
> Documentation,
> RuntimePlan,
> AuditView
> }

各投影外觀不必相同，但必須保持共同的：

- 世界身份；
- 狀態語意；
- 算子作用；
- 效果；
- 權限；
- 失敗；
- 完成條件；

- 版本與來源。

  本文提出「後文本型別判定」：

> 【Γ;
> W;
> P;
> H
> |-
> n
> :
> T
> !
> ε
> triangleright
> Q
> dashv
> O】

其中：

- n：語意節點或算子；
- T：值、狀態、事件、證據或世界差分型別；
- ε：效果簽章；
- Q：後置條件與完成義務；
- O：觀測、稽核與殘差義務；
- H：歷史與來源環境。

  此判定不只回答「這段程式型別是否正確」，還回答：

- 它指向何種世界；
- 需要哪些權限；

- 可能造成哪些效果；
- 是否能被安全組合；
- 如何驗證；
- 失敗如何處理；
- 執行後必須觀測什麼。

  本文也正式區分「編輯權、生成權、批准權與執行權」。任何主體可以提出或修改候選意圖；AI 可以生成候選結構與投影；驗證器可以判斷結構與效果合法性；但只有具合法委任與批准的主體才能取得世界差分提交權。

  形式上：

> 【CanEdit
> not⇒
> CanGenerate
> not⇒
> CanApprove
> not⇒
> CanExecute】

本文進一步提出「語意版本控制」。傳統版本控制比較文字行；後文本版本控制比較：

- 目的；
- 世界範圍；
- 實體身份；
- 狀態轉移；
- 規則；
- 效果；
- 權限；
- 不可逆性；

- 證據；
- 受影響主體。

  因此，一個只有三行文字變更的修改，也可能是重大世界差分；而數百行格式化或生成碼變更，可能沒有語意變化。

  本文也提出「後文本 Runtime」。它不只是執行虛擬機，而是：

> 【PostTextRuntime
> =
> WorldDifferenceCommitter
> +
> EffectObserver
> +
> PermissionEnforcer
> +
> EvidenceCollector
> +
> ResidualReconciler】

它接收經驗證的世界差分計畫，重新檢查世界版本、權限與安全守衛，分階段提交數位、制度與物理效果，收集外部回執與感測證據，並將實際差分與預測差分核對。

本文特別處理物理耦合。對物理設備、機器人、能源、醫療與生產系統而言，後文本語言必須正式區分：

- 命令；
- 控制器接受；
- 物理效果；
- 感測確認；
- 人類接受；
- 不可逆殘差。

  因此：

> 【Command
> ≠
> PhysicalEffect
> ≠
> VerifiedPhysicalState】

本文使用網站發布、資料系統、AI Agent、多角色研究平台、醫療工作流、自治設備與跨制度批准等案例，展示後文本統一框架如何把自然意圖、語意結構、多投影、驗證、Runtime 與世界證據連接起來。

本文最後提出可證偽研究綱領，包括：多投影語意保持、意圖—執行偏差、權限分離、效果預覽準確率、語意版本控制、AI 生成事故率、Runtime 差分核對、不可代理選擇保留、物理耦合確認、結構化程式教學與後文本開發效率。

本文將十八篇地基論文收束為一個總命題：

> 【Program
> =
> Intent
> +
> World Model
> +
> Governed Operators
> +
> Executable Difference
> +
> Observed Evidence
> +
> Historical Responsibility】

程式設計的未來，不是人類停止寫程式，也不是 AI 直接替人類決定世界。它是人類、AI、編譯器與 Runtime 各自承擔不同責任，共同把意圖轉化為可理解、可選擇、可驗證、可授權、可恢復且能誠實面對未知的世界變更結構。

關鍵詞： 後文本程式語言、意圖編譯、語意 IR、多重投影、世界差分、物理耦合、AI 程式設計、語意版本控制

SECTION

Abstract

This paper concludes the eighteen-paper foundation series by introducing a unified framework for post-textual programming languages.

Post-textual does not mean textless. It means that text no longer monopolizes the ontology of programs.

A post-textual programming language preserves intent, uncertainty, problem-world structure, governed operators, semantics, effects, permissions, verification, multiple projections, runtime execution, observation, and history.

The paper proposes a seven-layer architecture from intent to observed evidence, a post-textual typing judgment, semantic version control, separation of editing, generation, approval, and execution rights, and a runtime model for governed world-difference commitment and physical coupling.

Keywords: post-textual programming, intent compilation, semantic intermediate representation, multiple projections, world difference, physical coupling

「後文本」容易被誤解為視覺程式設計、語音程式設計，或完全取消來源碼。

本文所指並非如此。

後文本的真正含義是：

> 文字不再是程式唯一能被編輯、理解、版本化與執行的權威存在形式。

文字仍然可以存在，甚至在許多任務中仍然是最佳投影。

但以下內容不應永遠被壓縮進文字與作者腦中：

- 問題世界身份；
- 狀態生命週期；
- 算子效果；
- 權限；
- 外部證據；

- 不可逆性；
- 受影響主體；
- 失敗與補償；
- 來源意圖；
- 多角色理解。

  後文本語言要做的，不是摧毀文字，而是把文字放回它應有的位置：一種投影，而非全部本體。

  本文定義：

> 【mathbb L_(PT)
> =
> <
> I,
> U,
> W,
> Ω,
> S,
> M,
> E,
> P,
> V,
> Π,
> R,
> H
> >】

SECTION

2.1 意圖 I

保存目的、成功條件、非目標、停止條件與可接受殘差。

SECTION

2.2 不確定性 U

保存歧義、未知、爭議、假設與不可代理選擇。

SECTION

2.3 問題世界 W

保存實體、關係、狀態、規則、角色、邊界與外部性。

SECTION

2.4 算子集合 Ω

保存值、結構、狀態、事件、效果與治理算子。

SECTION

2.5 結構 S

保存合法節點、關係、作用域、圖與組合形式。

SECTION

2.6 語意 M

保存世界綁定、規範位置、身份與語意映射。

SECTION

2.7 效果 E

保存數位、外部、制度、物理與治理效果。

SECTION

2.8 權限 P

保存能力、作用域、委任、批准、撤銷與執行權。

SECTION

2.9 驗證 V

保存型別、不變量、測試、證據、完成與接受判準。

SECTION

2.10 投影 Π

保存文字、視覺、UI、工作流、契約、測試、文件與稽核視圖。

SECTION

2.11 Runtime R

負責差分提交、觀測、恢復與殘差核對。

SECTION

2.12 歷史 H

保存來源、版本、決策、生成、批准、執行、事故與變更。

本文提出：

> 【Intent
> →
> SemanticIR
> →
> OperatorGraph
> →

SECTION

3.1 意圖層

接收：

- 自然語言；
- 對話；
- 圖形；
- 表格；
- 操作示範；
- 既有程式與文件。

  此層允許模糊、未完成與多種候選。

SECTION

3.2 語意 IR 層

把意圖轉換為：

- 穩定世界身份；
- 目的；
- 狀態；
- 規則；
- 主體；
- 未知；
- 決策節點；
- 完成條件。

SECTION

3.3 算子圖層

把世界變更分解成可組合算子，並保存：

- 輸入與輸出；
- 前置與後置；
- 效果；
- 權限；
- 失敗；
- 證據；
- 組合條件。

SECTION

3.4 效果計畫層

預測：

- 狀態差分；
- 外部呼叫；
- 制度效果；
- 物理效果；
- 受影響主體；
- 不可逆節點；
- 補償與殘差。

SECTION

3.5 驗證投影層

生成並核查：

- 程式碼；
- UI；
- 契約；
- 工作流；
- 測試；
- 文件；
- 稽核規則；
- Runtime 計畫。

SECTION

3.6 世界 Runtime 層

重新檢查世界版本與權限，分階段提交差分。

SECTION

3.7 觀測證據層

收集真實事件、回執、感測、人類回應與殘差。

傳統編譯：

> Source
> →
> Executable

後文本編譯：

> Intent
> →
> Structure
> →
> Execution
> →
> Evidence
> →
> Revision

SECTION

4.1 Runtime 回饋

若觀測效果與宣告不同：

> ObservedEffects
> ≠
> DeclaredEffects

則應更新：

- 效果簽章；
- 世界模型；
- 驗證；
- 文件；
- 投影；
- 後續計畫。

SECTION

4.2 意圖修訂

世界證據也可能讓使用者重新思考目的。

因此：

> ObservedEvidence
> →
> Intent'

不是錯誤，而是意圖—世界耦合的正常循環。

SECTION

4.3 編譯不再一次完成

程式在長期生命中持續：

- 重新綁定；
- 重新驗證；
- 遷移；
- 觀測；
- 學習；
- 退役。

本文定義：

> 【CoreProgram
> =
> <
> Graph,
> Identity,
> Semantics,

SECTION

5.1 核心圖

節點表示：

- 實體；
- 狀態；
- 行動；
- 事件；
- 規則；
- 決策；
- 證據；
- 未知。

邊表示：

- 組合；
- 因果；
- 擁有；
- 依賴；
- 授權；
- 失敗；
- 追蹤。

SECTION

5.2 穩定身份

節點身份不因：

- 重新格式化；
- 移動檔案；
- 改變語言；
- 改變視圖位置；

  而消失。

SECTION

5.3 投影函數

> P⊠
> =
> π⊠
> (
> CoreProgram
> mid
> Role,
> Task,
> Scale,
> Platform
>)

SECTION

5.4 投影不是副本真相

任何投影若被修改，必須：

- 回寫核心；
- 標示平台特定補充；

- 或被視為非權威草稿。

  後文本語言可使用：

- 文字；
- 視覺圖；
- 表格；
- 表單；
- 自然語言；
- 語音；
- 示範；
- 直接操作；
- 時間軸；
- 狀態機；
- 三維或空間界面。

SECTION

6.1 模態不決定本體

視覺介面不自動是後文本。

若視覺方塊只是另一種字串，而語意仍靠人腦與隱藏程式碼推導，它仍未建立後文本本體。

SECTION

6.2 任務適配

- 文字適合精確局部表達；
- 圖形適合關係與流程；

- 表格適合大量規則與映射；
- 對話適合意圖與澄清；
- 時間軸適合事件與版本；
- 空間視圖適合物理設備與數位孿生。

SECTION

6.3 無障礙與替代投影

同一核心結構應允許不同感知與操作方式，而不是把程式設計能力綁定單一視覺或文字介面。

後文本語言仍有語法，但語法不必只是一維字串文法。

SECTION

7.1 圖語法

定義：

- 可用節點；
- 可用邊；
- 嵌套；
- 作用域；
- 端口；
- 合法拓撲。

SECTION

7.2 表格語法

定義欄位、關係、優先級、例外與衝突。

SECTION

7.3 對話語法

定義意圖、澄清、確認、撤回與批准行為。

SECTION

7.4 未完成語法

正式允許：

hole
unknown
ambiguous
contested
human-choice
external-evidence

SECTION

7.5 語法守衛

高風險節點若缺少：

- 主體；
- 權限；
- 效果；
- 證據；
- 恢復；

  則不能進入可執行結構。

  本文提出：

> 【Γ;
> W;
> P;
> H

SECTION

8.1 世界環境 W

節點綁定的實體、狀態、規則與版本。

SECTION

8.2 權限環境 P

誰可編輯、生成、批准、執行與觀測。

SECTION

8.3 歷史環境 H

來源意圖、過去決策、承諾、事故與遷移。

SECTION

8.4 型別 T

可包含：

- Value；
- Entity；
- State；
- Event；
- Action；
- Evidence；
- Decision；
- WorldDifference。

SECTION

8.5 效果 ε

描述：

- 讀；
- 寫；
- 事件；
- 外部呼叫；
- 制度；
- 物理；
- 治理；
- 不可逆性。

SECTION

8.6 後置義務 Q

執行成功後必須成立的世界條件。

SECTION

8.7 觀測義務 O

提交後必須取得的回執、感測、稽核、接受與殘差資訊。

傳統型別描述資料結構。

後文本語言需要世界型別。

SECTION

9.1 身份型別

區分具有相同資料形狀、但世界身份不同的對象。

SECTION

9.2 狀態型別

例如：

Order<PaymentPending>
Order<Paid>
Order<Cancelled>
合法算子只能接受相容狀態。

SECTION

9.3 證據型別

例如：

ProviderReceipt<Authorized>
SensorEvidence<DoorUnlocked>
HumanApproval<ProductionDeploy>

SECTION

9.4 權限型別

Capability<Deploy, Production, Until(t)>

SECTION

9.5 差分型別

描述候選變更的作用域、可逆性與受影響主體。

SECTION

9.6 未知型別

Unknown 不是空值，而是帶有下一步要求的正式世界型別。

本文提出：

> 【CanEdit
> not⇒
> CanGenerate
> not⇒
> CanApprove
> not⇒
> CanExecute】

SECTION

10.1 編輯權

可以提出、修改或分支候選意圖與結構。

SECTION

10.2 生成權

可以由核心結構生成程式碼、UI、工作流或 Runtime 計畫。

SECTION

10.3 批准權

可以讓某一高風險候選結構取得提交資格。

SECTION

10.4 執行權

可以在特定世界、時間、作用域與版本中實際提交差分。

SECTION

10.5 觀測權

某些證據包含敏感資訊，觀測與稽核也需最小權限。

SECTION

10.6 撤銷權

必須知道誰能撤銷：

- 委任；
- 批准；
- 執行排程；
- 投影發布；
- Runtime 能力。

SECTION

10.7 權限不能由介面推導

使用者看得到按鈕，不代表有合法權限。

Agent 能呼叫工具，也不代表本次任務已委任。

後文本語言必須把不可代理選擇建模為語言節點，而不是 UI 外部的倫理提醒。

SECTION

11.1 判定條件

若選擇可能永久改變：

- 身份；
- 財產；
- 權利；
- 公開狀態；
- 身體；
- 法律承諾；
- 長期委任；

則應檢查是否屬於特定主體保留的選擇。

SECTION

11.2 正式表示

> NonDelegable(c,a)

表示選擇 c 必須由主體 a 完成。

SECTION

11.3 AI 的角色

AI 可：

- 整理候選；
- 解釋後果；

- 模擬差分；
- 找出衝突；
- 建議。

  但不得把預測偏好冒充已作選擇。

SECTION

11.4 Runtime 守衛

不可代理節點未完成時，執行圖在該處停止。

傳統版本控制主要比較文字：

line added
line removed
line modified
後文本版本控制比較：

> Δ_(semantic)
> =
> <
> Δ I,
> Δ W,
> Δ S,
> Δ R,
> Δ E,
> Δ P,
> Δ V,
> Δ A
> >

其中：

- Δ I：目的變更；
- Δ W：世界範圍變更；

- ΔS：狀態與生命週期變更；
- ΔR：規則變更；
- ΔE：效果變更；
- ΔP：權限變更；
- ΔV：驗證與證據變更；
- ΔA：受影響主體變更。

SECTION

12.1 重大變更不由行數判定

一行：

requires_approval = false
可能比一千行格式調整更重大。

SECTION

12.2 語意合併

分支合併需判斷：

- 目的是否衝突；
- 狀態機是否仍合法；
- 效果是否擴張；
- 權限是否提升；
- 證據是否仍充分。

SECTION

12.3 歷史身份

節點重命名、搬移或投影改變時，身份仍需保持。

SECTION

12.4 生成物差分

大量生成碼差分可被折疊為其來源語意節點的少量變化。

SECTION

13.1 投影族

> Π(Core)
> =
> {
> P□,P□,…,P□
> }

SECTION

13.2 一致性

> Consistent(P□,P□mid Core)

並不要求兩者顯示相同資訊，而是要求重要語意不矛盾。

SECTION

13.3 可回寫投影

文字、視覺與表格可作為編輯投影，修改後應產生核心結構變更提案。

SECTION

13.4 唯讀投影

文件、監控與稽核視圖可以是唯讀衍生物。

SECTION

13.5 平台特定補充

某些實作細節只屬於特定後端，可附加在核心節點上，但不能靜默改變核心效果。

SECTION

13.6 投影漂移

若投影與核心不一致，應標記：

stale
diverged
unverified
platform-extension
runtime-observed

SECTION

14.1 人類

主要負責：

- 目的；
- 價值；
- 重要邊界；
- 不可代理選擇；
- 風險接受；
- 最終制度責任。

SECTION

14.2 AI

主要負責：

- 展開意圖；
- 建立候選模型；
- 找出歧義；
- 生成投影；
- 模擬效果；
- 提供解釋；
- 提醒缺失。

SECTION

14.3 編譯器與驗證器

主要負責：

- 結構；
- 型別；
- 狀態；
- 算子組合；
- 效果；
- 權限；
- 世界版本；
- 投影一致性。

SECTION

14.4 Runtime

主要負責：

- 最終守衛；
- 分階段提交；
- 外部工具調用；
- 感測與回執；
- 失敗隔離；
- 恢復；
- 核對。

SECTION

14.5 責任不能被模型融合抹除

即使單一 AI 系統同時進行理解、生成與執行，內部仍應保存上述責任階段，避免「模型認為可以」直接變成「世界已被修改」。

一個完整工具鏈至少包含：

SECTION

15.1 Intent Capture

接收自然語言、圖、表、示範與舊程式。

SECTION

15.2 Ambiguity and Decision Manager

保存歧義、未知、爭議與人類決策節點。

SECTION

15.3 World Model Registry

保存實體、狀態、規則、責任與版本。

SECTION

15.4 Semantic IR Store

保存核心語意圖與穩定身份。

SECTION

15.5 Operator Registry

保存算子簽章、效果、權限、失敗與恢復。

SECTION

15.6 Validator

執行型別、狀態、不變量、效果、權限與證據檢查。

SECTION

15.7 Projection Generator

生成文字、視覺、UI、工作流、契約、測試與文件。

SECTION

15.8 Effect Planner

建立預測世界差分與恢復計畫。

SECTION

15.9 World Runtime

提交合法世界差分。

SECTION

15.10 Observer and Reconciler

核對預測與實際效果。

SECTION

15.11 History Ledger

保存來源、決策、批准、提交、事故與殘差。

本文定義：

> 【PostTextRuntime
> =
> WorldDifferenceCommitter
> +
> EffectObserver
> +
> PermissionEnforcer
> +
> EvidenceCollector
> +
> ResidualReconciler】

SECTION

16.1 接收的不是任意程式碼

Runtime 應接收：

- 已驗證計畫；
- 世界版本；

- 權限能力；
- 效果簽章；
- 守衛；
- 觀測義務；
- 恢復路徑。

SECTION

16.2 執行前再驗證

由於世界會改變，高風險操作不能只依編譯時驗證。

SECTION

16.3 分階段提交

可使用：

prepare
approve
commit
observe
finalize

SECTION

16.4 執行偏離

若世界或 Runtime 偏離預測，Runtime 必須：

- 停止；
- 降級；
- 重新編譯；
- 補償；

- 請求人類。

SECTION

16.5 Runtime 不是最後真理

Runtime 的回覆仍只是證據來源，外部世界效果可能需要再次確認。

SECTION

17.1 三層物理行動

> Command
> →
> ControllerAcceptance
> →
> PhysicalEffect
> →
> VerifiedPhysicalState

SECTION

17.2 不可等同

> 【Command
> ≠
> PhysicalEffect
> ≠
> VerifiedPhysicalState】

SECTION

17.3 感測語意

感測器輸出需要：

- 身份；
- 校準；
- 時間；
- 信心；
- 適用範圍；
- 故障狀態。

SECTION

17.4 安全不變量

物理算子需明示：

- 安全區；
- 最大能量；
- 緊急停止；
- 人員位置；
- 設備狀態；
- 時間窗口。

SECTION

17.5 數位孿生

數位孿生可作為物理世界投影，但：

> DigitalTwin
> ≠
> PhysicalWorld

其差異必須持續由感測與核對更新。

SECTION

17.6 物理殘差

即使控制被逆轉，也可能保留能耗、磨損、位置、時間與人員影響。

原始意圖：

把新版網站發布出去。

SECTION

18.1 意圖層

需要知道：

- 哪個網站；
- 哪個版本；
- 哪個環境；
- 成功條件；
- 是否允許自動切流量。

SECTION

18.2 語意 IR

綁定：

- Site；
- Release；
- Deployment；
- PublicTraffic；

- Approval；
- Verification。

SECTION

18.3 算子圖

ValidateRelease
→ BuildArtifact
→ Deploy
→ Verify
→ ShiftTraffic
→ ObservePublicJourney

SECTION

18.4 投影

同一核心生成：

- 部署程式碼；
- 發布工作流；
- 批准介面；
- 測試；
- 回滾計畫；
- 發布文件；
- 稽核紀錄。

SECTION

18.5 世界耦合

部署成功不等於公開完成；公開完成也不能撤回外部快取與既有閱讀。

意圖：

整理本月的營收。

SECTION

19.1 歧義

需要確定：

- 時區；
- 下單、付款或結算；
- 退款；
- 稅；
- 幣別；
- 資料新鮮度。

SECTION

19.2 世界型別

Money<Currency>
Revenue<RecognitionRule, Period>
Evidence<LedgerSnapshot>

SECTION

19.3 多投影

可生成：

- SQL；
- 資料轉換；
- 試算表；
- 圖表；

- 管理摘要；
- 資料來源說明；
- 稽核核對。

SECTION

19.4 語意版本

會計規則改變時，即使 SQL 只有少量變動，也應標示為重大語意版本。

意圖：

整理研究、修改論文並發布。

SECTION

20.1 責任分解

- 搜尋與讀取；
- 證據評估；
- 論文修改；
- 格式檢查；
- 發布；
- 索引更新。

SECTION

20.2 權限分離

讀取、寫草稿、覆寫正式文件、發布與通知必須是不同能力。

SECTION

20.3 人類決策

AI 可以提出新斷言，但關鍵主張與正式發布可以保留人類批准。

SECTION

20.4 Runtime 守衛

發布前重新檢查：

- 文件版本；
- 引用；
- 敏感內容；
- 發布權限；
- 網站狀態。

SECTION

20.5 執行偏離

若發布失敗，Agent 不應回滾已完成研究修改；這是兩個不同世界責任。

SECTION

21.1 核心程式

保存：

- 研究問題；
- 定義；
- 命題；
- 證據；
- 實驗；

- 反例；
- 論文；
- 發布。

SECTION

21.2 角色投影

研究者看到依賴與證據。

讀者看到論文與限制。

維護者看到版本與 Runtime。

稽核者看到來源與主張狀態。

AI Agent 看到機器可讀算子圖。

SECTION

21.3 狀態語意

proposed
open
partially-supported
contradicted
verified
published
withdrawn

SECTION

21.4 關鍵區分

published 不等於 verified。

文件完成不等於命題成立。

SECTION

21.5 長期歷史

失敗實驗與被推翻路徑仍應保留，避免後續 Agent 重複走入相同死路。

意圖：

提醒病人服藥，必要時通知醫師。

SECTION

22.1 世界綁定

- 病人；
- 處方；
- 醫囑版本；
- 時區；
- 停藥狀態；
- 聯絡同意。

SECTION

22.2 決策節點

何時通知醫師、是否通知家屬，可能需要病人或專業制度選擇。

SECTION

22.3 完成層級

scheduled
message-accepted
delivered
read
patient-confirmed

clinically-reviewed

SECTION

22.4 權限與隱私

不同投影只能顯示完成角色任務所需的最少資訊。

SECTION

22.5 效果限制

提醒系統不能把「病人未確認」直接推論為「病人未服藥」。

意圖：

每天早上自動啟動設備。

SECTION

23.1 時間結構

保存：

- 絕對時間；
- 時區；
- 重複規則；
- 過期策略；
- 暫停；
- 例外日期。

SECTION

23.2 物理守衛

- 維護狀態；

- 人員位置；
- 感測器；
- 能源；
- 緊急停止；
- 最大運行參數。

SECTION

23.3 執行鏈

ScheduleTriggered
→ RevalidateWorld
→ RequestStart
→ ControllerAccepted
→ SensorObserved
→ StableOperationVerified

SECTION

23.4 偏離

若振動或溫度超出預測，Runtime 必須停止後續計畫，而不是因排程要求而堅持完成。

SECTION

23.5 歷史

每次實際執行均需連回排程意圖、批准版本與感測證據。

意圖：

讓這個帳號取得正式管理權。

SECTION

24.1 資料更新不足

把 role=admin 寫入資料庫，不一定構成合法制度授權。

SECTION

24.2 制度結構

需要：

- 授權者；
- 被授權者；
- 能力；
- 作用域；
- 時效；
- 批准程序；
- 撤銷；
- 稽核；
- 申訴。

SECTION

24.3 治理算子

ProposeGrant
→ Review
→ Approve
→ Activate
→ Audit
→ RevokeOrRenew

SECTION

24.4 語意版本控制

若管理權範圍擴大，不能只視為角色名稱不變的內部修改。

一個 MVP 不需要一次實現全部理論，但至少應具備：

- 結構化意圖節點；
- 穩定世界身份；
- 算子登錄；
- 明示效果與權限；
- 未知與人類決策節點；
- 文字與視覺兩種投影；
- 語意差分；
- 執行前效果預覽；
- Runtime 行動身份；
- 執行後證據與核對。

SECTION

25.1 最小核心

可使用圖資料結構或結構化文件保存核心 IR。

SECTION

25.2 初期投影

先支援：

- Markdown／文字；
- YAML／JSON；
- 狀態與算子圖；

- 生成程式碼；
- 驗證報告。

SECTION

25.3 初期 Runtime

先限制於：

- 本地檔案；
- 沙盒程式；
- 可回復部署；
- 低風險 Agent 工具。

SECTION

25.4 後續物理耦合

物理控制必須在數位世界差分、觀測與恢復結構成熟後再逐步接入。

- 後文本等於沒有文字： 為追求新穎而放棄高效率文字工具。
- 視覺化即後文本： 只有方塊與連線，沒有世界語意與穩定身份。
- 自然語言即程式： 一句話直接取得世界修改權。
- 核心仍是生成碼： 所謂 IR 只是暫時生成過程，無權威身份與歷史。
- 多投影多真相： 文字、圖、UI 與工作流可獨立修改而互相漂移。
- 語法節點無世界綁定： 結構合法但不知道指向何種世界。
- 效果隱藏： 高風險作用仍包裝為普通函式或按鈕。
- 權限外掛化： 語言與編譯器無法知道誰能執行。
- 生成權等於執行權： AI 產生候選後直接提交世界。

- 批准權與執行權混合： 單一服務可以自行批准並執行高風險操作。
- 不可代理選擇消失： 模型以高信心推測替代主體決定。
- 未知被空值化： 歧義、爭議與結果未知無法阻止執行。
- 文字 diff 僭位： 重大世界效果變更被視為微小行數修改。
- 生成碼爆炸： 多後端生成產生大量無法理解的差分。
- Runtime 只執行不核對： 工具回傳成功就宣稱世界完成。
- 命令等於物理效果： 缺少感測與安全證據。
- 數位孿生僭位： 模型狀態被當成物理世界本身。
- 沙盒效果洩漏： 網路、費用、權限與物理能力未被真正限制。
- AI 解釋不可追蹤： 無法連回原始意圖、模型版本與來源。
- 世界版本缺席： 舊計畫在新世界中直接執行。
- 投影角色錯配： 使用者被迫閱讀工程圖，工程師只看到簡化 UI。
- 證據義務缺席： 不知道什麼代表完成、失敗或未知。
- 事故未回饋語言： Runtime 偏離與失敗沒有更新算子、效果與驗證。
- 退役無語意： 停止 Runtime 後仍留下權限、資料、投影與外部依賴。
- 平台綁定核心： 語意 IR 被單一框架或供應商格式壟斷。
- 形式化過度： 所有低風險、可逆操作都被龐大治理阻塞。
- 自動化過度： 為效率省略人類決策與正當程序。
- 人類確認過度： 每一低階步驟都要求確認，使重要決策被確認疲勞淹沒。
- AI 單體責任混合： 同一模型內部理解、批准、執行與稽核無階段隔離。
- 物理耦合提前： 尚未建立差分、證據與恢復，就直接控制高風險設備。

SECTION

27.1 多投影語意保持

將同一核心程式投影為文字、視覺、UI、工作流與程式碼，再由不同投影修改與重建，測量：

- 世界身份；
- 狀態；
- 規則；
- 效果；
- 權限；
- 失敗；

  的保持率。

SECTION

27.2 意圖—執行偏差

比較原始意圖、語意 IR、預測差分與實際世界差分：

> ε_(I→ W)
> =
> Distance
> (
> Intent,
> ObservedWorldOutcome
>)

研究偏差主要來自意圖遺失、模型錯誤、投影漂移、Runtime 偏離或外部性。

SECTION

27.3 不確定性保留率

統計原始意圖中的未知、歧義、爭議與人類決策，有多少在生成過程中被正確保存而非靜默填補。

SECTION

27.4 不可代理選擇保留率

檢查涉及身份、財產、權利、醫療、公開與物理世界的決策，是否真正停留到合法主體完成。

SECTION

27.5 四權分離效果

比較編輯、生成、批准與執行權混合及分離系統，在越權、事故與責任判定上的差異。

SECTION

27.6 語意版本控制效益

比較文字 diff 與語意 diff 對：

- 重大效果變更；
- 權限提升；
- 世界範圍擴張；
- 不可逆性；
- 受影響主體；

的檢出率。

SECTION

27.7 生成碼壓縮率

研究以核心語意變更摘要取代大量生成碼差分，是否提高審查準確率並降低認知負荷。

SECTION

27.8 AI 直接執行—分階段編譯差異

比較 AI 直接由自然語言呼叫工具，以及經意圖 IR、效果預覽、批准與 Runtime 守衛後執行的：

- 任務完成率；
- 越權率；
- 重複效果；
- 恢復成本；
- 使用者信任。

SECTION

27.9 效果預覽準確率

比較預測差分與實際差分，並分析哪些效果最難預測：

- 外部網路；
- 人類反應；
- 制度；
- 物理；
- 長期殘差。

SECTION

27.10 Runtime 差分核對率

統計世界提交後，有多少能正式分類為：

- matched；

- partial；
- diverged；
- unknown；
- compensation-required；
- human-acceptance-required。

SECTION

27.11 物理耦合確認率

比較只記錄命令送出與同時保存控制器接受、感測證據、信心和安全守衛的系統。

SECTION

27.12 效果沙盒完整度

測量沙盒對檔案、網路、費用、權限、敏感資料、治理與物理效果的實際攔截率。

SECTION

27.13 角色化投影效益

比較所有角色使用同一來源碼／總圖，以及使用角色與任務特定投影時的理解、決策與錯誤率。

SECTION

27.14 後文本教學

比較語法中心教學與意圖—世界—算子—效果—證據教學，對初學者建立真實系統的能力差異。

SECTION

27.15 結構化 Hole 與 AI 生成

比較普通文字補全與帶有預期型別、世界位置、權限與效果限制的 Hole，在生成正確性與可解釋性上的差異。

SECTION

27.16 Runtime 回饋學習

研究未宣告效果、事故與補償紀錄回饋算子登錄與語意 IR 後，是否降低重複事故。

SECTION

27.17 平台可移植性

將同一核心程式投影到不同語言、雲端、工作流引擎與 Agent Runtime，測量語意與效果保持。

SECTION

27.18 長期維護

比較只有來源碼與文件的系統，以及具有核心語意、歷史、投影與世界日誌的系統，在五年尺度上的：

- 交接；
- 遷移；
- 事故；
- 退役；
- 知識單點。

SECTION

27.19 制度 Runtime

研究批准、申訴、冷靜期、雙重簽署與權限撤銷被正式建模後，是否改善制度一致性與可稽核性。

SECTION

27.20 人機責任可辨識性

測量事故後能否區分：

- 人類目的與選擇；
- AI 推論；
- 編譯器判定；
- Runtime 行動；
- 外部系統效果。
- 後文本不等於沒有文字。
- 後文本的核心是文字不再壟斷程式本體。
- 程式的權威存在應是具有身份、語意、效果、權限、證據與歷史的結構。
- 文字、視覺、UI、工作流、契約與程式碼是核心程式的不同投影。
- 多投影不要求外觀相同，但要求核心世界語意一致。
- 後文本語言必須正式保存意圖與不確定性。
- 不完整、歧義與人類決策節點是合法語言結構。
- 符號只有綁定世界語意、效果與治理後，才成為完整算子。
- 後文本型別系統應包含世界身份、狀態、證據、權限與差分型別。
- 後文本判定必須同時保存效果、後置義務與觀測義務。
- 編輯、生成、批准、執行與觀測權不應被混為同一能力。
- AI 生成候選結構不代表候選已獲批准或可被執行。
- 不可代理選擇必須在語言與 Runtime 中被保留。

- 語意版本控制比字元行數更接近真實世界變更。
- 重大語意差分應顯示目的、範圍、狀態、規則、效果、權限與受影響主體。
- 後文本編譯是從意圖到證據的閉環，而非一次性來源碼翻譯。
- Runtime 觀測到的未宣告效果必須回饋語意核心。
- 後文本 Runtime 是世界差分提交器、效果觀測器與殘差核對器。
- 高風險操作必須在提交前重新驗證世界版本與權限。

20.

> Command
> ≠
> PhysicalEffect
> ≠
> VerifiedPhysicalState

- 數位孿生、模型與模擬不能冒充物理世界。
- 程序沙盒不足以限制完整世界效果。
- 低風險、可逆、局部操作應被高度壓縮；高風險、跨主體與不可逆操作應被展開。
- 後文本語言不應用形式化壓垮人類，而應只在真正重要的邊界增加結構。
- 人類、AI、編譯器與 Runtime 具有不同且不可完全互相取代的責任。
- 程式成功不只需要執行完成，也需要世界效果被觀測、核對與接受。
- 程式的歷史應保存意圖、決策、批准、執行、事故、補償與殘差。

28.

> 【後文本程式語言的目標，】

> 【不是讓機器更快地執行人類說出的每一句話，】

> 【而是讓人類與 AI 能共同把意圖】

> 【轉化為可理解、可選擇、可驗證、】

> 【可授權、可恢復並能對世界後果負責的結構。】

第 2 冊六篇形成以下鏈條。

SECTION

29.1 非文本語言本體

> Programming Language
> ≠
> Text

文字只是可編輯與序列化投影。

SECTION

29.2 符號算子

> Executable Symbol
> =
> Governed Operator

符號具有輸入、輸出、條件、效果、權限、證據與組合規則。

SECTION

29.3 三層存在

> Language
> =
> Syntax

合法形成、世界意義與世界效果互不等同。

SECTION

29.4 意圖中介表示

> Intent
> →
> SemanticIR
> →
> EffectPlan
> →
> Projections

自然意圖先被結構化，而非直接執行。

SECTION

29.5 可編譯世界

> Execution
> =
> Governed World Difference

Runtime 提交、觀測並核對世界差分。

SECTION

29.6 後文本統一

> 【Post-Textual Language
> =
> Intent
> +
> Semantic Structure

SECTION

30.1 第 0 冊：程式之前

第 0 冊回答「為什麼計算」與「計算介入什麼世界」。

其核心是：

> 【I
> ≠
> C
> ≠
> P】

即意圖宇宙、計算機宇宙與物理宇宙不可互相混同。

並建立：

- 三宇宙耦合；
- 表示落差；
- 數位存在域；
- 生成與限制；
- 計算目的與世界選擇。

SECTION

30.2 第 1 冊：系統性程式設計

第 1 冊回答「程式如何成為可理解、可維護的系統」。

其核心是：

> 【Program
> =
> Executable Problem Model】

並建立：

- 問題世界；
- 資料—狀態—事件—行動；
- 責任模組與契約；
- 外部結構記憶；
- 失敗、驗證、恢復與維護。

SECTION

30.3 第 2 冊：程式語言的本質

第 2 冊回答「程式語言究竟表示什麼」。

其核心是：

> 【Language
> =
> Intent-Bearing World Structure
> +
> Governed Executable Transformation】

並建立：

- 非文本語言本體；
- 符號算子；
- 語法—語意—效果；
- 意圖 IR；
- 可編譯世界；
- 後文本統一框架。

SECTION

30.4 三冊總鏈

> 【Why Change the World
> →
> How to Model and Maintain Change
> →
> How Language Represents and Executes Change】

中文可表示為：

> 【想要什麼
> →
> 如何把世界建模
> →
> 如何讓語言合法改變世界】

整個系列最初的問題是：

> 我們到底想要什麼？

十八篇之後，這個問題沒有被簡化成一個最佳化目標。

因為「想要」本身包含：

- 主體；
- 價值；
- 邊界；
- 不確定；
- 他者；
- 權利；

- 時間；
- 可撤回性；
- 世界後果。

程式設計不能只回答：

怎麼做？
它還必須回答：

誰想要？
為什麼？
影響誰？
誰能決定？
哪些內容仍未知？
哪些效果不可逆？
何時算完成？
失敗後由誰承擔？
世界真的發生了什麼？
本文因此把程式重新收束為：

> 【Program
> =
> Intent
> +
> World Model
> +
> Governed Operators
> +
> Executable Difference
> +
> Observed Evidence
> +
> Historical Responsibility】

而程式語言則是使這些結構能被：

- 形成；

- 表達；
- 組合；
- 投影；
- 驗證；
- 授權；
- 執行；
- 觀測；
- 修訂；

的共同介質。

這個框架不否定傳統程式碼。

相反地，它把程式碼放入更大的結構中。

來源碼仍可負責精確表達局部演算法與實作；但目的、世界、權限、效果、證據與責任不應繼續被迫只存在於命名、註解、框架慣例與少數工程師腦中。

後文本程式語言也不意味著 AI 取代人類主體。

它要求更加清楚地區分：

- 人類提出目的與完成不可代理選擇；
- AI 展開、比較、生成與解釋候選結構；
- 編譯器驗證組合、效果與權限；
- Runtime 提交並觀測世界差分；
- 所有參與者共同對證據、殘差與歷史負責。

因此，真正的未來程式設計不是：

> 人類說一句話，
> AI 就直接改變世界。

而是：

人類提出意圖，
系統保存未知與選擇，
AI 展開語意結構，
編譯器檢查合法性，
合法主體授權效果，
Runtime 提交世界差分，
證據回到人類與系統，
世界結果再次修正意圖。

本文的最終收束是：

> 【Intent
> →
> Meaning
> →
> Structure
> →
> Verification
> →
> Authorization
> →
> World Difference
> →
> Evidence
> →
> Responsibility】

以及十八篇系列的最終命題：

> 【程式的終點不是執行，】

> 【而是意圖、計算與世界之間】

> 【形成一條可理解、可驗證、可撤回、】

> 【可恢復並能對後果負責的耦合鏈。】

```
post_text_program:
 program_id: "publish-research-release"
 semantic_version: "3.1"
 intent:
  goal: "publish the approved research release"
  non_goals:
   - "claim peer-review acceptance"
 uncertainty:
  - id: "external-indexing-time"
    kind: "externally-controlled"
 world:
  entities:
   - "research-release"
   - "publication-site"
   - "public-index"
  current_state: "approved-not-public"
 operators:
  - "ValidateRelease"
  - "GenerateProjections"
  - "DeployPublication"
  - "VerifyPublicEvidence"
 effects:
  digital:
   - "create public page"
   - "update index"
  external:
   - "content may be cached"
  irreversible:
   - "third parties may download content"
 permissions:
  edit: "research-team"
  approve: "publication-owner"
  execute: "deployment-runtime"
 verification:
  - "content hash matches approved release"
  - "public page is reachable"
  - "claim status remains unchanged"
 projections:
  - "markdown"
  - "html"
  - "metadata"
  - "deployment-plan"
  - "audit-record"
semantic_change:
 change_id: "grant-admin-scope-expansion"
```

```
  before:
    capability: "manage-content"
    scope: "project-A"
  after:
    capability: "manage-all-resources"
    scope: "organization"
  classification:
    text_lines_changed: 1
    semantic_severity: "critical"
  affected:
    actors:
      - "all organization members"
    effects:
      - "read sensitive resources"
      - "modify organization settings"
  requirements:
    - "security review"
    - "dual approval"
    - "new audit policy"
runtime_commit_package:
  world_change_id: "start-device-117"
  based_on_world_version: 88
  operator_graph:
    root: "StartDeviceSafely"
  guards:
    - "maintenance_mode == false"
    - "safety_zone_clear == true"
    - "approval.valid == true"
  permissions:
    capability: "device-start"
    expires_at: "2026-07-27T09:05:00+08:00"
  predicted_effects:
    physical:
      - "device reaches stable operation"
    limits:
      rpm_max: 1200
  observation_obligations:
    - "controller acceptance"
    - "rpm sensor"
    - "vibration sensor"
    - "human safety confirmation"
  failure_plan:
    divergence: "controlled-stop"
    unknown: "hold-and-escalate"
toolchain:
```

intent_capture: true
semantic_ir: true
operator_registry: true
effect_system: true
uncertainty_nodes: true
human_decision_nodes: true
projection_registry: true
semantic_diff: true
world_version_guard: true
runtime_evidence: true
reconciliation: true
history_ledger: true

- 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

第 0 冊《程式之前》

- PU-0-01　三重宇宙論：意圖、計算與物理存在的基本區分
- PU-0-02　三宇宙耦合動力學：從想要、表示到世界改變
- PU-0-03　表示落差：意圖、計算模型與物理現實之間的不可完備映射
- PU-0-04　數位宇宙作為人工存在域：實體、狀態、規則與歷史
- PU-0-05　生成與限制：計算機宇宙如何創造並封閉可能性
- PU-0-06　我們到底想要什麼：計算目的、價值邊界與世界選擇

SECTION

第 1 冊《系統性程式設計》

- PU-1-01　程式不等於程式碼：可執行問題模型的基本定義
- PU-1-02　問題世界建模：實體、關係、規則與系統邊界
- PU-1-03　資料—狀態—事件—行動：程式系統的四元動力結構
- PU-1-04　責任、模組與契約：從檔案分類到系統邊界
- PU-1-05　可視化作為外部結構記憶：多角色、多尺度的程式理解
- PU-1-06　失敗也是程式：驗證、可觀測、恢復與長期維護

SECTION

第 2 冊《程式語言的本質》

- 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 系列理論

- Neo.K with Aletheia，《三重宇宙論：意圖、計算與物理存在的基本區分》，2026。
- Neo.K with Aletheia，《三宇宙耦合動力學：從想要、表示到世界改變》，2026。

- Neo.K with Aletheia，《表示落差：意圖、計算模型與物理現實之間的不可完備映射》，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，《問題世界建模：實體、關係、規則與系統邊界》，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，《可視化作為外部結構記憶：多角色、多尺度的程式理解》，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，《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，《符號作為算子：從靜態字元到可組合計算閉包》，2026。
- Neo.K with Aletheia，《語法—語意—效果：程式語言的三層存在結構》，2026。
- Neo.K with Aletheia，《意圖中介表示：從自然意圖到多重可執行投影》，2026。
- Neo.K with Aletheia，《可編譯世界：程式執行作為世界狀態差分》，2026。

SECTION

一般理論背景

- Backus, J., “Can Programming Be Liberated from the von Neumann Style?”, 1978.
- Scott, D. and Strachey, C., “Toward a Mathematical Semantics for Computer Languages,” 1971.
- Hoare, C. A. R., “An Axiomatic Basis for Computer Programming,” 1969.
- Pierce, B. C., Types and Programming Languages, 2002.
- Harel, D., “Statecharts: A Visual Formalism for Complex Systems,” 1987.
- Knuth, D. E., “Literate Programming,” 1984.
- Erdweg, S. et al., “The State of the Art in Language Workbenches,” 2013.

- Omar, C. et al., “Hazelnut: A Bidirectionally Typed Structure Editor Calculus,” 2017.
- Moggi, E., “Notions of Computation and Monads,” 1991.
- Plotkin, G. D. and Power, J., “Algebraic Operations and Generic Effects,” 2003.
- Gray, J. and Reuter, A., Transaction Processing, 1992.
- Lamport, L., Specifying Systems, 2002.
- Kleppmann, M., Designing Data-Intensive Applications, 2017.
- Henzinger, T. A., “The Theory of Hybrid Automata,” 1996.
- Alur, R., Principles of Cyber-Physical Systems, 2015.
- Research on bidirectional transformations, language workbenches, program synthesis, effect systems, digital twins, runtime verification, and human-AI collaborative programming.

SECTION

v0.1 — 2026-07-27

- 完成十八篇地基論文第十八篇。
- 完成第 2 冊《程式語言的本質》六篇封頂。
- 完成三冊十八篇「程式宇宙書系：基礎論文系列」封頂。
- 正式定義後文本不等於無文字，而是取消文字對程式本體的壟斷。
- 建立後文本程式語言十二元模型。
- 建立意圖至觀測證據的七層統一架構。
- 將單向編譯擴展為 Runtime 證據回饋閉環。
- 定義核心程式與多模態投影。
- 建立後文本語法、世界型別與擴展判定。

- 區分編輯、生成、批准、執行、觀測與撤銷權。
- 將不可代理選擇建模為語言與 Runtime 節點。
- 建立語意版本控制與語意合併。
- 建立人類、AI、編譯器與 Runtime 的責任分工。
- 建立後文本編譯器十一項核心組件。
- 建立後文本 Runtime。
- 建立命令、物理效果與驗證物理狀態的分層。
- 提出最小可行後文本工具鏈。
- 加入網站、資料報表、AI Agent、研究平台、醫療、自治設備與制度批准案例。
- 提出三十類失敗模式與二十項可證偽研究方向。
- 完成第 2 冊六篇統合與十八篇三冊總統合。