生成與限制：計算機宇宙如何創造並封閉可能性

Generation and Constraint: How the Computational Universe Creates and Closes Possibility

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

https://thisoneisneok.com/html/papers/pu-0-05.html

SECTION

Generation and Constraint: How the Computational Universe Creates and Closes Possibility

論文編號： PU-0-05

系列：「程式宇宙書系」第 0 冊《程式之前》

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

前篇將數位宇宙定義為由實體、身分、狀態、規則、事件、歷史、權限、數位時間與 Runtime／Kernel 共同維持的人工存在域。本篇進一步研究其最核心的動力：計算機宇宙不只是保存既有世界，也會生成新對象、新規則、新行動與新世界；然而，同一套生成機制也必然排除、封閉或壓縮其他可能性。

本文提出：

> 【Computation
> =
> Generation
> +
> Constraint】

有限符號、有限規則與有限運行時，能生成巨大但並非無限自由的可構造空間：

> mathcal W_C
> =
> Closure
> (
> Σ,
> R,
> K,
> P,
> S☒
>)

其中：

- Σ：可用符號與構件；
- R：組合與轉換規則；
- K：Runtime／Kernel；
- P：權限與角色；
- S☒：初始狀態；
- mathcal W_C：該計算機宇宙可到達的世界集合。

  本文把計算生成性定義為：由有限結構產生新對象、新狀態、新組合、新行動與新歷史分支的能力。生成性可表現為：

- 結構組合；

- 程式執行；
- 內容生成；
- 世界模擬；
- 規則變換；
- Agent 行動；
- 自動建模；
- 自我修改；
- 新制度形成。

但生成並非沒有邊界。任何可生成空間都由其語法、型別、資料模型、狀態機、權限、介面、演算法、資源、時間、法規與物理載體共同限定。本文因此提出「雙重空間」：

> mathcal W_(possible)
> ⊇
> mathcal W_(expressible)
> ⊇
> mathcal W_(constructible)
> ⊇
> mathcal W_(authorized)
> ⊇
> mathcal W_(realizable)

其中：

- mathcal W_(possible)：概念或物理上可能的狀態；
- mathcal W_(expressible)：可被語言或模型表示的狀態；
- mathcal W_(constructible)：可由現有結構組合出的狀態；
- mathcal W_(authorized)：制度允許執行的狀態；
- mathcal W_(realizable)：在現有資源與物理條件下真正可實現的狀態。

本文主張，程式語言與系統設計的本質不只是「告訴機器做什麼」，而是定義一個人工宇宙的可生成閉包與非法世界集合：

> 【L
> ⇒
> (
> mathcal W_L^(+),
> mathcal W_L^(-)
>)】

其中：

- mathcal W_L^(+)：語言或系統允許被生成的世界；
- mathcal W_L^(-)：被排除、禁止、無法表達或無法到達的世界。

  本文區分六類限制：

- 語法限制：哪些結構可被寫出；
- 語意限制：哪些結構具有合法意義；
- 型別與狀態限制：哪些狀態被視為有效；
- 權限限制：誰可生成或修改什麼；
- 資源與物理限制：哪些狀態雖可描述卻無法實現；
- 介面與制度限制：哪些意圖因沒有操作、分類或申訴位置而無法進入世界。

  本文特別提出「不可見封閉」：系統不只以明確錯誤拒絕某些狀態，也會透過沒有欄位、沒有按鈕、沒有角色、沒有 API、沒有資料、沒有權限或沒有語言構件，使某些可能性在使用者眼中根本不再出現。

  因此：

> 【Absence of an Operation
> ≠
> Absence of a Need】

本文也提出生成—限制張力。限制不必然是壓迫；它也能建立安全、可組合性、可驗證性與合作基礎。型別排除非法狀態，權限限制未授權行動，交易約束避免世界不一致，語法規則使多人共享可理解結構。成熟系統的問題不是消除限制，而是區分：

- 哪些限制保護世界；
- 哪些限制只是歷史偶然；
- 哪些限制服務權力；
- 哪些限制應可被修改；
- 哪些限制必須對受影響者可見。

本文提出「限制可治理原則」：凡是能夠顯著改變主體可行動空間的數位限制，都應具有來源、理由、適用域、版本、異議與變更路徑。

本文使用程式語言、社群平台、遊戲世界、公共資格系統、AI Agent 與生成式設計等案例，分析生成性如何被制度與架構塑形。本文最後提出可證偽研究綱領，包括表達覆蓋率、非法狀態排除率、操作缺席效應、介面誘導、權限集中、生成空間多樣性、限制透明度、規則修改可達性與創新封閉。

本文為第 0 冊最後一篇〈我們到底想要什麼〉建立直接前提：既然程式系統必然同時生成與限制世界，那麼真正的規範問題就不是「我們能做什麼」，而是「我們希望哪些世界能被生成，哪些世界應被阻止，以及誰有權做出這些選擇」。

關鍵詞： 計算生成性、限制理論、可生成閉包、非法世界、型別、權限、介面封閉、數位制度、程式語言本質

SECTION

Abstract

The previous paper defined the digital universe as an artificial domain of entities, identity, state, rules, events, history, permissions, digital time, and runtime. This paper investigates its central dynamic: computational systems generate new entities, states, actions, and worlds, while the same mechanisms also exclude and close alternative possibilities.

The paper proposes:

> Computation
> =
> Generation
> +
> Constraint

A computational world is defined as a closure over symbols, rules, runtime, permissions, and initial state:

> mathcal W_C
> =
> Closure
> (
> Σ,
> R,
> K,
> P,
> S⊠
>)

The paper distinguishes conceptual possibility, expressibility, constructibility, authorization, and physical realizability. It analyzes syntactic, semantic, type, permission, physical, interface, and institutional constraints.

A key contribution is the concept of invisible closure: possibilities can disappear not only through explicit prohibition but through absent fields, buttons, roles, APIs, categories, or representational structures.

The paper argues that constraints are not inherently oppressive. They can create safety, compositionality, verification, and shared order. Mature systems must make constraints governable, traceable, contestable, and revisable.

Keywords: computational generation, constraint, constructible worlds, invalid states, permissions, interfaces, digital institutions

程式設計常被描述為創造。

人們用程式建立：

- 新工具；
- 新遊戲；
- 新市場；
- 新身份；
- 新內容；
- 新工作流程；
- 新 Agent；
- 新模擬世界。

但每當系統說「這些東西可以存在」時，也同時說：

其他東西不能存在，
不能被表示，
不能被操作，
或不被承認。

例如一個訂單系統若只有：

pending
paid
shipped
completed
cancelled

那麼：

- 爭議中；
- 部分交付；
- 已補償但未撤銷；
- 多方責任未決；

可能沒有正式位置。

所以生成與限制不是兩個獨立階段，而是同一設計行為的兩面。

定義某計算宇宙：

> mathcal C
> =
> <
> Σ,
> R,
> K,
> P,
> S⊠
> >

其可生成世界集合為：

> 【mathcal W_C
> =
> Closure
> (
> Σ,
> R,
> K,
> P,
> S⊠
>)】

SECTION

2.1 閉包的意義

若某狀態能從初始狀態經合法操作有限次到達：

> S⊠
> xrightarrowa⊠
> S⊠
> xrightarrowa⊠
> …

則：

> S⊠
> ∈
> mathcal W_C

SECTION

2.2 閉包不是所有可能世界

即使語言具有高度組合性，其閉包仍由：

- 可用符號；
- 合法規則；
- Runtime 行為；
- 權限；
- 初始條件；

決定。

SECTION

2.3 世界可達性

定義：

> Reachable(S)
> ⇔
> ∃
> (
> a⊠,…,a⊠
>)
> :
> S⊠
> →

一個世界可以被描述，但未必可達。

本文提出：

> 【mathcal W_(possible)
> ⊇
> mathcal W_(expressible)
> ⊇
> mathcal W_(constructible)
> ⊇
> mathcal W_(authorized)
> ⊇
> mathcal W_(realizable)】

SECTION

3.1 可能空間

概念上、邏輯上或物理上可能存在的狀態。

SECTION

3.2 可表達空間

現有語言、資料模型與介面能描述的狀態。

SECTION

3.3 可構造空間

系統現有構件、演算法與 Runtime 能組合出的狀態。

SECTION

3.4 已授權空間

具有合法權限與制度批准的狀態。

SECTION

3.5 可實現空間

在資源、時間、能源、硬體與物理條件下真正可完成的狀態。

SECTION

3.6 層間落差

一個想法可能可表達但不可構造。

一個程式可能可構造但未授權。

一個行動可能已授權但物理上無法完成。

SECTION

4.1 組合生成

有限元素依規則產生大量結構。

SECTION

4.2 狀態生成

執行使世界進入新狀態。

SECTION

4.3 對象生成

建立新帳號、文件、角色、任務、資產或 Agent。

SECTION

4.4 規則生成

系統能建立新規則、DSL、政策或工作流。

SECTION

4.5 歷史生成

每次事件都創造新的世界歷史。

SECTION

4.6 意圖生成

工具能力、介面與可見選項會使使用者形成原先沒有的目標。

程式語言通常只有有限：

- 關鍵字；
- 操作符；
- 型別；
- 語法規則；
- 標準構件。

  但藉由遞迴、抽象與組合，可產生巨大程式空間：

> |mathcal P_L|
> ≫
> |Σ_L|

其中：

- Σ_L：語言基本符號；

- mathcal P_L：可形成的程式集合。

  生成性主要來自：

- 遞迴；
- 參數化；
- 模組；
- 高階抽象；
- 組合；
- 資料驅動；
- Runtime 反覆執行。

  因此語言的力量不只來自構件數量，也來自構件如何閉合。

  沒有任何限制的符號集合，不會自動形成可理解語言。

  規則提供：

- 結構；
- 預期；
- 可驗證性；
- 共享理解；
- 合法組合；
- 錯誤判定。

  所以：

> 【Constraint
> not≡
> Mere Reduction】

限制也可能是：

> 【Constraint
> =
> Condition of Stable Generation】

沒有型別，某些組合失去保障。

沒有權限，任何主體都可任意改寫世界。

沒有交易邊界，狀態可能永久不一致。

SECTION

7.1 語法限制

語法決定哪些符號排列能成為合法表達。

> x
> ∈
> mathcal L_(syntax)

不代表 x 已有正確世界意義，但語法先排除大量無法解析的結構。

SECTION

7.2 語意限制

有些語法合法的表達，仍可能：

- 沒有定義；
- 作用於錯誤對象；
- 違反前置條件；
- 無法在 Runtime 中解釋。

  語意限制決定：

> Meaningful(x)

SECTION

7.3 型別與狀態限制

型別與不變量排除不合法世界。

> ValidState(S)
> ⇔
> S
> |=
> T
> ∧
> S
> |=
> Inv

例如：

- 餘額不得為非法值；
- 已刪除任務不可再次完成；
- 未付款訂單不可出貨；
- 無身份實體不可持有權限。

SECTION

7.4 權限限制

即使行動語法與語意合法，主體也未必有權執行。

> Executable(a)
> ⇔
> Valid(a)
> ∧

SECTION

7.5 資源與物理限制

某世界可被描述，也可被程式構造，但可能因：

- 計算量；
- 記憶；
- 能源；
- 延遲；
- 裝置；
- 材料；
- 人力；

  而無法實現。

SECTION

7.6 介面與制度限制

若系統沒有：

- 按鈕；
- 欄位；
- 分類；
- 流程；
- 角色；
- 申訴；
- API；

某些需求雖然真實存在，卻沒有進入世界的通道。

SECTION

8.1 明確禁止

系統可以直接回覆：

permission denied
invalid state
unsupported operation
此時限制至少可見。

SECTION

8.2 不可見封閉

更深層的限制是：某可能性沒有任何表示位置。

例如：

- 表單只有男性與女性，沒有其他或拒答；
- 任務只有完成與未完成，沒有放棄或不再適用；
- 平台只有按讚，沒有表達保留；
- Agent 只能選擇執行，沒有正式提出異議的狀態。

SECTION

8.3 缺席操作命題

> 【AbsentOperation(a)
> not⇒
> AbsentNeed(a)】

沒有操作，只表示世界設計者未提供此種可能性。

SECTION

8.4 認知封閉

長期使用某套系統後，人們可能不再想到被排除的選項。

所以不可見封閉不只限制行動，也限制意圖形成。

SECTION

9.1 schema 決定可存在對象

資料 schema 看似只是儲存格式，實際上決定：

- 有哪些實體；
- 哪些屬性重要；
- 哪些關係合法；
- 哪些狀態能被保存。

SECTION

9.2 缺少欄位的制度效果

若醫療系統沒有照護負擔欄位，照護壓力就很難進入資源分配。

若工作系統沒有非正式貢獻欄位，某些勞動便難以被承認。

SECTION

9.3 schema 演化

世界會改變，schema 也必須演化。

但遷移需要處理：

- 舊資料；

- 新類型；
- 歷史語意；
- 版本；
- 相容；
- 不可逆轉換。

SECTION

9.4 schema 主權

誰能修改 schema，誰就能決定哪些數位存在獲得正式位置。

SECTION

10.1 型別的積極作用

型別不只是檢查錯誤。

它定義：

> mathcal W_T^(legal)

也就是哪些世界狀態可被構造。

SECTION

10.2 非法狀態不可表示

理想型別設計追求：

> Make invalid states unrepresentable.

例如將付款狀態分離為：

```
UnpaidOrder
PaidOrder
```

ShippedOrder

可避免未付款訂單直接出貨。

SECTION

10.3 型別也可能過度封閉

若世界真實情況比型別更複雜，過度嚴格型別可能：

- 排除例外；
- 強迫錯誤分類；
- 妨礙演化；
- 讓新狀態無法進入。

SECTION

10.4 開放與封閉型別

高穩定核心適合封閉型別。

高變動邊界則需：

- 擴展；
- 版本；
- unknown；
- extension field；
- 人類審核。

SECTION

11.1 狀態機不是中性流程圖

狀態機定義：

- 誰能前進；
- 誰被卡住；
- 是否能返回；
- 是否有例外；
- 哪些狀態被視為結束。

SECTION

11.2 終止狀態

若系統將 rejected 設為不可申訴的終止狀態，就把制度決定寫進技術結構。

SECTION

11.3 可逆邊

一個世界是否允許：

rejected → appealed
deleted → restored
completed → disputed
反映其是否承認錯誤、異議與歷史重開。

SECTION

11.4 狀態機可治理

重大狀態轉移應具有：

- 規則來源；
- 觸發者；
- 證據；
- 時間；

- 回復；
- 申訴。

SECTION

12.1 權限決定誰能創造世界

建立帳號、發布內容、創建資產、修改規則與啟動 Agent，都是生成行為。

SECTION

12.2 生成權

本文定義：

> G_P(a,w)
> =
> actora
> may generate world-statew

SECTION

12.3 生成權集中

若只有平台能：

- 建立新類型；
- 開放新能力；
- 調整排序；
- 修改資產；
- 改寫規則；

則生成性集中於平台。

SECTION

12.4 使用者生成不等於使用者主權

平台可能允許使用者創作內容，但：

- 平台控制可見性；
- 平台保留刪除權；
- 平台控制格式；
- 平台決定變現；
- 平台可終止世界。

  這是受控生成，而非完整主權。

SECTION

13.1 affordance

介面透過可見操作告訴使用者：

- 可以做什麼；
- 應該做什麼；
- 哪些行動容易；
- 哪些行動困難。

SECTION

13.2 操作成本

即使兩個選項都存在，成本不同也會造成偏向。

> P(a□)
> ∝
> (1)/(Cost(a□))

較容易的操作更可能被採取。

SECTION

13.3 深層封閉

若申訴功能藏在多層選單，而購買按鈕始終可見，兩者形式上都存在，但實際可達性不同。

SECTION

13.4 介面生成率

可測量某介面使哪些行動與意圖更常出現。

介面因此是世界生成器，而非單純視圖。

SECTION

14.1 排序生成可見世界

在資訊過量環境中，排序決定使用者實際看到的世界。

> mathcal W_(visible)
> =
> Rank
> (
> mathcal W_(available)
>)

SECTION

14.2 不可見內容的存在困境

內容仍在資料庫中，但若永遠不被推薦，對使用者而言接近不存在。

SECTION

14.3 演算法封閉

推薦系統可能持續放大既有偏好，使替代可能性越來越不可見。

SECTION

14.4 生成性與單一化

生成式系統可以大量產生內容，卻可能因相似資料、獎勵模型與平台偏好造成風格收斂。

所以產量增加不等於可能性多樣性增加。

SECTION

15.1 可計算不等於可負擔

某模型理論上可訓練，但只有少數機構能負擔：

- 算力；
- 電力；
- 資料；
- 人才；
- 基礎設施。

SECTION

15.2 資源決定生成權

資源差異會決定誰能建立：

- 大模型；
- 全球平台；

- 高擬真模擬；
- 大規模 Agent；
- 數位基礎設施。

SECTION

15.3 計算階級

可將生成能力差異表示為：

$$
G\boxtimes
=
f
(
\text{Compute}\boxtimes,
\text{Energy}\boxtimes,
\text{Data}\boxtimes,
\text{Capital}\boxtimes,
\text{Access}\boxtimes
)
$$

SECTION

15.4 開源與共享基礎設施

開源可以降低部分門檻，但不能完全消除硬體、能源與部署差異。

SECTION

16.1 安全限制

安全限制用來排除：

- 非法狀態；

- 未授權行動；
- 不可恢復破壞；
- 資料洩漏；
- 身分冒用；
- 資源失控。

SECTION

16.2 壓迫性限制

壓迫性限制可能：

- 保護既得權力；
- 排除異議；
- 讓退出困難；
- 讓分類無法被挑戰；
- 將管理便利置於個體權利之上。

SECTION

16.3 不能只看形式

同一個「禁止修改」規則，可能：

- 防止未授權破壞；
- 也可能阻止使用者更正錯誤資料。

必須檢查：

- 誰被保護；
- 誰被限制；

- 誰能例外；
- 誰能改規則；
- 是否可申訴。

SECTION

16.4 限制正當性函數

可暫定：

$$
\begin{aligned}
&J(c) \\
&= \\
&f \\
&(\\
&\text{Purpose}, \\
&\text{Necessity}, \\
&\text{Proportionality}, \\
&\text{Transparency}, \\
&\text{Contestability}, \\
&\text{Reversibility} \\
&)
\end{aligned}
$$

限制是否正當，不能只以技術有效性判定。

本文提出：

> 凡是能顯著改變主體可表示、可選擇、可行動或可退出空間的數位限制，都必須具有可識別來源、明示理由、適用域、版本、異議機制與變更路徑。

SECTION

17.1 限制來源

限制可能來自：

- 語言設計；
- 安全政策；
- 法律；
- 產品決策；
- 商業模式；
- 資料缺失；
- 物理資源；
- 歷史遺留。

SECTION

17.2 限制版本化

規則改變時，應保存：

- 舊版本；
- 新版本；
- 修改者；
- 修改理由；
- 受影響對象；
- 遷移方式。

SECTION

17.3 限制異議

受影響者應能提出：

此限制不適用於我的情況。
此分類造成不當排除。

此權限範圍過大。
此狀態應允許回復。
此操作缺席造成傷害。

SECTION

17.4 限制變更

不是所有限制都應由終端使用者自由修改，但至少應有：

- 正式提案；
- 人類審查；
- 例外程序；
- 版本更新；
- 替代世界；
- 退出與遷移。

SECTION

18.1 硬限制

直接禁止：

> a
> ∉
> mathcal A_(legal)

例如權限拒絕與型別錯誤。

SECTION

18.2 軟限制

允許但提高成本：

- 多一步確認；
- 較低優先級；
- 速率限制；
- 額外審核；
- 警告。

SECTION

18.3 預設引導

使用者可以改變，但系統預先選擇某方案。

SECTION

18.4 排序引導

選項都存在，但顯示順序不同。

SECTION

18.5 隱性引導

透過：

- 顏色；
- 大小；
- 位置；
- 文案；
- 速度；

提高某種行動機率。

SECTION

18.6 治理差異

硬限制需要高正當性。

軟限制與引導則需要揭露其誘導效果，避免把行為操控偽裝成自由選擇。

SECTION

19.1 生成模型輸出空間

設模型生成候選集合：

$$
Y \sim p_\theta(y \mid x)
$$

真正可交付輸出還需經：

- 安全；
- 正確性；
- 版權；
- 風格；
- 任務；
- 權限；
- 人類接受；

過濾。

SECTION

19.2 可接受空間

> mathcal Y_(accepted)
> =
> mathcal Y_(generated)
> ∩
> mathcal C_(safety)
> ∩
> mathcal C_(task)
> ∩
> mathcal C_(rights)

SECTION

19.3 過度限制

若過濾器過強，可能排除：

- 正當研究；
- 少數文化表達；
- 邊界藝術；
- 合法異議；
- 非主流語言。

SECTION

19.4 過度生成

若限制不足，則可能產生：

- 虛假；
- 侵權；
- 危險；
- 不可追責；
- 大量低品質內容。

成熟生成系統必須把生成與限制共同設計，而不是事後補丁。

SECTION

20.1 模組化創新

清楚介面與穩定限制，能讓不同團隊在邊界內創新。

SECTION

20.2 過度固定

過度僵硬的：

- schema；
- API；
- 格式；
- 審核；
- 標準；

可能壓制新用法。

SECTION

20.3 破壞性創新與世界相容

新結構若完全破壞舊世界，會產生：

- 資料遺失；
- 身分斷裂；
- 關係消失；
- 工具失效；
- 權利無法遷移。

SECTION

20.4 演化邊界

理想系統應區分：

- 穩定核心；
- 可擴展邊界；
- 實驗區；
- 相容層；
- 遷移機制。

生成能力需要自由，但持續世界需要連續性。

SECTION

21.1 單一世界修改的風險

若所有新規則都直接修改唯一正式世界，創新與穩定會衝突。

SECTION

21.2 分支

可建立：

> W⊠
> →
> (
> W_(stable),
> W_(experimental)
>)

實驗世界允許：

- 新類型；
- 新規則；
- 新介面；
- 新 Agent；
- 新權限模型。

SECTION

21.3 合併

分支成果若要回到正式世界，需要：

- 語意差分；
- 風險檢查；
- 歷史遷移；
- 權限審核；
- 人類接受。

SECTION

21.4 分支不是逃避治理

實驗世界仍需：

- 明確邊界；
- 受影響者知情；
- 不污染正式資料；
- 可清除；
- 可退出。

SECTION

22.1 封閉系統的誘惑

設計者常希望所有情況都進入一般規則。

SECTION

22.2 真實世界的例外

例外可能來自：

- 新情境；
- 資料錯誤；
- 權利衝突；
- 緊急狀態；
- 少數需求；
- 不可預測事件。

SECTION

22.3 例外通道

成熟世界應提供：

- 人類覆核；
- 暫時豁免；
- 申訴；
- 特殊狀態；
- 規則修訂提案；
- 事後稽核。

SECTION

22.4 例外濫用

例外過多也會破壞一致性與公平。

因此需要記錄：

- 誰批准；
- 為何批准；
- 影響何人；
- 是否形成新先例。

SECTION

23.1 定義

系統快速增加新功能、新類型、新規則與新 Agent，卻沒有同步增加：

- 驗證；
- 治理；
- 觀測；

- 文件；
- 遷移；
- 退出；

會形成生成性債務。

SECTION

23.2 形式

> D_G
> =
> G_(new)
> -
> C_(governed)

其中：

- G_(new)：新增生成能力；
- C_(governed)：已被治理、測試與可逆化的能力。

SECTION

23.3 後果

生成性債務會造成：

- 世界規則碎片化；
- 權限擴張；
- 歷史不可理解；
- 例外堆積；
- 不可預期交互作用；

- 無法退出。

SECTION

24.1 定義

系統保留大量過時限制，卻沒有重新檢查其必要性。

SECTION

24.2 來源

- 歷史技術限制；
- 舊法規；
- 過時商業流程；
- 不再存在的安全假設；
- 臨時補丁；
- 舊資料格式。

SECTION

24.3 後果

限制性債務會：

- 阻礙創新；
- 增加操作成本；
- 排除新使用者；
- 固化權力；
- 讓系統看似必然而其實只是歷史偶然。

SECTION

24.4 限制審計

應定期詢問：

這個限制仍有必要嗎？
它現在保護誰？
它現在傷害誰？
能否由更細緻限制取代？

程式語言透過語法、型別、作用域與 Runtime 生成可執行程式空間。

它同時排除：

- 無法解析的表達；
- 無語意操作；
- 非法型別組合；
- 未授權效果；
- 不受支援的記憶模型。

  安全語言利用限制提高可靠性。

  但語言若缺乏：

- 外部效果表示；
- 權限；
- 並行語意；
- 視覺投影；
- 新型資料結構；

  也會限制開發者能自然思考與建構的系統。

  平台允許使用者生成：

- 貼文；

- 影片；
- 關係；
- 社群；
- 聲譽。

但平台控制：

- 格式；
- 字數；
- 可見性；
- 推薦；
- 變現；
- 刪除；
- 申訴。

所以平台是受控生成世界。

使用者生成內容，平台生成可見性與制度效果。

遊戲規則限制玩家：

- 能去哪裡；
- 能使用什麼；
- 能與誰互動；
- 能否死亡；
- 能否復活；
- 能否改變世界。

正是這些限制，使遊戲目標、策略與挑戰成立。

但沙盒、MOD 與世界編輯器會開放更高層生成權，使玩家不只在世界內行動，也能修改世界規則。

公共系統用類型與規則生成：

- 合格；
- 不合格；
- 待補件；
- 審核中；
- 已批准。

  若沒有：

- 特殊困難；
- 申訴；
- 人工審查；
- 例外；

  真實需求可能被封閉在世界之外。

  這表示系統的限制直接改變資源分配與生活結果。

  Agent 的生成空間可能包括：

- 計畫；
- 工具調用；
- 文件；
- 程式；
- 世界狀態修改。

  其限制包括：

- 工具白名單；

- 權限；
- 預算；
- 時間；
- 沙盒；
- 人類批准；
- 不可逆操作禁止。

一個成熟 Agent Runtime 不追求最大自由，而追求：

> 【Maximum Useful Agency
>
> under
>
> Governable Constraints】

- 生成崇拜： 只增加能力，不建立治理與恢復。
- 限制厭惡： 把所有限制都視為效率障礙。
- 限制自然化： 把歷史設計選擇說成技術必然。
- 不可見封閉： 沒有操作或類型，卻不承認需求被排除。
- schema 僭位： 資料模型被視為世界完整分類。
- 型別過度封閉： 為了安全而排除正當新狀態。
- 狀態機終局僭位： 不允許申訴、重開或爭議。
- 權限集中： 少數主體壟斷生成與規則修改權。
- 介面形式自由： 選項存在，但成本與可見性極不對稱。
- 排序封閉： 可用內容存在，卻被演算法永久隱藏。
- 產量等於多樣性： 大量生成被誤認為可能性擴張。

- 資源中立錯覺： 忽略算力、能源與資本決定生成權。
- 安全名義濫用： 以安全遮蔽商業控制或異議壓制。
- 引導隱形化： 介面操控被描述為自由選擇。
- 新功能直上正式世界： 沒有分支、試驗與遷移。
- 例外消滅： 強迫所有現實進入一般規則。
- 例外黑箱： 特權者可秘密越過規則。
- 生成性債務： 新能力超過治理能力。
- 限制性債務： 過時限制持續累積。
- 世界退出缺失： 使用者只能接受既有規則，不能遷移。

SECTION

31.1 表達覆蓋率

測量特定領域的重要現實狀態中，有多少能被現有語言、schema 或介面表示：

> C_E
> =
> (|mathcal W_(expressible)∩mathcal W_(relevant)|)/(|mathcal W_(relevant)|)

SECTION

31.2 可構造率

在可表達狀態中，多少能由現有構件與 Runtime 實際生成。

SECTION

31.3 授權收縮率

比較可構造世界與被授權世界：

> R_A
> =
> 1-
> (|mathcal W_(authorized)|)/(|mathcal W_(constructible)|)

SECTION

31.4 非法狀態排除率

測量型別、約束與狀態機能排除多少已知非法世界。

SECTION

31.5 正當例外召回率

測量系統能否辨識原規則未涵蓋、但應被合法處理的例外。

SECTION

31.6 操作缺席效應

增加原先缺失的操作後，觀察使用者是否大量採用，以判斷原世界是否壓抑真實需求。

SECTION

31.7 介面誘導效應

比較不同按鈕位置、預設值、排序與流程對行動選擇的影響。

SECTION

31.8 生成多樣性

不只測量輸出數量，也測量：

- 結構差異；
- 觀點差異；
- 路徑差異；
- 少數形式保留；
- 長尾覆蓋。

SECTION

31.9 限制透明度

測量使用者是否知道：

- 限制存在；
- 限制來源；
- 限制理由；
- 如何提出異議；
- 是否可修改。

SECTION

31.10 規則修改可達性

測量提出規則修訂到正式審查所需步驟、時間與成本。

SECTION

31.11 生成權集中度

可使用類似集中度指標，衡量生成新類型、修改規則與開放能力的權力分布。

SECTION

31.12 生成性債務

追蹤新增能力與已完成治理、驗證、回復、文件及退出機制之間的差額。

SECTION

31.13 限制性債務

記錄仍存在但已無法提出有效正當理由的限制。

SECTION

31.14 分支創新成功率

比較直接修改正式世界與先在分支世界試驗，對事故率、採用率與遷移成本的影響。

- 計算同時是生成機制與限制機制。
- 每個計算宇宙都有由符號、規則、Runtime、權限與初始狀態決定的可生成閉包。
- 可表達不等於可構造。
- 可構造不等於已授權。
- 已授權不等於物理可實現。
- 程式語言定義合法世界，也定義非法世界。
- 限制不是生成的外部敵人，而常是穩定生成的條件。
- schema 決定哪些對象獲得正式數位存在位置。
- 型別能排除非法狀態，也可能過度封閉新狀態。
- 狀態機中的終止、返回與申訴邊，具有制度與政治意義。
- 權限是生成與修改世界的能力分配。

- 使用者可生成內容，不代表使用者擁有世界主權。
- 介面透過操作可見性與成本塑造可行動空間。
- 排序演算法生成使用者實際可見的世界。
- 內容產量增加不等於可能性多樣性增加。
- 計算資源分布會形成生成能力階級。
- 安全限制與壓迫性限制不能只靠形式區分，必須檢查目的、比例、透明、異議與可逆性。
- 所有重大數位限制都應可追溯、版本化、申訴與修改。
- 世界分支能部分調和創新與穩定，但仍需治理。
- 例外不必等於規則失敗，可能是世界超出原模型的證據。
- 新能力超過治理能力會形成生成性債務；過時限制持續存在會形成限制性債務。

22.

> 【程式設計不是在空白中創造世界，】

> 【而是在選定哪些可能性被打開，】

> 【哪些可能性被關閉。】

SECTION

33.1 承接 PU-0-01

三重宇宙論指出計算機宇宙具有人工規則與結構相對自治。

本文說明這種自治的具體內容，是定義自己的可生成與不可生成空間。

SECTION

33.2 承接 PU-0-02

三宇宙耦合動力學指出工具會改變意圖。

本文進一步說明：工具透過開放與封閉操作空間，生成某些意圖並使其他意圖不可見。

SECTION

33.3 承接 PU-0-03

表示落差指出未被模型表示的內容不能被當成不存在。

本文將此問題推進為不可見封閉：缺少欄位、按鈕、型別或 API，會把現實需求排除於數位世界之外。

SECTION

33.4 承接 PU-0-04

數位宇宙論建立了人工實體、規則與制度。

本文研究這些規則如何產生存在、分配生成權並封閉世界邊界。

SECTION

33.5 銜接 PU-0-06

下一篇將回答：

- 哪些可能世界值得生成；
- 哪些限制應被建立；
- 哪些限制不可被交給單一平台或 Agent；
- 哪些目的不應被單純最佳化；
- 誰有權決定計算方向；
- 人類究竟想從計算機宇宙得到什麼。

  程式語言常被理解為一組語法。

系統架構常被理解為模組與服務的安排。

介面常被理解為操作入口。

權限常被理解為安全設定。

但從計算宇宙的角度看，它們共同完成同一件事：

> 決定一個人工世界中，什麼可以存在、什麼可以發生、誰可以行動，以及哪些可能性從一開始就沒有位置。

有限規則之所以強大，不是因為它能毫無限制地生成一切，而是因為它能建立一個穩定、可重複、可組合的可能性閉包。

正是限制，使程式可以：

- 被解析；
- 被驗證；
- 被共享；
- 被執行；
- 被保護；
- 被協作。

但正是同樣的限制，也可能：

- 固化錯誤分類；
- 排除少數需求；
- 集中平台主權；
- 壓縮意圖；
- 隱藏替代方案；
- 使歷史偶然變成制度必然。

因此，成熟計算理論不能只問：

這個系統可以生成多少東西？

還必須問：

它只能生成哪些東西？
哪些世界被型別排除？
哪些需求沒有欄位？
哪些主體沒有生成權？
哪些限制真正保護安全？
哪些限制只是權力與歷史的沉積？
限制能否被看見、反駁、修改與遷移？

本文將程式系統重新定義為：

> 【Program System
> =
> Possibility Generator
> +
> World Constraint System】

而程式語言的更深層本質是：

> 【L
> =
> Grammar of Constructible Worlds
> +
> Boundary of Invalid Worlds】

這意味著，程式設計從來不是純技術選擇。

每一個型別、欄位、狀態、權限、排序與介面，都在對可能性進行分配。

本文的最終命題是：

> 【計算機宇宙的自由，】

> 【不是沒有邊界，】

> 【而是邊界能被理解、證明、質疑、修改，】

> 【並由受其影響者共同治理。】

```yaml
computational_world:
  world_id: "task-world-v2"
  primitives:
    entities:
      - "user"
      - "task"
      - "project"
  states:
    task:
      - "proposed"
      - "active"
      - "completed"
      - "cancelled"
      - "disputed"
  transitions:
    - from: "active"
      to: "completed"
      requires:
        - "completion-evidence"
    - from: "completed"
      to: "disputed"
      allowed_by:
        - "owner"
        - "reviewer"
  constraints:
    safety:
      - "deleted task cannot execute"
    rights:
      - "private task cannot be read by non-owner"
  extensions:
    unknown_state_allowed: true
    human_review_route: true
constraint_record:
  constraint_id: "production-deploy-approval"
  type: "authorization"
  effect:
    prevents:
      - "agent direct production deployment"
  purpose:
    - "prevent irreversible public changes"
  origin:
    policy: "deployment-governance-v3"
```

```
  owner: "release-board"
scope:
  actors:
    - "autonomous-agent"
  environments:
    - "production"
review:
  last_reviewed: "2026-07-27"
  next_review_due: "2026-10-27"
contestability:
  appeal_available: true
  exception_process: "emergency-release-review"
reversibility:
  constraint_can_be_changed: true
  change_requires:
    - "security-review"
    - "human-approval"
```

等級　生成能力　限制治理

GC0　固定功能　限制隱藏且不可質疑

GC1　可配置　有錯誤訊息但無來源

GC2　可擴展 schema／API　限制有文件與版本

GC3　分支與實驗世界　有申訴、例外與回復

GC4　Agent 與自動生成　權限、證據、預算與不可逆控制

GC5　世界規則可演化　限制可追溯、可遷移、可共同治理

GC6　多投影可編譯世界　生成、限制、殘差與目的共同編譯

- 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

Neo.K／EveMissLab 相關理論

- 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，《數位宇宙作為人工存在域：實體、狀態、規則與歷史》，2026。
- Neo.K with Aletheia，《結構先於文字：Nova 與後文本程式語言本體論》，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，《可編譯世界：從程式執行到世界狀態演化》，2026。

SECTION

一般理論背景

- Chomsky, N., Syntactic Structures, 1957.
- Simon, H. A., The Sciences of the Artificial, 1969.
- Scott, D. and Strachey, C., “Toward a Mathematical Semantics for Computer Languages,” 1971.
- Milner, R., “A Theory of Type Polymorphism in Programming,” 1978.
- Norman, D. A., The Design of Everyday Things, 1988.
- Lessig, L., Code and Other Laws of Cyberspace, 1999.
- Pierce, B. C., Types and Programming Languages, 2002.
- Alexander, C., The Nature of Order, 2002–2004.
- Zittrain, J., The Future of the Internet and How to Stop It, 2008.

- Ostrom, E., Understanding Institutional Diversity, 2005.
- Winner, L., “Do Artifacts Have Politics?”, 1980.

SECTION

v0.1 — 2026-07-27

- 完成十八篇地基論文第五篇。
- 建立計算生成與限制的統一命題。
- 定義可生成世界閉包與五層可能空間。
- 區分六類生成形式與六類限制。
- 提出不可見封閉與缺席操作命題。
- 分析 schema、型別、狀態機、權限、介面、演算法與資源的世界封閉效果。
- 提出生成權與計算階級。
- 區分安全限制與壓迫性限制。
- 建立限制可治理原則。
- 區分硬限制、軟限制、預設、排序與隱性引導。
- 分析自動生成的可接受空間。
- 建立分支世界、例外通道、生成性債務與限制性債務。
- 加入程式語言、社群平台、遊戲世界、公共資格系統與 AI Agent 案例。
- 提出二十類失敗模式與十四項可證偽研究方向。
- 完成與 PU-0-06《我們到底想要什麼》的銜接。