# John Backus：從 FORTRAN 到函數級程式設計的自我反省

John Backus: From FORTRAN to the Self-Critique of Function-Level Programming

v1.0 / 2026-07-30 / Neo.K / 公開版／第三部設計師個案正式研究

https://thisoneisneok.com/html/papers/pldst-011.html

英文名稱： John Backus: From FORTRAN to the Self-Critique of Function-Level Programming

系列： Programming Language Designer Style Taxonomy（PLDST）

文件編號： PLDST-011

版本： v1.0

日期： 2026-07-30

作者： Neo.K

文件狀態： 公開版／第三部設計師個案正式研究

SECTION

## 摘要

John Backus 在程式語言史上呈現一種罕見的雙重角色：他先領導團隊建立 FORTRAN，使高階數學表示能被翻譯為接近手寫效率的機器程式；二十年後，他又在圖靈獎演講中批判以變數、賦值、狀態與逐字操作為中心的傳統語言，並承認自己對這種複雜性可能負有一部分責任。[R1][R2][R3]

若只以「FORTRAN 之父」描述 Backus，會忽略他後半生對函數級程式設計、程式代數與非馮紐曼模型的探索；若把他描述成「拋棄命令式、皈依函數式」，又會把他的工作錯寫成現代 Lambda-calculus 函數語言的直接前身。Backus 的 FP 系統特別重視 Function-level programming：程式由既有函數與 Combining forms 組成，盡量不透過命名變數、逐字賦值及顯式遞迴描述計算。他不只批判傳統命令式語言，也認為既有 Applicative systems 在歷史狀態、實用性和完整系統方面仍有不足。[R1]

本文使用 PLDST 的時間相位、多主體歸因、複雜度配置、責任配置與核心—擴張模型，將 Backus 的設計生涯分為四個相位：

- 機器摩擦消除期：Speedcoding 與早期自動程式設計；
- 實用高階語言與最佳化編譯期：FORTRAN I／II／III；
- 形式描述與語言結構期：ALGOL、BNF 與語言規格；
- 函數級反省期：Closed applicative languages、FP、程式代數及對其限制的後期承認。

本文的核心判斷是：

> Backus 的深層風格並不是「命令式」或「函數式」二選一，而是反覆嘗試把人類從當代機器與語言的低階偶發負擔中解放出來，同時要求新抽象仍具有可實作、可推理與可轉換的結構。
> FORTRAN 將人從機器碼、索引管理與手工優化中提高一層；FP 則試圖再把人從 FORTRAN 所代表的狀態、賦值與逐字控制中提高一層。兩者表面相反，深層問題設定卻一致：

> 【消除不必要的操作細節
> +
> 保留計算能力
> +
> 使機器承擔轉譯與最佳化】

然而，Backus 的後期自我評價同樣重要：他承認 FP 難以乾淨納入輸入、輸出及其他周邊問題，並把其作為完整系統的路線稱為最終未成功。這使他的風格不是封閉教條，而是一種能對自己的革命方案進行二次否定的反身設計。

關鍵詞： John Backus、FORTRAN、函數級程式設計、FP、馮紐曼瓶頸、程式代數、BNF、編譯器、PLDST

SECTION

## 一、本文研究的不是「FORTRAN 所有歷史」

本文主要研究 Backus 可直接歸因的設計決策與思想變化，不把後來數十年的 Fortran 標準、生態與編譯器全部歸入他的個人風格。

分析範圍包括：

- Speedcoding；
- IBM 704 FORTRAN I／II／III；
- 早期最佳化編譯目標；
- ALGOL 語法描述與 BNF；
- 1973 年 Closed applicative language 工作；
- 1977 圖靈獎演講及 FP；
- 後期口述歷史中的自我評價。

SECTION

## 二、FORTRAN 是團隊成果

Backus 是 FORTRAN 計畫的發起者和領導者，但 FORTRAN I 編譯器不是單人作品。

Computer History Museum 保存的原始團隊包括：

- John Backus；
- Sheldon Best；
- Richard Goldberg；
- Lois Mitchell Haibt；
- Harlan Herrick；
- Grace Mitchell；
- Robert Nelson；
- Roy Nutt；
- David Sayre；
- Peter Sheridan；

- Irving Ziller。[R4]

Backus 本人也將 FORTRAN I 編譯器的困難問題描述為主要規劃者和程式設計者的集體成果。[R2]

因此：

創始與領導功勞：Backus 高
核心規格與問題設定：Backus 高
最佳化編譯器實作：團隊共同
長期 Fortran 演化：後續標準、實作者與使用者共同

SECTION

## 三、FP 的個人權重較高，但仍非完全單人

Backus 在口述歷史中把早期函數程式設計探索形容為主要由自己進行，並提及曾短暫與 Edgar F. Codd 合作；後來也有 John H. Williams 等研究者加入相關工作。[R5][R6]

因此，FP 相較 FORTRAN 更能反映 Backus 個人後期風格，但：

- 實作；
- 後續 FL／FFP；
- 相關硬體；
- 函數程式設計整體發展；

仍不能全部歸於 Backus。

SECTION

## 四、早期問題不是語言美學，而是程式設計太困難

Backus 早期面對的環境包括：

- 機器碼與粗糙組合語言；
- 手工配置暫存器與儲存位置；
- 缺乏內建浮點或索引能力；
- 程式除錯耗時；

- 機器時間昂貴；
- 自動程式設計系統通常使執行速度降低。

  他在口述歷史中把機器碼程式設計形容為很糟糕的工作，Speedcoding 的直接動機就是「讓它稍微容易一點」。[R5]

SECTION

## 五、Speedcoding：以虛擬機器交換人類便利

Speedcoding 為 IBM 701 提供較高階的：

- 浮點；
- 索引；
- 數學操作；
- 直譯執行。

  其配置大致是：

> C_(programmer)downarrow
>
> K_(runtime)uparrow

它降低程式設計負擔，卻以明顯執行成本換取便利。

當 IBM 704 直接提供浮點與索引暫存器後，Backus 判斷需要重新設計：不再只建立慢速虛擬機器，而要把高階數學語言翻譯成高效率機器碼。[R5]

SECTION

## 六、第一個深層風格：便利必須跨過性能信任門檻

當時程式設計者不相信編譯器能產生接近手寫品質的程式。

FORTRAN 要取得採用，不只必須：

- 更容易寫；
- 更容易讀；
- 支援數學公式；

  還必須證明：

> Performance_(compiled)
> ≈
> Performance_(hand-coded)

所以 Backus 的早期人本設計不是以犧牲效率為前提，而是把最佳化責任交給編譯器。

SECTION

## 七、FORTRAN 的原始問題框架

FORTRAN 的核心問題可以表示為：

讓科學家使用接近數學的形式描述計算
同時讓機器自動產生可接受效率的程式
這不是只建立新語法，而是同時建立：

- 語言；
- 編譯器；
- 最佳化；
- I/O；
- 除錯；
- 文件；
- 使用者信任。

SECTION

## 八、語言與編譯器是不可分割的產品

FORTRAN 的成功依賴：

> LanguageDesign
> +
> Optimization
> +
> MachineKnowledge
> +
> Engineering
> +
> Distribution

如果語言好寫但生成程式太慢，使用者仍會回到手寫機器碼。

因此 Backus 早期風格可稱為：

> 實用編譯器現實主義：高階抽象只有在編譯器能可靠支付其機器成本時，才會成為真正可採用的語言。

SECTION

## 九、複雜度配置

FORTRAN 將複雜度由每位使用者移至編譯器團隊：

> N·
> (
> C_(register)
> +
> C_(address)
> +
> C_(control)
> +
> C_(optimization)
> )

轉為：

> C_(compiler implementation)
> +
> N· C_(high-level program)

這是 PLDST-002 所稱的 攤銷型複雜度轉移。

編譯器極難建立，但其成果可以由大量程式設計者重用。

SECTION

## 十、責任配置

FORTRAN 將下列責任交給編譯器：

- 表達式翻譯；
- 儲存配置；
- 暫存器使用；
- 迴圈最佳化；
- 部分索引處理；
- 指令排序。

程式設計者仍負責：

- 科學模型；
- 演算法；
- 資料結構；
- 數值假設；
- I/O 規格。

這不是消滅責任，而是將機械責任集中給專業工具。

SECTION

## 十一、表達自由與限制

FORTRAN 對科學計算極有效，但其表達模型仍圍繞：

- 數值；
- 陣列；
- 變數；
- 賦值；
- 控制流；
- 儲存狀態。

這些特徵在當時是對機器碼的巨大提升，後來卻成為 Backus 批判的對象。

SECTION

## 十二、從製造語言到描述語言

Backus 參與 ALGOL 58／60 相關工作，並提出形式化描述程式語言語法的方法；該方法後由 Peter Naur 改進及用於 ALGOL 60 報告，形成今日所稱 Backus–Naur Form。[R1][R7]

這項工作反映另一種風格：

> 語言不只要能被實作，也應能以獨立於單一編譯器的形式被精確描述。

SECTION

## 十三、規格作為共同協作介面

BNF 將語法結構從：

- 編譯器程式；

- 非正式英文；
- 個人直覺；

  轉成可被：

- 設計者；
- 實作者；
- 標準委員會；
- 研究者；

  共同討論的形式物件。

  它提高：

> Q_(specification)
> +
> C_(multi-implementation)
> +
> A_(governance)

SECTION

## 十四、Backus 的形式化不是脫離工程

他的形式語法工作與 FORTRAN 的實用編譯器工作並不矛盾：

- 編譯器要求可實作語言；
- 規格要求可傳達語言；
- 程式代數要求可推理語言。

  三者構成其生涯中逐步上移的抽象階梯。

SECTION

# 十五、不是對硬體人物的責備

Backus 所稱「馮紐曼語言」不是責怪 John von Neumann 個人。

他明確指出，名稱是為了標示傳統語言與儲存程式電腦之間的起源及風格關係，並補充自己可能對問題負有一部分責任。[R1]

SECTION

# 十六、核心批判：語言模仿儲存機器

他的對應關係是：

變數　　≈ 儲存格
賦值　　≈ 取值、運算、寫回
控制陳述　≈ 跳躍與測試
程序執行　≈ 逐字搬運資料

因此傳統程式設計者不只描述問題，而必須規劃大量資料如何穿越 CPU 與 Store 之間的瓶頸。

SECTION

# 十七、物理瓶頸與知識瓶頸

Backus 的重要洞見不是只有記憶體頻寬。

他把瓶頸分為：

## 17.1 物理瓶頸

CPU 與 Memory 之間逐字傳輸。

## 17.2 智識瓶頸

程式設計者被迫以逐字、索引、狀態變化思考，而不是以問題中的大型概念單位思考。

因此：

> Architecture
> →
> LanguageModel
> →
> CognitiveModel

SECTION

## 十八、賦值造成兩個世界

Backus 把傳統語言分成：

- 表達式世界；
- 陳述與賦值世界。

  表達式較容易具備代數性質；賦值與副作用則破壞：

- 替換；
- 組合；
- 等式推理；
- 局部理解。

  Structured programming 在他看來只能改善陳述世界的秩序，沒有移除賦值所造成的根本分裂。[R1]

SECTION

## 十九、Object level 與 Function level

一般函數語言常以：

- 變數；
- Lambda；
- 遞迴；

- 函數定義；

  操作資料。

  Backus 的 FP 更強調：

- 已有函數；
- Combining forms；
- Composition；
- Construction；
- Condition；
- Insert／Reduce；
- Apply-to-all。

  程式在 函數層級 組合，而不是反覆命名資料元素。

SECTION

## 二十、減少名稱與逐元素控制

FP 希望：

描述整體資料轉換
而不是
描述每個儲存位置如何被逐步更新

因此其理想程式具有：

- 較少變數；
- 較少賦值；
- 較少顯式迴圈；
- 較多組合；
- 較強代數轉換能力。

SECTION

## 二十一、程式代數

Backus 主張，若程式由具有良好性質的 Combining forms 組成，可以建立：

- 等式；
- 變換規則；
- 程式推導；
- 程式方程求解；
- 正確性推理；
- 最佳化。

  其理想是：

> Program
> ∈
> AlgebraicObjects

而不只是無法整體推理的狀態操作序列。

SECTION

## 二十二、不是單純 Lambda calculus 路線

Backus 的演講同時批評三類模型：

- 簡單操作模型；
- Applicative models；
- Von Neumann models。

  他認為 Lambda-calculus 類 Applicative model 雖具有簡潔基礎與清楚語義，卻難以保存跨程式歷史資訊；因此他希望發展能處理歷史狀態、又不回到馮紐曼逐字風格的新模型。[R1]

所以：

Backus FP
≠
現代所有函數式語言
≠
純 Lambda calculus 的直接同義詞

SECTION

## 二十三、FORTRAN 不是被簡單否定

Backus 沒有否定 FORTRAN 在其歷史條件下的價值。

FORTRAN 解決的是：

- 人與機器碼的距離；
- 編譯器效率；
- 科學程式生產力。

他的後期批判是：

> 當高階語言已建立後，語言仍然過度模仿儲存機器，下一層抽象革命沒有完成。

SECTION

## 二十四、FP 也被他自己否定了一部分

在 2006 年口述歷史中，Backus 說明：

- Combining forms 的概念相對容易；
- 要把 FP 建成立即能處理完整現實問題的系統，尤其 I/O 及其他周邊問題，變得混亂；
- 該路線作為完整系統最終未成功；
- 仍需要有人找到乾淨納入這些問題的方法。[R5]

這是重要反證：

> Critique_(vonNeumann)
> not⇒
> FP_(complete)

SECTION

## 二十五、他仍保留的判斷

即使承認完整系統問題，Backus 仍認為：

- 資料到資料的轉換適合函數程式設計；
- 程式應更高層描述想要的轉換；
- 軟體複雜度仍是未被解決的核心問題；
- I/O 與現實世界互動需要新的乾淨模型。[R5]

SECTION

## 二十六、第一相位：摩擦消除者

問題：機器碼太難
策略：建立更容易的操作層
代價：Runtime／直譯成本
代表：Speedcoding

SECTION

## 二十七、第二相位：實用編譯器建築師

問題：高階語言不受效率信任
策略：高階數學表示 + 強最佳化編譯
責任：把低階機械工作集中給編譯器
代表：FORTRAN

SECTION

## 二十八、第三相位：形式結構設計者

問題：語言缺乏可共享的精確描述

策略：形式語法
代表：BNF／ALGOL

SECTION

## 二十九、第四相位：代數反身批判者

問題：傳統高階語言仍模仿儲存機器
策略：函數級組合、程式代數、非逐字轉換
代表：FP

SECTION

## 三十、第五相位：限制承認者

問題：FP 難以完整處理 I/O 與周邊世界
判斷：核心方向仍有價值，完整系統方案不足
這不是思想失敗，而是風格中重要的「可否證性」。

SECTION

## 三十一、問題 framing

Backus 反覆把問題理解為：

> 人類正在替機器支付太多本可由系統承擔的操作細節。

SECTION

## 三十二、價值優先序

```
V_(Backus)
≈
(
Abstraction,
Efficiency,
FormalStructure,
Composability,
ConceptualClarity,
Implementability
```

SECTION

# 三十三、複雜度配置

## FORTRAN

> C_(programmer)downarrow
>
> C_(compiler)uparrow

## FP

> C_(state reasoning)
> +
> C_(control detail)
> downarrow

> C_(combining system)
> +
> C_(implementation)
> +
> C_(I/O integration)
> uparrow

SECTION

# 三十四、責任配置

Backus 傾向把：

- 翻譯；
- 最佳化；

- 代數轉換；
- 低階控制；

  交給語言系統。

  但他不認為抽象可以逃避實作：

- FORTRAN 必須產生高效程式；
- FP 必須有有效機器或實作模型；
- 語言必須處理現實 I/O。

SECTION

## 三十五、控制與推導偏好

他傾向：

- 使用者描述 What；
- 系統推導 How；
- 以結構化 Combining forms 取代逐步命令；
- 以代數規則轉換程式。

  但這不是讓系統「猜意圖」，而是要求語言具有可形式化、可推導的組合結構。

SECTION

## 三十六、核心—擴張偏好

Backus 後期反對：

- 功能不斷堆疊；
- 狀態框架內建大量特殊規則；
- 程序宣告與 Naming convention 膨脹。

他偏好：

- 少量 Combining forms；
- 從已有程式建立新程式；
- 核心規則具有代數性。

SECTION

## 三十七、FORTRAN 的成功不能只歸因語言理論

它同時依賴：

- IBM 704；
- 團隊工程；
- 最佳化；
- 發布；
- 科學市場；
- 文件與培訓。

SECTION

## 三十八、BNF 不是 Backus 單人最終形式

Backus 提出原始形式語法方法，Peter Naur 對 ALGOL 60 描述作出重要改進與制度化使用，因此稱為 Backus–Naur Form 比只稱 Backus Form 更符合多主體歸因。[R7]

SECTION

## 三十九、FP 並未成為完整主流語言

原因包括：

- I/O；

- 狀態；
- 生態；
- 實作；
- 硬體；
- 使用者習慣；
- 與主流語言的整合。

其理論影響不能等同直接採用成功。

SECTION

## 四十、後期訪談是回顧性材料

Backus 對早期工作與 FP 成敗的評價具有重要價值，但距事件多年，必須與同期論文、團隊文件及實作史交叉查核。

時期 問題 決策 複雜度去向 主要風格

1950s 初 機器碼困難 Speedcoding Runtime 摩擦消除

1954–1957 高階語言效率不可信 FORTRAN＋最佳化編譯器 編譯器團隊 實用工程

1958–1960 語言規格不精確 形式語法 規格與工具 形式建築

1960s–1970s 語言仍過度依賴狀態 Closed applicative／FP Combining forms 代數極簡

1977–1981 程式難以轉換推理 程式代數 形式系統 變換導向

後期回顧 FP 難處理完整世界 承認 I/O／周邊缺口 未解問題 反身修正

SECTION

## 四十一、主要原型

Backus 同時屬於：

- 實用編譯器現實主義者；

- 形式語言建築師；
- 語義極簡與組合擴張者；
- 反身式範式批判者。

SECTION

## 四十二、不適合的簡單標籤

不應只稱：

命令式語言設計者
函數式語言設計者
FORTRAN 之父
反馮紐曼主義者
更精確的風格描述是：

> 以工程方式建立高階抽象，又在抽象成為新限制後，從其內部發起下一輪批判的人。

SECTION

## 四十三、最重要的連續性

FORTRAN 與 FP 的共同點不是語法，而是：

> HumanDetailBurdendownarrow
>
> SystemTransformationResponsibilityuparrow

SECTION

## 四十四、最重要的不連續性

FORTRAN：

在馮紐曼機器上建立更高階表面
FP：

試圖改變程式本身的概念模型與可推理基礎

SECTION

## 四十五、最重要的自我修正

Backus 從：

高階語言能解決機器碼問題

走向：

傳統高階語言仍被機器模型限制

再走向：

反狀態的函數級方案仍未乾淨處理完整世界

這是一條三階反省鏈。

John Backus 的價值不只在於建立 FORTRAN，也不只在於預言函數式程式設計。

他的設計生涯展示：

- 抽象必須以真實工程跨過採用門檻；
- 編譯器可以把大量低階複雜度從所有使用者集中給少數專家；
- 語言應被精確描述，才能成為跨實作與跨共同體的形式物件；
- 成功語言仍可能保存其所超越機器的思考方式；
- 程式若具有代數組合結構，才更容易被轉換與推理；
- 新範式也必須面對 I/O、狀態與完整系統，不可只靠純粹核心自我證成。

本文對 Backus 的 PLDST 判定為：

> 【Pragmatic Abstraction Engineer
> →
> Formal Language Architect
> →
> Reflexive Algebraic Critic】

他的深層指紋不是某一種語法，而是：

> 【反覆把「人必須如何操作」
>
> 提升成「人想完成什麼轉換」】

同時，他的晚年反思提醒：

> 任何宣稱解放程式設計的新核心，只要不能乾淨容納現實世界的效果、狀態與周邊條件，就仍不是完整答案。

人物：John Backus
主要語言／系統：Speedcoding、FORTRAN、ALGOL syntax、FP
核心時期：1950–1980s
主要問題：人類承擔過多機器與控制細節
主要策略：高階翻譯、最佳化、形式語法、函數級組合
複雜度去向：Compiler／formal system
責任去向：機械翻譯與最佳化交給系統
主要保護對象：科學使用者、程式可推理性
主要限制：I/O、狀態、完整系統整合
歸因信心：高

[R1] John Backus, “Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs,” Communications of the ACM 21(8), 1978, pp. 613–641.

— 馮紐曼瓶頸、賦值批判、FP、Combining forms 與程式代數。

[R2] John Backus, “The History of FORTRAN I, II, and III,” HOPL／Annals of the History of Computing.

— FORTRAN 的形成、團隊、早期自動程式設計、編譯器與最佳化。

[R3] IBM, “Fortran” and “John Backus,” IBM History.

— FORTRAN 的歷史定位、Backus 的職業脈絡與 IBM 團隊。

[R4] Computer History Museum Software Preservation Group, “History of FORTRAN and FORTRAN II.”

— 原始 FORTRAN 團隊、程式碼、設計文件及編譯器保存。

[R5] Grady Booch and Gardner Hendrie, “Oral History of John Backus,” Computer History Museum, 2006.

— Speedcoding、FORTRAN 團隊、FP 動機、I/O 限制與後期自評。

[R6] Computer History Museum Software Preservation Group, “History of John Backus’s Functional Programming Project.”

— FP、FL 及相關研究者與實作材料。

[R7] ACM Turing Award citation and ALGOL／BNF historical materials preserved by ACM, IBM and the Computer History Museum.

— Backus 的形式語法貢獻、Naur 的後續作用與命名邊界。

[R8] John Backus, “Programming Language Semantics and Closed Applicative Languages,” POPL 1973.

— Closed applicative language 的形式背景及語言與實現的區分。

[T-M] Machine-friction phase
[T-F] FORTRAN／compiler phase
[T-S] Formal-syntax phase
[T-P] FP／program-algebra phase
[T-R] Retrospective limitation phase
[S-P] Pragmatic abstraction
[S-C] Compiler-centered allocation
[S-F] Formal specification
[S-A] Algebraic composition
[S-R] Reflexive self-critique

SECTION

# D.1 FORTRAN 的作者歸因

第二輪重新核對 Backus 的 HOPL 歷史論文、IBM 歷史頁面與 Computer History Museum 保存資料：

- Backus 發起並領導 FORTRAN 計畫；
- CHM 保存的原始團隊名單包含十一位主要成員；
- Backus 的歷史論文另將 FORTRAN I 編譯器六個區段的九位主要規劃者與程式設計者稱為集體完成困難問題的主體；
- 不同數字對應不同統計範圍，不應被合併成「只有九人」或「固定十一人完成所有工作」；
- 本文因此使用「Backus 領導的團隊」與「主要編譯器規劃／實作者」兩層表述。

SECTION

# D.2 FORTRAN 的歷史地位

IBM 與 CHM 常把 FORTRAN 描述為第一個廣泛使用的高階程式語言，以及早期具有高品質最佳化能力的編譯系統。

本文沒有使用「世界上第一個程式語言」等過強說法，也沒有否認：

- Laning–Zierler；
- A-2；
- Autocode；
- Plankalkül；
- 其他早期自動程式設計系統。

  本文的判定集中在「廣泛採用、高階科學語言與最佳化編譯器的結合」。

SECTION

## D.3 BNF 與 Peter Naur

第二輪重新核對 ACM 圖靈獎資料、ALGOL 歷史與 HOPL 保存材料：

- Backus 為 IAL／ALGOL 58 提出 Metalinguistic formula；
- Peter Naur 在 ALGOL 60 Report 中修改符號、編輯並制度化使用；
- Donald Knuth 後來主張將 BNF 解讀為 Backus–Naur Form，而不是把它稱為一種真正的 Normal form；
- 因此本文將 Backus 記為原始形式方法的重要提出者，同時保留 Naur 對最終形式與傳播的獨立功勞。

SECTION

## D.4 FP 與一般函數式程式設計

第二輪重新核對 1978 圖靈獎演講：

- Backus 的 FP 以 Function-level programming、Combining forms 與程式代數為中心；
- 他同時批評傳統 Von Neumann language 與若干既有 Applicative model；

- 因此不能把 FP 直接等同 Haskell、ML、Scheme 或所有 Lambda-calculus 傳統；
- 本文使用「函數級」而非籠統「純函數式」，以保存其特殊設計方向。

SECTION

## D.5 Backus 的自我反省

第二輪核對 2006 年口述歷史：

- Backus 明確說 FP 形成完整系統時，I/O 與其他周邊問題變得混亂；
- 他把其完整系統路線稱為最終未成功；
- 他仍認為資料到資料的轉換特別適合函數程式設計；
- 他仍把「描述想要的轉換，而不是如何一步步完成」視為正確方向。

本文因此沒有把後期訪談解釋成全面撤回 1978 年批判，而是「方向保留、方案不完整」。

SECTION

## D.6 時間線校準

1950：加入 IBM
1953–1954：Speedcoding 與 FORTRAN 提案形成
1954–1957：FORTRAN I 設計及編譯器工程
1958–1960：ALGOL／形式語法工作
1960s–1970s：Closed applicative／FP 研究
1977／1978：圖靈獎演講發表與刊登
1981 前後：Function-level program 後續研究
2006：CHM 口述歷史
本文沒有把所有階段視為互相替代的瞬間轉向，而是允許實用、形式與函數級研究在時間上部分重疊。

SECTION

## D.7 公開評價邊界

本文的「反身式範式批判者」是 PLDST 分析推論，不是 Backus 自稱的學派名稱。

它表示：

- 他批判自己曾協助建立的主流框架；
- 又在晚年批判自己的替代方案未能處理完整世界；
- 這種二階自我修正可由原始論文與口述歷史支持。

  它不表示 Backus 是所有現代函數式方法的直接來源。