小林直樹先生の研究の流れ

— 線形論理による並行オブジェクト指向言語から、時相論理の高階モデル検査まで —

児玉工房 / AICE · AIPL プロジェクト

2026 年 07 月 30 日 11:20 JST 版

概要

小林直樹（東京大学）の研究を、1992 年の線形論理に基づく並行計算から 2026 年現在に至るまで、五つの時期に分けて辿る。書誌は DBLP の `Naoki Kobayashi 0001` の記録に基づき、全 229 件（CoRR を除いて 203 件）から主要なものを選んだ。

本稿の出発点について、一点だけ前提を正しておきたい。ご依頼は「時相論理を並列オブジェクト指向言語に導入した研究から」であったが、DBLP の記録では初期の並行オブジェクト指向言語の基礎づけは**時相論理ではなく線形論理**に基づいている（ACL および OOPSLA 1994 の型理論的基礎づけ）。時相論理・モーダル論理が現れるのはむしろ後期であり、2009 年のモーダル μ 計算との等価性、2016 年以降の時相検証と高階不動点論理 HFL がそれである。

したがって「論理が一貫して中心にある」という見立ては正しく、変わったのは論理の役割である。初期の線形論理は**プログラムを書くための論理**であり、後期の時相論理は**プログラムを検証するための論理**である。本稿はこの転回を軸に全体を読む。

目次

1 はじめに

1.1 対象と方法

対象は東京大学の小林直樹による研究である。同姓同名が DBLP に十数名あるため、`Naoki Kobayashi 0001`（Graduate School of Information Science and Technology, University of Tokyo）の記録に限定した。全 229 件、うち CoRR（プレプリント）を除くと 203 件である。年別の件数は 1992 年の 1 件から始まり、2020 年・2021 年に各 12 件で最多となる。

書誌はすべて DBLP で確認した。内容の記述は、論文題名・掲載場所から確実に言えることを基本とし、本文まで確認できたものについてはそれを明示する。

1.2 前提の訂正 —- 出発点は線形論理である

ご依頼の「時相論理を並列オブジェクト指向言語に導入した研究」に相当するものは、DBLP の記録には見当たらない。初期の一連の仕事は**線形論理**に基づく。

- 1992 **Naoki Kobayashi, Akinori Yonezawa**. Asynchronous Communication Model Based on Linear Logic. *Parallel Symbolic Computing.*（1995 年に *Formal Aspects of Computing* 誌）
- 1993 **Naoki Kobayashi, Akinori Yonezawa**. ACL —- A Concurrent Linear Logic Programming Paradigm. *ILPS.*
- 1994 **Naoki Kobayashi, Akinori Yonezawa**. Type-Theoretic Foundations for Concurrent Object-Oriented Programming. *OOPSLA.*

一方、時相論理・モーダル論理は後期に、しかも**検証の対象**として現れる。

- 2009 モーダル μ 計算のモデル検査と等価な型システム（LICS）
- 2016 高階関数プログラムの時相検証（POPL）
- 2019 高階関数プログラムのための時相論理（SAS）
- 2017–2025 高階不動点論理 HFL の一連の研究

注意 1. この二つは無関係ではない。線形論理は「資源をちょうど一度使う」という規律を型に持ち込む論理であり、小林の初期の仕事はそれで並行計算を書けるようにした。時相論理は「いつ何が起こるか」を述べる論理であり、後期の仕事はそれを型システムに翻訳して自動検証を可能にした。**論理を言語に入れる**ことから**論理を型に翻訳する**ことへ、という一貫した方法論として読める。

2　第 1 期（1992–1999）—- 線形論理と並行オブジェクト

2.1　ACL —- 線形論理を並行計算の基盤にする

出発点は、線形論理に基づく非同期通信モデルである（1992、1995）。これを並行論理プログラミングのパラダイムとしてまとめたのが **ACL**（A Concurrent Linear Logic Programming Paradigm, ILPS 1993）であり、さらに高階化したものが Higher-Order ACL である（Higher-Order Concurrent Linear Logic Programming, 1994/1995）。

線形論理を使う動機は明快である。並行計算においてメッセージは**一度送られて一度受け取られる**資源であり、線形論理の $\multimap$ と $\otimes$ はその規律をそのまま表す。論理式が通信の意味論を与え、証明が計算に対応する。

2.2　OOPSLA 1994 —- 並行オブジェクト指向 ＝ 並行計算 ＋ レコード

ご依頼の起点に最も近いのがこの論文である。中心となる見方は次の等式である。

$$\textit{concurrent object-oriented programming} = \textit{concurrent calculus} + \textit{record}$$

基礎となる並行計算として Higher-Order ACL を選び、それをレコード演算で拡張した体系に並行オブジェクト指向言語を**符号化**できることを示す。そして Higher-Order ACL が**多相型推論を備えた強い型システム**を持つため、この符号化を経由して並行オブジェクト指向言語のプログラムが**自動的に型検査できる**。

つまりこの論文は「並行オブジェクト指向言語に型推論を与える」ことを、言語に直接型システムを作り込むのではなく、**型付きの並行計算への翻訳**によって達成した。

2.3　基礎づけの整理と、解析・分散への展開

- 1995 **Naoki Kobayashi, Akinori Yonezawa**. Towards Foundations of Concurrent Object-Oriented Programming —- Types and Language Design. *Theory and Practice of Object Systems*.
- 1995 **Naoki Kobayashi, Motoki Nakade, Akinori Yonezawa**. Static Analysis of Communication for Asynchronous Concurrent Programming Languages. *SAS*.

- 1996 **Haruo Hosoya, Naoki Kobayashi, Akinori Yonezawa**. Partial Evaluation Scheme for Concurrent Languages and Its Correctness. *Euro-Par*.
- 1999 **Naoki Kobayashi, Toshihiro Shimizu, Akinori Yonezawa**. Distributed Concurrent Linear Logic Programming. *Theoretical Computer Science*.
- 1999 **Naoki Kobayashi, Akinori Yonezawa**. Distributed and concurrent objects based on linear logic. *FMOODS*.（招待講演）

型と言語設計の関係を整理し、通信の静的解析、部分評価、分散化へと広げている。この時期はすべて米澤明憲との共同研究であり、ABCL の系譜にある並行オブジェクト指向言語の理論的基礎づけという文脈に置かれる。

3　第 2 期（1996–2008）—- 型付きプロセス計算

論理と言語の対応から、**型システムで並行プログラムの性質を保証する**方向へ移る。対象は π 計算である。

3.1　線形性を π 計算へ

- 1996 **Naoki Kobayashi, Benjamin C. Pierce, David N. Turner**. Linearity and the Pi-Calculus. *POPL*.（1999 年に *TOPLAS*）

チャネルを「ちょうど一度使う」線形型を π 計算に導入する。線形論理の規律が、論理の側からではなく**型の側から**持ち込まれている。

3.2　デッドロック自由性を型で保証する

- 1997 **Naoki Kobayashi**. A Partially Deadlock-Free Typed Process Calculus. *LICS*.（1998 年に *TOPLAS*）
- 1998 **Eijiro Sumii, Naoki Kobayashi**. A Generalized Deadlock-Free Process Calculus. *HLCL*.
- 2000 **Naoki Kobayashi**. An Implicitly-Typed Deadlock-Free Process Calculus. *CONCUR*.
- 2002 . A Type System for Lock-Free Processes. *Information and Computation*.
- 2006 **Naoki Kobayashi**. A New Type System for Deadlock-Free Processes. *CONCUR*.
- 2007 . Type-Based Analysis of Deadlock for a Concurrent Calculus with Interrupts. *ESOP*.

一連の仕事は「型が付けばデッドロックしない」を、部分的な保証から出発して徐々に強め、型推論を暗黙化し、最後は割り込みを持つ計算にまで広げている。**型で活性（liveness）を保証する**という、当時としては踏み込んだ方向である。

注意 2. 本プロジェクトの AIPL$^-$ の健全性レポートが「デッドロック自由は一般には言えない —- `await` がある」と述べ、進行定理の第三の枝としてそれを明示したのは、この系統の仕事が**型で言えるところまで**を押し広げてきた歴史の反対側にいる。小林の型システムは、通信構造に制約を課すことでその枝を消しにいく。

3.3 汎用型システムと資源使用解析

- 2001 **Naoki Kobayashi**. A Generic Type System for the Pi-Calculus. *POPL*.（2004 年に *TCS*）
- 2001/2002/2005 **Naoki Kobayashi et al.**. Resource Usage Analysis. *APLAS 2001 / POPL 2002 / TOPLAS 2005*.

前者は、個別の性質ごとに型システムを作るのをやめ、**型システムを生成する枠組み**を与える。後者は「ファイルを開いたら閉じる」のような資源の使用順序を型で追う解析で、線形型の実用的な帰結である。

3.4 情報流と、モデル検査との最初の接触

- 2002 . Type-Based Information Analysis for Low-Level Languages. *APLAS*.
- 2008 . Linear Declassification. *ESOP*.
- 2006 **Hiroshi Unno, Naoki Kobayashi, Akinori Yonezawa**. Combining type-based analysis and model checking for finding counterexamples against non-interference. *PLAS*.

最後の一本が、次の時期への橋渡しになる。型に基づく解析は健全だが完全ではないので、「安全と言えない」と答えたときに、それが本物の欠陥なのか解析の粗さなのかが分からない。その判別に**モデル検査で反例を探す**という役割分担である。この時点ではまだ型とモデル検査は**別の道具**として併用されている。

4 第 3 期（2009–2016）—- 高階モデル検査は交差型の型検査である

ここで方向が変わる。型とモデル検査が併用される道具から、**同じ問題**になる。

4.1 出発点 —- Ong の決定可能性

- 2006 **C.-H. Luke Ong**. On Model-Checking Trees Generated by Higher-Order Recursion Schemes. *LICS*.

高階再帰スキーム（HORS）が生成する無限木の MSO 性質検査が決定可能であることが示された。ただし証明はゲーム意味論に基づき、実装できるアルゴリズムを与えなかった。

4.2 交差型による定式化

- 2009 **Naoki Kobayashi**. Types and Higher-Order Recursion Schemes for Verification of Higher-Order Programs. *POPL*.

安全性（trivial automata で表せる性質）について、性質 φ から交差型システムを構成し

$$G \text{ の生成する木が } \varphi \text{ を満たす} \quad \Longleftrightarrow \quad G \text{ がその型システムで型付く}$$

を示した。ゲーム意味論を経由しない、型理論の言葉による特徴づけである。

4.3 等価性の一般化 —- モーダル μ 計算

- 2009 **Naoki Kobayashi, C.-H. Luke Ong**. A Type System Equivalent to the Modal Mu-Calculus Model Checking of Higher-Order Recursion Schemes. *LICS*.
- 2009/2011 **Naoki Kobayashi, C.-H. Luke Ong**. Complexity of Model Checking Recursion Schemes for Fragments of the Modal Mu-Calculus. *ICALP 2009 / Logical Methods in Computer Science 2011*.

上の対応が安全性に留まらずモーダル μ 計算の全体に拡張された。**ここで初めて時相論理（モーダル μ 計算）が主題として現れる**。そしてこの結果により、「モデル検査を型推論で解く」と「型推論をモデル検査で解く」は同じ命題の言い換えになる。

4.4 等価性を実装に変える

等価性は決定手続きを与えるが、素朴に実装すると型の候補が爆発する。

- 2011 **Naoki Kobayashi**. A Practical Linear Time Algorithm for Trivial Automata Model Checking of Higher-Order Recursion Schemes. *FoSSaCS*.
- 2010 **Naoki Kobayashi, Naoshi Tabuchi, Hiroshi Unno**. Higher-order multi-parameter tree transducers and recursion schemes for program verification. *POPL*.
- 2011 **Naoki Kobayashi, Ryosuke Sato, Hiroshi Unno**. Predicate abstraction and CEGAR for higher-order model checking. *PLDI*.

n 階で n 重指数時間完全という最悪計算量にもかかわらず、実際のプログラムから生じる入力では線形に近く振る舞う —- という主張が実装可能性を開いた。そして述語抽象と CEGAR を高階へ持ち込むことで、整数を扱う高階関数プログラムの自動検証器が現実のものになった。

4.5 対象の拡張は型システムの拡張である

- 2010 **Takeshi Tsukada, Naoki Kobayashi**. Untyped Recursion Schemes and Infinite Intersection Types. *FoSSaCS*.
- 2013 **Naoki Kobayashi, Atsushi Igarashi**. Model-Checking Higher-Order Programs with Recursive Types. *ESOP*.
- 2015 **Naoki Kobayashi, Xin Li**. Automata-Based Abstraction Refinement for μHORS Model Checking. *LICS*.
- 2016 **Xin Li, Naoki Kobayashi**. Equivalence-Based Abstraction Refinement for μHORS Model Checking. *ATVA*.

等価性が確立すると、**型システムを広げることがモデル検査の対象を広げること**になる。無限交差型で型無し再帰スキームへ、再帰型を持つプログラムへ、と対象が広がる。いずれも「型システムを設計し、その型検査がモデル検査に等価であることを示す」という同じ筋で進む。

5　第 4 期（2016–2024）—- 時相論理と高階不動点論理 HFL

時相論理が、検証したい性質を書く言語として正面に出てくる。

- 2016 . Temporal verification of higher-order functional programs. *POPL*.
- 2017 . On the relationship between higher-order recursion schemes and higher-order fixpoint logic. *POPL*.
- 2018 . Higher-Order Program Verification via HFL Model Checking. *ESOP*.
- 2019 . A Temporal Logic for Higher-Order Functional Programs. *SAS*.
- 2019 . Temporal Verification of Programs via First-Order Fixpoint Logic. *SAS*.

高階不動点論理 HFL は、モーダル μ 計算を高階に拡張した論理である。2017 年の POPL 論文が HORS と HFL の関係を明らかにし、2018 年の ESOP 論文が「プログラム検証を HFL のモデル検査に還元する」という枠組みを与えた。ここでの発想は第 3 期と同型である —- **検証問題を、決定手続きを持つ別の問題へ翻訳する**。

5.1　HFL を解く

- 2019 **Youkichi Hosoi, Naoki Kobayashi, Takeshi Tsukada**. A Type-Based HFL Model Checking Algorithm. *APLAS*.
- 2019 **Ryosuke Sato, Naoki Iwayama, Naoki Kobayashi**. Combining higher-order model checking with refinement type inference. *PEPM*.
- 2019 . Reduction from branching-time property verification of higher-order programs to HFL validity checking. *PEPM*.
- 2020 . Predicate Abstraction and CEGAR for $\nu\mathrm{HFL}_{\mathbb{Z}}$ Validity Checking. *SAS*.
- 2020 . A New Refinement Type System for Automated $\nu\mathrm{HFL}_{\mathbb{Z}}$ Validity Checking. *APLAS*.
- 2023 . HFL(Z) Validity Checking for Automated Program Verification. *Proc. ACM Program. Lang.*.

PEPM 2019 の一本は本稿の文脈で特に重要である。**精製型（refinement type）の推論と高階モデル検査を組み合わせる**もので、「モデル検査を型推論の道具として使う」という読み方に文字どおり答える。逆向きに、モデル検査の前段として精製型推論で問題を小さくする使い方もされる。

5.2　論理そのものの研究へ

- 2020 . A Probabilistic Higher-Order Fixpoint Logic. *FSCD*.（2021 年に *LMCS*）
- 2021 . A Cyclic Proof System for $\mathrm{HFL}_{\mathbb{N}}$. *CSL*.
- 2020 . Fold/Unfold Transformations for Fixpoint Logic. *TACAS*.
- 2022 . Asynchronous Unfold/Fold Transformation for Fixpoint Logic. *FLOPS*.（2024 年に *Science of Computer Programming*）
- 2020 . On Average-Case Hardness of Higher-Order Model Checking. *FSCD*.
- 2021 . An Overview of the HFL Model Checking Project. *HCVS@ETAPS*.

確率的な拡張、循環証明系、変換規則、平均時計算量 —- 論理の側の研究へも広がる。2021 年の概説が、この時期のプロジェクト全体の見通しを与えている。

6 第 5 期（2020–2026）—- 実用検証系への展開

近年は、これらの理論を実際の言語と実際のソルバへ接続する仕事が目立つ。

6.1 制約付きホーン節（CHC）への還元

- 2020 . RustHorn: CHC-Based Verification for Rust Programs. *ESOP*.（2021 年に *TOPLAS*）
- 2021 . Symbolic Automatic Relations and Their Applications to SMT and CHC Solving. *SAS*.
- 2023 . Argument Reduction of Constrained Horn Clauses Using Equality Constraints. *APLAS*.
- 2023 . Higher-Order Property-Directed Reachability. *Proc. ACM Program. Lang.*.
- 2025 . Automated Catamorphism Synthesis for Solving Constrained Horn Clauses over Algebraic Data Types. *SAS*.

RustHorn は、Rust の所有権（ownership）による別名の無さを利用して、ポインタを持つプログラムの検証を CHC へ還元する。所有権型システムという**言語側の規律**が、検証の還元を可能にしている —- 線形性を型で扱ってきた第 2 期の系譜がここに現れる。

6.2 型システムの実用化

- 2020 . ConSORT: Context- and Flow-Sensitive Ownership Refinement Types for Imperative Programs. *ESOP*.
- 2022 . Parameterized Recursive Refinement Types for Automated Program Verification. *SAS*.
- 2023 . Gradual Tensor Shape Checking. *ESOP*.
- 2025 . On Decidable and Undecidable Extensions of Simply Typed Lambda Calculus. *Proc. ACM Program. Lang.*.
- 2025 . On the Relationship between Dijkstra Monads and Higher-Order Fixpoint Logic. *ESOP*.

Gradual Tensor Shape Checking は、機械学習のプログラムでテンソルの形（shape）が合わないという実際の誤りを、段階的型付け（gradual typing）で捉えようとするものである。理論の道具を現代の題材に向けた例といえる。

6.3 π 計算への回帰と、機械学習の利用

- 2021 . Termination Analysis for the π-Calculus by Reduction to Sequential Program Termination. *APLAS*.
- 2021 . Sized Types with Usages for Parallel Complexity of Pi-Calculus Processes. *CONCUR*.
- 2021 . Toward Neural-Network-Guided Program Synthesis and Verification. *SAS*.

π 計算が再び現れるが、扱う性質は停止性と並列計算量であり、道具は「逐次プログラムの問題への還元」である。また探索をニューラルネットで導く試みも始まっている。

7 通して見える方法論

三十余年を通して、手口は驚くほど一貫している。

M1 論理を計算に対応させる。線形論理と非同期通信（第 1 期）、モーダル μ 計算と交差型（第 3 期）、HFL とプログラム検証（第 4 期）。対応が付けば、一方の道具が他方で使える。

M2 型システムを設計し、その型検査が別の問題に等価であることを示す。これが第 3 期以降の中核である。「型が付く $\iff$ 性質を満たす」を示せば、型推論アルゴリズムがそのまま検証器になる。

M3 等価性を実装可能なアルゴリズムに変える。最悪計算量が絶望的でも、実際の入力での振る舞いを測って実用性を示す（FoSSaCS 2011）。抽象化が足りなければ CEGAR で洗練する（PLDI 2011）。

M4 対象を広げたいときは型システムを広げる。無限交差型、再帰型、精製型、所有権型、サイズ型 —- 拡張したい対象に応じて型の側を設計する。

もう一つ、**還元**という手口が全体を貫く。プログラム検証を HFL の妥当性検査へ、π 計算の停止性を逐次プログラムの停止性へ、Rust の検証を CHC へ。難しい問題を、解ける問題へ翻訳する。

8 年表（主要論文）

年	会議・誌	論文
1992	Parallel Symb. Comp.	Asynchronous Communication Model Based on Linear Logic
1993	ILPS	ACL —- A Concurrent Linear Logic Programming Paradigm
1994	OOPSLA	**Type-Theoretic Foundations for Concurrent Object-Oriented Programming**
1994	TPPP	Higher-Order Concurrent Linear Logic Programming
1995	TAPOS	Towards Foundations of Concurrent OO Programming —- Types and Language Design
1995	SAS	Static Analysis of Communication for Asynchronous Concurrent Languages
1996	POPL	**Linearity and the Pi-Calculus**（Pierce, Turner と）
1997	LICS	**A Partially Deadlock-Free Typed Process Calculus**
1999	POPL	Quasi-Linear Types
1999	TCS	Distributed Concurrent Linear Logic Programming
2000	CONCUR	An Implicitly-Typed Deadlock-Free Process Calculus
2001	POPL	**A Generic Type System for the Pi-Calculus**

年	会議・誌	論文
2002	POPL	**Resource Usage Analysis**（2005 TOPLAS）
2006	CONCUR	A New Type System for Deadlock-Free Processes
2006	PLAS	型に基づく解析とモデル検査の併用（Unno, Yonezawa と）
2007	ESOP	Type-Based Analysis of Deadlock for a Calculus with Interrupts
2009	POPL	**Types and Higher-Order Recursion Schemes for Verification**
2009	LICS	**A Type System Equivalent to the Modal Mu-Calculus Model Checking of HORS**（Ong と）
2010	FoSSaCS	Untyped Recursion Schemes and Infinite Intersection Types
2010	POPL	Higher-order multi-parameter tree transducers
2011	FoSSaCS	A Practical Linear Time Algorithm for Trivial Automata MC of HORS
2011	PLDI	**Predicate abstraction and CEGAR for higher-order model checking**
2013	ESOP	Model-Checking Higher-Order Programs with Recursive Types
2015	LICS	Automata-Based Abstraction Refinement for μHORS
2016	POPL	**Temporal verification of higher-order functional programs**
2017	POPL	On the relationship between HORS and higher-order fixpoint logic
2018	ESOP	**Higher-Order Program Verification via HFL Model Checking**
2019	PEPM	Combining higher-order model checking with refinement type inference
2019	SAS	A Temporal Logic for Higher-Order Functional Programs
2020	ESOP	**RustHorn: CHC-Based Verification for Rust Programs**
2020	ESOP	ConSORT: Ownership Refinement Types for Imperative Programs
2020	FSCD	A Probabilistic Higher-Order Fixpoint Logic
2021	CSL	A Cyclic Proof System for $\mathrm{HFL}_{\mathbb{N}}$
2021	SAS	Toward Neural-Network-Guided Program Synthesis and Verification
2023	PACMPL	HFL(Z) Validity Checking for Automated Program Verification
2023	ESOP	Gradual Tensor Shape Checking
2023	PACMPL	Higher-Order Property-Directed Reachability
2025	PACMPL	On Decidable and Undecidable Extensions of Simply Typed λ Calculus

年	会議・誌	論文
2025	ESOP	On the Relationship between Dijkstra Monads and HFL

9　本プロジェクト（AICE / AIPL）との接点

三点ある。

9.1　OOPSLA 1994 がやったことと、AIPL がやっていること

OOPSLA 1994 の主張は「並行オブジェクト指向言語を型付き並行計算へ符号化すれば、多相型推論によって自動的に型検査できる」であった。本プロジェクトが AIPL（ABCL^c+）に対して行っているのは、符号化ではなく**言語に直接**型推論を作り込むという別の道であるが、達成しようとしていることは同じ —- **並行オブジェクト指向言語のプログラムを自動で型検査する**。

同じ問題に対する二つのアプローチとして、比べる価値がある。符号化は理論的な保証が得やすいが、エラーメッセージが翻訳先の言葉になる。直接作り込む方は実装が素直だが、健全性を自分で示さなければならない。本プロジェクトが Coq で健全性を機械検証しているのは、後者の道の代償である。

9.2　2006 PLAS —- 型が「言えない」ときの役割分担

型に基づく解析は健全だが完全ではない。拒否されたときそれが本物の欠陥か解析の粗さかが分からない、という問題に対し、Unno–Kobayashi–Yonezawa はモデル検査で反例を探すという分業を提案した。

本プロジェクトが導入した**型と実行値の差分テスト**（型検査が付けた戻り値型と実行時に `reply` された値の型を突き合わせる）は、道具立てはずっと素朴だが役割分担としては同じ形である —- 型が言い切れないところを、実際の実行で埋める。実際にこれで整数演算が `float` を返していた不整合が見つかっている。

9.3　デッドロックの扱い

AIPL$^-$ の進行定理は「値・簡約・`await` 待ち」の三択であり、デッドロックは型では排除できないと明示している。小林の第 2 期の仕事はまさにその枝を型で消しにいくものであった（部分的デッドロック自由、ロックフリー、割り込みつき計算）。

つまり本プロジェクトの現状は、小林が 1997 年から取り組んだ問題の**入口**に立っている。通信構造に型で制約を課せば、`await` 待ちの枝を狭めていける —- という道筋が既に示されている。

10　本稿の限界

- 書誌は DBLP に依拠した。DBLP に収録されない国内論文・解説・著書は含まれていない。実際の業績はここに挙げたより広い。
- 内容の記述は、本文まで確認できたものと、題名・掲載場所から確実に言えることに限った。個々

の技術的な詳細には踏み込んでいない。

- 共著者は代表的なものだけを挙げた。省略した共著者が多数ある。
- 「時相論理を並列オブジェクト指向言語に導入した研究」に相当するものは DBLP の記録には見当たらなかった。第 1.2 節に述べたとおり、初期は線形論理である。もし別の文脈（国内発表など）を指しておられるなら、本稿はその部分を欠いている。

参考

書誌はすべて DBLP の `https://dblp.org/pid/k/NaokiKobayashi.xml`（Naoki Kobayashi 0001）で確認した。第 3 期以降の書誌は、本リポジトリの `survey_mc_typeinference.pdf`（モデル検査と型推論）と重なる。OOPSLA 1994 論文の内容は `https://dl.acm.org/doi/10.1145/191081.191088` を参照した。