

From Execution to Exploration:

*Bridging the Usability Gap in
Formal Natural Language Inference*

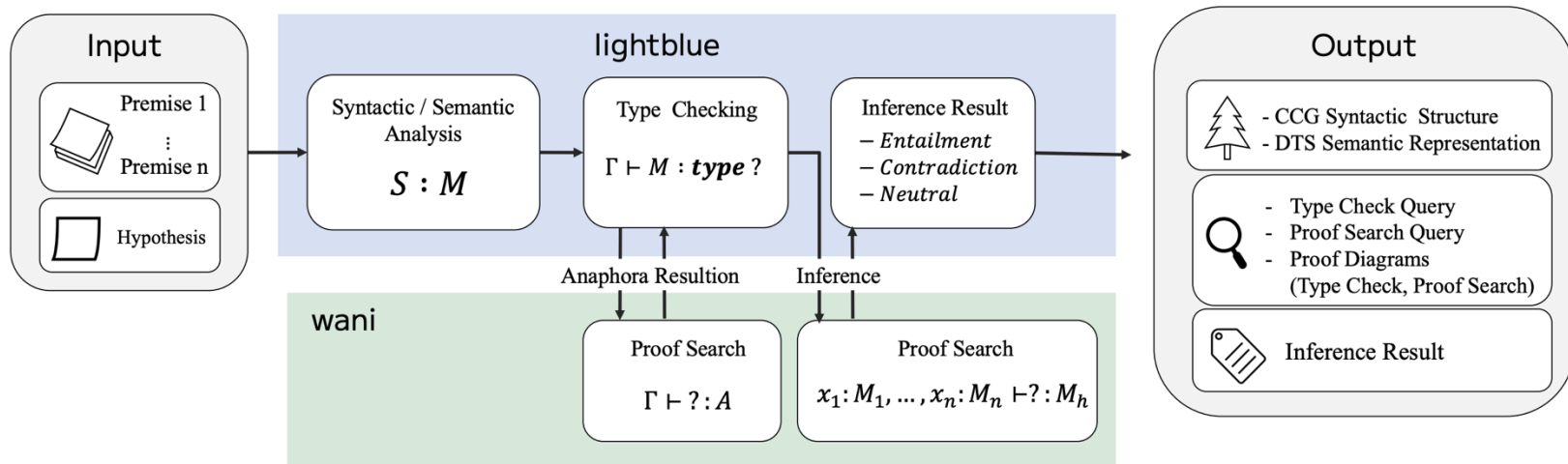
Koharu Saeki Daisuke Bekki

Ochanomizu University

BriGap-3 Université Paris Cité July 11, 2026

Our Natural Language Inference Pipeline

lightblue: a linguistically-oriented formal NLI system integrating **syntactic parsing**, **semantic composition**, **type checking**, and **proof search**.



Overview of the inference pipeline of *lightblue* (Tomita et al., 2025)

This is what a grammar developer sees today



The grammar developer must find the single path they care about **by hand** in a 3.7 MB file.

one full route: syntactic/semantic parsing → type-checking → proof search

Same problem, same engine — a different way of working

Today (exhaustive)	
260 paths	>120s backend time

With Express	
10 paths	12s backend time

120s is the timeout cap: under exhaustive execution the search never finished.

We did not optimize the parser or prover.

We changed **who controls the reasoning process** — and **when computation happens**.

Why we choose formal NLI — and the catch

The appeal

Formal NLI grounds inference in **linguistic theory**, logical forms, and theorem proving, making inference **transparent** and **verifiable**, with explicit intermediate representations.

The catch

The *rich linguistic representations* that buy this transparency come back, **at development time**, as a **branching explosion**.

Where the branches come from

lightblue (Tomita et al., 2025) ... CCG (Steedman, 2000) + DTS (Bekki & Mineshima, 2017) — three sequential stages:

- 1 **Syntactic parsing and semantic composition:** CYK chart parsing → multiple admissible syntactic structures.
- 2 **Type checking & anaphora resolution:** underspecified terms trigger proof search → multiple outcomes per structure.
- 3 **Proof search** wani (Daido & Bekki, 2020) tests entailment / contradiction.

Key point

Type checking is **sequential**. Each outcome is the context for the next sentence
⇒ ambiguity **compounds across sentences**.

Running example: active → passive

Problem B

Premise

Taro-ga Hanako-o shikat-ta.
(Taro scolded Hanako.)

Hypothesis

Hanako-ga shikar-are-ta.
(Hanako was scolded.)

Expected label: **Yes**



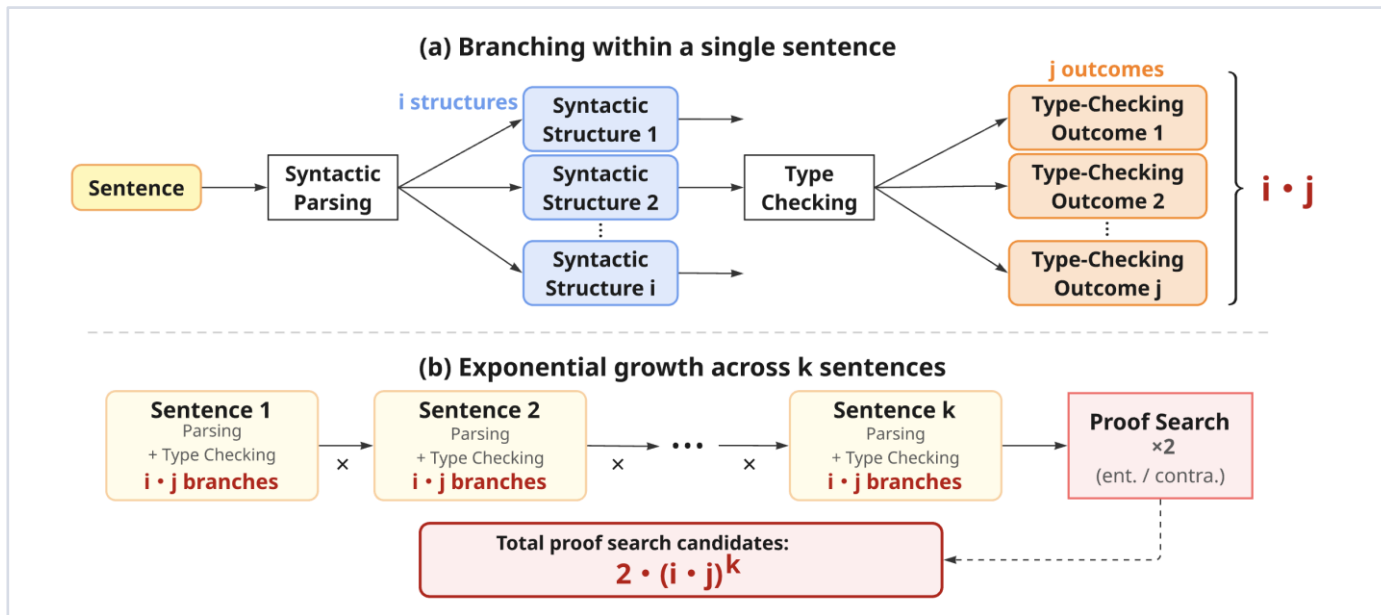
What has to be explored?

To verify the passive analysis, the developer must trace:

1. one syntactic structure
2. one type-checking outcome
3. the resulting proof-search query

Goal: trace **one theoretically interesting path**, not every possible analysis.

The branching problem

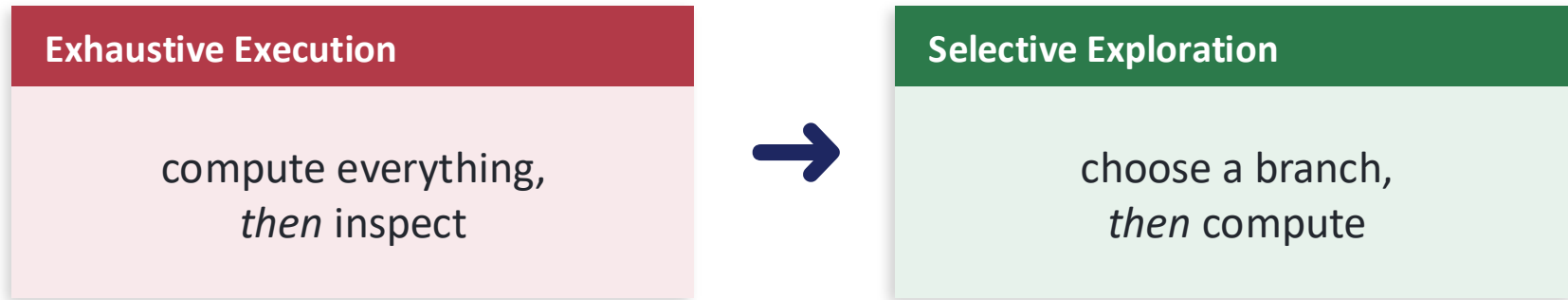


$2 \cdot (i \cdot j)^k$ proof search queries.

Even $k=2, i=j=4 \rightarrow$ **up to 512 queries** for a two-sentence problem.

The developer wants *one*.

The shift



Under exhaustive execution, the developer's judgment never enters until computation is already over.

The developer selects **one** candidate at each branching point.
Active paths never grow to $2 \cdot (i \cdot j)^k$ — they stay **manageable at all times**.

Investigate: localize to a substring

sentence: 学校まで太郎が走った。 太郎が走った 32 S[v:5:r<1>][term|attr<7>][+t<8>]NP[ni] | score: 0.98 候補: 32 件

Full sentence("Taro ran to school") Selected substring("Taro ran") 32 candidates

部分文字列「太郎が走った」の解析候補

"Taro ran"'s Syntactic Structure①

太郎が走った >Bx

S v:5:r:[1] \NP_{ni}
 term|attr:[7]

$\lambda x_0. \lambda k_0. \left[\begin{array}{c} x_1: \text{entity} \\ u_0: \text{走る/はしる/ガニ}(x_1, \text{太郎/たろう; 太郎/たろう}, x_0) \\ \left[\begin{array}{c} u_1: \# \text{タ}(x_1) \\ k_0(x_1) \end{array} \right] \end{array} \right]$

"Taro ran"'s Syntactic Structure②

太郎が走った >Bx

S v:5:r:[1] \NP_o
 term|attr:[7]

$\lambda x_0. \lambda k_0. \left[\begin{array}{c} x_1: \text{entity} \\ u_0: \text{走る/はしる/ガヲ}(x_1, \text{太郎/たろう; 太郎/たろう}, x_0) \\ \left[\begin{array}{c} u_1: \# \text{タ}(x_1) \\ k_0(x_1) \end{array} \right] \end{array} \right]$

"Taro ran"'s Syntactic Structure③

太郎が走った >Bx2

S v:5:r:[1] \NP_o \NP_{ni}
 term|attr:[7]

$\lambda x_0. \lambda x_1. \lambda k_0. \left[\begin{array}{c} x_2: \text{entity} \\ u_0: \text{走る/はしる/ガヲニ}(x_2, \text{太郎/たろう; 太郎/たろう}, x_1, x_0) \\ \left[\begin{array}{c} u_1: \# \text{タ}(x_2) \\ k_0(x_2) \end{array} \right] \end{array} \right]$

"Taro ran"'s Syntactic Structure④

太郎が走ったrel <B

N / MNP_{ni}

$\lambda x_0. \lambda x_1. \lambda x_2. \lambda k_0. \left[\begin{array}{c} x_3: \text{entity} \\ u_0: \left[\begin{array}{c} u_1: \text{走る/はしる/ガヲニ}(x_3, \text{太郎/たろう; 太郎/たろう}, x_2, x_0) \\ \# \text{タ}(x_3) \\ x_1(x_2)(k_0) \end{array} \right] \end{array} \right]$

CYK retains structures for every span. Select **"Taro ran"** from **"Taro ran to school"** → candidates appear instantly, **without re-running the pipeline.**

Explore: choose, then compute(1/2)

click to select this structure

Premise1: 太郎が花子を叱った。

score: 0.87
typecheck

太郎が花子を叱った。

WRAP

+

S_{decl}
 $x_0: \text{entity}$
 $u_0: \text{叱る/しかる/ガヲ}(x_0, \text{太郎/たろう}; \text{太郎/たろう}, \text{花子/はなこ})$
 $\# \text{タ}(x_0)$

score: 0.85
typecheck

太郎がヨ 花子を叱った。

WRAP

+

S_{decl}
 $x_0: \text{entity}$
 $u_0: \left[\begin{array}{l} x_1: \text{entity} \\ \text{花子}(x_1, x_0) \end{array} \right]$
 $x_2: \text{entity}$
 $u_2: \text{叱る/しかる/ガヲ}(x_2, \text{太郎/たろう}; \text{太郎/たろう}, \pi_1(u_0))$
 $\# \text{タ}(x_2)$

⋮

candidates ranked by score

Pick one syntactic structure → proceed to type checking.

Explore: choose, then compute(2/2)

Hypothesis: 花子が叱られた。

score: 0.87
typecheck

花子がpro叱られた。

WRAP

S_{decl}
 $x_0: \text{entity}$
 $u_0: \text{叱る/しかる/ガニ}(x_0, @ \text{entity}, \text{花子/はなこ})$
 $\# \text{タ}(x_0)$

the chosen type checking outcome

Selected

$s_0: \left[\left[\begin{array}{c} x_0: \text{entity} \\ u_0: \text{叱る/しかる/ガニ}(x_0, \text{太郎/たろう}; \text{太郎/たろう}, \text{花子/はなこ}) \\ \# \text{タ}(x_0) \end{array} \right] \vdash \left[\begin{array}{c} x_1: \text{entity} \\ u_2: \text{叱る/しかる/ガニ}(x_1, x_1, \text{花子/はなこ}) \\ \# \text{タ}(x_1) \end{array} \right] : \text{type}(\Sigma F) \right] +$

$s_0: \left[\left[\begin{array}{c} x_0: \text{entity} \\ u_0: \text{叱る/しかる/ガニ}(x_0, \text{太郎/たろう}; \text{太郎/たろう}, \text{花子/はなこ}) \\ \# \text{タ}(x_0) \end{array} \right] \vdash \left[\begin{array}{c} x_1: \text{entity} \\ u_2: \text{叱る/しかる/ガニ}(x_1, \pi_1(s_0), \text{花子/はなこ}) \\ \# \text{タ}(x_1) \end{array} \right] : \text{type}(\Sigma F) \right] +$

$\vdots$

then proceed to proof search

Pick one syntactic structure $\rightarrow$ one type-checking outcome $\rightarrow$ proof search.

One candidate per stage, per sentence $\Rightarrow$ no exponential blow-up.

Share: export a reproducible query

Premise 1

太郎はりんごとブロッコリーを食べる。

"Taro eats apples and broccoli"

Hypothesis

太郎は果物と野菜を食べる。

"Taro eats fruits and vegetables"

Query (pos)

保存

Button: "Save"

$$\begin{array}{c}
 s_0: \left[\begin{array}{c} u_0: \left[\begin{array}{c} x_0: \text{entity} \\ x_1: \text{entity} \\ \text{りんご}(x_1, x_0) \end{array} \right] \\ \left[\begin{array}{c} x_2: \text{entity} \\ \text{食べる/たべる/ガヲニ}(x_2, \text{太郎/たろう}; \text{太郎/たろう}, \pi_1(u_1), x_2) \end{array} \right] \end{array} \right] \\
 \left[\begin{array}{c} u_4: \left[\begin{array}{c} x_3: \text{entity} \\ x_4: \text{entity} \\ \text{ブロッコリー}(x_4, x_3) \end{array} \right] \\ \left[\begin{array}{c} x_5: \text{entity} \\ \text{食べる/たべる/ガヲニ}(x_5, \text{太郎/たろう}; \text{太郎/たろう}, \pi_1(u_4), x_5) \end{array} \right] \end{array} \right]
 \end{array}
 \vdash ? :
 \begin{array}{c}
 u_7: \left[\begin{array}{c} u_8: \left[\begin{array}{c} x_6: \text{entity} \\ x_7: \text{entity} \\ \text{果物}(x_7, x_6) \end{array} \right] \\ \left[\begin{array}{c} x_8: \text{entity} \\ \text{食べる/たべる/ガヲニ}(x_8, \text{太郎/たろう}; \text{太郎/たろう}, \pi_1(u_8), x_8) \end{array} \right] \end{array} \right] \\
 \left[\begin{array}{c} u_{11}: \left[\begin{array}{c} x_9: \text{entity} \\ x_{10}: \text{entity} \\ \text{野菜}(x_{10}, x_9) \end{array} \right] \\ \left[\begin{array}{c} x_{11}: \text{entity} \\ \text{食べる/たべる/ガヲニ}(x_{11}, \text{太郎/たろう}; \text{太郎/たろう}, \pi_1(u_{11}), x_{11}) \end{array} \right] \end{array} \right]
 \end{array}$$

The interactively formulated proof-search query is exported in a format **natively loadable by wani** — the prover developer reproduces the failure instantly.

Search space: the number we promised

Two NLI problems, $k=2$, $i=4$, $j=4 \rightarrow$ up to **512 queries** under exhaustive execution.

	Exhaustive — Paths	Time	Express — Paths	Time
Problem A	260	>120s	10	12s
Problem B	128	>120s	28	94s

Problem A: **96% fewer paths** ($260 \rightarrow 10$).

Problem B: exhaustive never reached the target query; Express did, in 28 paths.

Collaboration becomes possible

“Taro eats apples and broccoli” $\models$ “Taro eats fruits and vegetables”
→ **Unknown** (expected **Yes**).

- 1 **Grammar developer** narrows down to the problematic query in Express.
- 2 **Grammar developer** exports it in wani-loadable format and shares it.
- 3 **wani developer** reproduces the failure and pinpoints the root cause in minutes.

Result

Previously: manual transcription from screenshots (~1 hour). **$\approx 12\times$ faster debugging**, across developer roles.

The broader bridge

Formal NLI bridges **formal linguistics** and **computational linguistics**— theory made explicit and testable.

Why it matters

If the development workflow of the systems that embody this bridge stays impractical, **the bridge cannot be crossed**. By making that workflow tractable and transparent, *Express* keeps the connection practically sustainable.

Acknowledgments

This work was supported in part by JST CREST Grant Number JPMJCR2565 and JSPS KAKENHI Grant Number JP23H03452, Japan.