RETICULATE: Session Types as Algebraic Reticulates

Lattice Verification for Session Types with Parallel Composition

Alexandre Zua Caldeira Vasco T. Vasconcelos

LASIGE, Faculdade de Ciências
Universidade de Lisboa

2026

Outline

1. Motivation
2. Session Type Syntax
3. The Reticulate Tool
4. Benchmarks
5. Self-Reference & New Benchmarks
6. Web Demo
7. Conclusion

The Problem

Session types for objects

A session type describes the valid protocol for interacting with an object: which methods to call, in what order, under what conditions.

Limitation: existing systems assume *linear ownership* — exactly one client at a time.

Reality: concurrent access is everywhere:

- File I/O: concurrent readers and writers
- AI agents: tool calls $\|$ notifications (MCP)
- CI/CD: parallel lint, test, build jobs
- WebSockets: send $\|$ receive

The Insight

Parallel composition ⇒ lattice structure

Adding a *parallel constructor* $S_1 \parallel S_2$ to session types forces the state space to form a **lattice**.

Without $\parallel$:
State space is a tree/DAG.
Lattice structure is trivial.

With $\parallel$:
State space becomes a *product lattice*.
Meets = forks, joins = synchronizations.

$$L(S_1 \parallel S_2) \;=\; L(S_1) \times L(S_2)$$

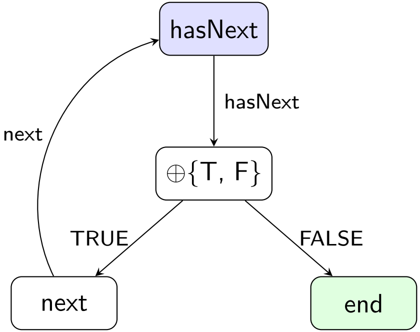
Grammar

$$
\begin{array}{rcll}
S & ::= & \&\{m_1 : S_1,\ \ldots,\ m_n : S_n\} & \text{branch (external choice)} \\
 & \mid & \oplus\{l_1 : S_1,\ \ldots,\ l_n : S_n\} & \text{selection (internal choice)} \\
 & \mid & S_1 \parallel S_2 & \textbf{parallel composition} \\
 & \mid & \mu X \,.\, S & \text{recursion} \\
 & \mid & X & \text{type variable} \\
 & \mid & \textbf{end} & \text{terminated} \\
 & \mid & S_1 \,.\, S_2 & \text{sequencing (sugar)}
\end{array}
$$

Key novelty: the $\parallel$ constructor — two concurrent sub-protocols on the same object, free interleaving until both reach **end**.

Example: Java Iterator

```
rec X . &{hasNext: ⊕{TRUE: &{next: X}, FALSE: end}}
```



- 4 states, 4 transitions
- 2 SCCs (cycle from recursion)
- Lattice: ✓
- Recursion visible as back-edge

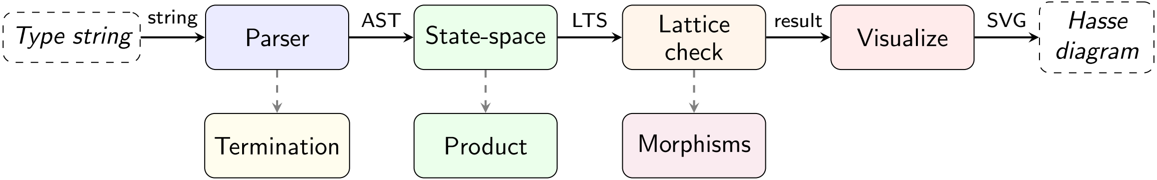
Example: Two-Buyer (with ∥)

```
lookup . getPrice . (proposeA . end ∥proposeB .
  ⊕{ACCEPT: pay . end, REJECT: end})
```

- Two buyers propose concurrently (∥).
- Buyer B may accept and pay, or reject.
- Product lattice: $2 \times 4 = 8$ states in the parallel region.
- **10 states, 14 transitions, lattice:** ✓

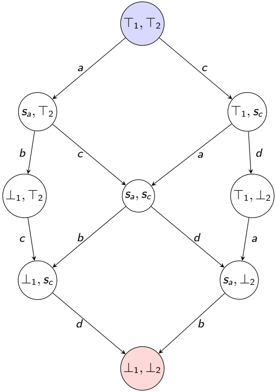
The ∥ constructor models the real concurrency: `proposeA` and `proposeB` may happen in any order.

Architecture



- 8 modules, **499 tests**, 17 benchmarks
- Python 3.12, pure library + CLI + web demo
- Product construction for $\parallel$ verified empirically on all benchmarks

Product Lattice



$a\,.\,b\,.\,\textbf{end} \;\|\; c\,.\,d\,.\,\textbf{end}$
$\Rightarrow\; 3 \times 3 = 9$ state product lattice.

- Top $= (\top_1, \top_2)$: fork point
- Bottom $= (\bot_1, \bot_2)$: join point
- Componentwise meets and joins

Lattice Verification

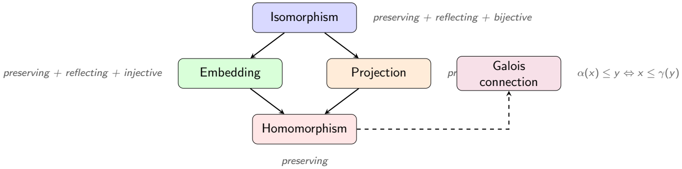
Given a state space $L(S)$, check the reachability poset:

1. **Quotient by SCCs** (Tarjan's) — handles cycles from μ
2. **Compute reachability** — topological-order DP
3. **Check four conditions:**

Condition	Meaning
Top reachable	$\top$ reaches all states
Bottom reachable	All states reach $\bot$
All meets exist	$\forall\, a, b :\ a \wedge b$ exists
All joins exist	$\forall\, a, b :\ a \vee b$ exists

Theorem: If $L(S_1)$ and $L(S_2)$ are lattices, then $L(S_1 \parallel S_2)$ is a lattice.

Morphism Hierarchy


Isomorphism
preserving + reflecting + bijective
preserving + reflecting + injective
Embedding
Projection
pr
Galois connection
$\alpha(x) \leq y \Leftrightarrow x \leq \gamma(y)$
Homomorphism
preserving

- Structure-preserving maps between state spaces
- **Isomorphism**: protocol equivalence
- **Embedding**: one protocol refines another
- **Galois connection**: abstraction–concretization (Cousot & Cousot)

17 Benchmark Protocols

#	Protocol	$\|Q\|$	$\|\delta\|$	SCCs	$\|$
1	Java Iterator	4	4	2	
2	File Object	5	5	4	
3	SMTP	7	8	4	
4	HTTP Connection	5	7	3	
5	OAuth 2.0	7	11	6	
6	Two-Buyer	10	14	10	✓
7	MCP (AI agent)	7	17	5	✓
8	A2A (Google agent)	5	7	4	
9	File Channel	7	12	7	✓
10	ATM	7	11	6	
11	Reentrant Lock	6	7	2	
12	WebSocket	6	15	6	✓
13	DB Transaction	5	9	3	
14	Pub/Sub	4	7	3	
15	DNS Resolver	4	8	2	
blue!8 16	**Reticulate Pipeline**	**14**	**18**	**14**	✓
blue!8 17	**GitHub CI Workflow**	**11**	**13**	**11**	✓

All 17 benchmarks produce lattices. 6 use $\|$.

Self-Reference: The Reticulate Pipeline

The reticulate web demo's own analysis pipeline as a session type:

```
input . parse .
  ⊕{OK: buildStateSpace .
     ⊕{OK: (checkLattice . checkTermination
             . checkWFPar . end
             ‖ renderDiagram . end)
            . returnResult . end,
        ERROR: end},
     ERROR: end}
```

- 14 states, 18 transitions
- Lattice: ✓
- WF-Par: ✓
- Terminating: ✓

The ‖ models real concurrency:

- Left: verification checks
- Right: diagram rendering
- Independent on shared state space

GitHub CI Workflow

A CI/CD pipeline — parallel jobs, branching on success/failure:

```
trigger . checkout . setup .
  (lint . end ∥ test . end) .
  ⊕{PASS: deploy . ⊕{OK: end,
                    FAIL: rollback . end},
    FAIL: end}
```

- 11 states, 13 transitions
- Lattice: ✓
- WF-Par: ✓
- Terminating: ✓

Natural ∥ usage:

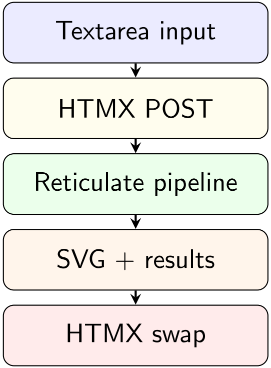
- `lint` and `test` run in parallel
- Fork-join before deployment
- Rollback on deploy failure

Session types capture the pipeline contract that developers already follow.

Interactive Web Tool

Stack: FastAPI + HTMX

- Enter any session type string
- Select from 17 benchmark protocols
- Instant results:
  - Pretty-printed type
  - State count, transitions, SCCs
  - Lattice verdict (4 sub-checks)
  - Termination & WF-Par status
  - Interactive SVG Hasse diagram
- Server-side rendering, zero JS overhead



Textarea input

HTMX POST

Reticulate pipeline

SVG + results

HTMX swap

Summary

Contributions

1. The $\|$ constructor for session types on objects — forces lattice structure
2. Reticulate: 8 modules, 499 tests, 17 benchmarks
3. Morphism hierarchy (iso $\subset$ embedding $\subset$ projection $\subset$ homomorphism) $+$ Galois connections
4. Web demo for instant, interactive analysis
5. Self-referential validation: the tool analyzes its own pipeline

Key result

All 17 benchmark protocols produce lattices — from classic patterns (Iterator, SMTP) through modern AI protocols (MCP, A2A) to the tool's own pipeline.

Future Work

- **Bica Reborn:** Java annotation-based session type checker, using RETICULATE as the verification back-end
- **Thread safety:** concurrency-level checking for methods in $\|$ branches (requires bytecode analysis)
- **Subtyping:** extending the lattice framework to session type subtyping with $\|$
- **Bisimulation:** connecting morphisms to behavioral equivalences on session types
- **Nested parallelism:** $\|$ inside $\|$ branches

Thank you

Alexandre Zua Caldeira

LASIGE / FCUL, Universidade de Lisboa

azcaldeira@fc.ul.pt

Try it: `uvicorn web.app:app`

Reticulate — session types as algebraic reticulates