AICE: A Typed Actor-based Intelligent Computing Environment

Yasushi Kodama
Faculty of Business Administration
Hosei University

Abstract

We present AICE (Actor-based Intelligent Computing Environment), a lightweight and distributed AI execution framework built on a typed actor language AIPL. AICE integrates asynchronous message passing, selective receive, and AI agent composition under a statically typed setting.

We introduce a three-layer type system consisting of value types, actor types, and communication types, and show that type safety alone is insufficient to guarantee correctness of asynchronous communication.

We formalize the operational semantics of selective receive and propose an optional integration of session types for protocol-level guarantees.

1 Introduction

Modern AI systems require distributed, concurrent, and lightweight execution environments. Existing systems lack formal guarantees and are often too heavy for edge or embedded deployment.

This paper proposes AICE, a typed actor-based computing environment built on AIPL.

Contributions

- A typed actor language AIPL
- AICE architecture for AI execution
- Formal operational semantics for asynchronous messaging
- Analysis of limitations of type safety

- Optional session type integration

2 Language Overview

2.1 Syntax

$$
\begin{aligned}
e &::= x \mid n \mid e_1 + e_2 \mid \mathit{new}\ C \mid \mathit{send}\ e.m(e^*) \\
s &::= e \mid s_1; s_2 \mid \mathit{if}\ e\ \mathit{then}\ s_1\ \mathit{else}\ s_2 \mid \mathit{select}\ \{b_i\} \\
b &::= \mathit{when}\ m(x^*) \to s
\end{aligned}
$$

3 Type System

3.1 Types

$$\tau ::= \mathit{int} \mid \mathit{string} \mid \mathit{unit} \mid \tau_1 \to \tau_2 \mid \mathit{actor}(C)$$

Extended form:

$$\mathit{actor}(C, P)$$

3.2 Typing Judgement

$$\Gamma \vdash e : \tau \quad \Gamma \vdash s : \mathit{unit}$$

3.3 Send Typing

$$\frac{\Gamma \vdash e : \mathit{actor}(C) \quad m : \tau_1 \times \cdots \times \tau_n \to \mathit{unit} \in C \quad \Gamma \vdash e_i : \tau_i}{\Gamma \vdash \mathit{send}\ e.m(e_1, \ldots, e_n) : \mathit{unit}}$$

3.4 Select Typing

$$\frac{\Gamma \vdash s_i : \mathit{unit} \quad \forall i}{\Gamma \vdash \mathit{select}\ \{b_i\} : \mathit{unit}}$$

4 Operational Semantics

4.1 Configuration

$$\langle A, M, s \rangle$$

4.2 Send

$$\langle A, M, send\ B.m(v)\rangle \to \langle A, M, unit\rangle$$

Mailbox update:

$$MB_B := MB_B \cup \{(m, v, A)\}$$

4.3 Select

$$\frac{(m, v, sender) \in M}{\langle A, M, select\rangle \to \langle A, M \setminus \{m\}, s\rangle}$$

4.4 Blocking

If no message matches:

$$\langle A, M, select\rangle \to wait$$

4.5 Non-determinism

Multiple matches:

$$select \to s_1 \quad \text{or} \quad s_2$$

5 Limitations of Type Safety

We show that type safety does not guarantee communication correctness.

5.1 Example

```
send calc.add(3,4);
send calc.result(10);
```

Even if well-typed, execution order is not guaranteed.

6 Session Types (Optional)

6.1 Protocol

$$P = A \to B : m_1; B \to A : m_2$$

6.2 Design

Session types are optional and do not interfere with HM inference.

7 AICE Architecture

- Actor runtime
- Scheduler
- Mailbox system
- Network layer
- AI actors

7.1 AI Actor Roles

- PlannerActor
- LLMActor
- MemoryActor
- ToolActor

8 Case Study

8.1 Calculator

```
send calc.add(3,4)
```

8.2 AI Pipeline

$$User \rightarrow Planner \rightarrow LLM \rightarrow Memory$$

9 Discussion

Key insight:

$$\text{Value typing} \neq \text{Communication correctness}$$

10 Related Work

Actor systems, session types, and AIOS frameworks.

11 Conclusion

We proposed AICE and demonstrated that type safety alone is insufficient for asynchronous actor systems.

Future work includes protocol inference, formal verification, and deployment on embedded systems.