A Type System for Genesis: Structural Modes Beyond the Turing Machine

Anonymous

Abstract

This paper presents a type system for Genesis, a structural extension of the Turing machine augmented with four primitive operators: $\Box$, $\triangle$, $\circ$, and $\ldots$. Unlike traditional type systems, Genesis types classify both values and modes of computation. We define the typing judgments, modal kinds, subtyping relations, and safety properties of the system, and show that it is sound, stratified, and erasable to a classical Turing-complete core.

1 Introduction

Type systems traditionally enforce correctness properties over values and program structure. In Genesis, the objective is broader: to make explicit the structural mode under which computation occurs. Stability, generativity, reflection, and open-ended execution are treated as first-class concerns. This paper introduces a type system that captures these concerns while preserving classical notions of soundness.

2 Typing Judgments

Typing judgments in Genesis take the form

$$\Gamma \vdash e : \tau \ @\ \kappa$$

where Γ is a typing context, e an expression, τ a value type, and κ a computational mode. The annotation κ records the structural conditions under which e is evaluated.

3 Computational Modes

Definition 1 (Modes)**.** *The set of computational modes is*

$$\kappa ::= \Box \mid \triangle \mid \circ \mid \ldots$$

corresponding to stability, generativity, reflection, and open continuation.

Modes are partially ordered:

$$\Box \leq \triangle \leq \circ \leq \ldots$$

This order governs safe coercions between modes.

4 Value Types

Value types are defined inductively as

$$\tau ::= \texttt{Unit} \mid \texttt{Bool} \mid \texttt{Nat} \mid \tau_1 \to \tau_2 \mid \Box\tau \mid \triangle\tau \mid \circ\tau \mid \ldots\tau$$

Modal type constructors express structural guarantees attached to values.

5 The Stability Type $\Box$

Definition 2 (Stable Type)**.** *A value of type $\Box\tau$ is invariant across computation steps.*

Lemma 1. *The $\Box$ type constructor is idempotent and supports weakening.*

Stable values may be used wherever values of type τ are expected, but not conversely.

6 The Generative Type $\triangle$

Definition 3 (Generative Type)**.** *A value of type $\triangle\tau$ may evolve during execution, possibly changing its internal structure while preserving type correctness.*

Proposition 1. *Values of type $\triangle\tau$ cannot be implicitly coerced to $\Box\tau$.*

This restriction prevents unsound assumptions of stability.

7 The Reflective Type $\circ$

Definition 4 (Reflective Type)**.** *A value of type $\circ\tau$ may inspect or modify its own definition or execution context.*

Reflection is stratified by depth, written $\circ^n\tau$, ensuring that no expression reflects upon itself at the same level.

8 The Open Type $\ldots$

Definition 5 (Open Continuation Type)**.** *A value of type $\ldots\tau$ denotes a computation that may not terminate but must remain productive.*

Lemma 2. *$\ldots\tau$ is not equivalent to divergence; it admits observable progress.*

9 Function Types and Mode Flow

Function types are annotated with modes:

$$\tau_1@\kappa_1 \to \tau_2@\kappa_2$$

Proposition 2. *Arguments may flow from lower to higher modes, but not from higher to lower modes.*

This enforces structural safety across function application.

10 Subtyping and Coercion

Genesis supports a limited subtyping relation:

$$\Box\tau \leq \tau \leq \triangle\tau \leq \circ\tau \leq \ldots\tau$$

Subtyping preserves type safety while allowing structural flexibility.

11 Type Soundness

Theorem 1 (Preservation). *If $\Gamma \vdash e : \tau@\kappa$ and $e \to e'$, then $\Gamma \vdash e' : \tau@\kappa'$ with $\kappa \leq \kappa'$.*

Theorem 2 (Progress). *If $\emptyset \vdash e : \tau@\kappa$, then either e is a value, e steps, or $\kappa = \ldots$ and e is productive.*

12 Erasure and Conservativity

Theorem 3 (Erasure). *Removing all modal annotations yields a well-typed program in a standard Turing-complete core language.*

This establishes that the Genesis type system adds structural meaning without increasing computational power.

13 Conclusion

The Genesis type system demonstrates that types can classify not only values but modes of existence within computation. By enforcing stability, generativity, reflection, and openness at the type level, Genesis provides a foundation for reasoning about modern computational systems while remaining faithful to classical theory.