Typed Program Examples in Genesis

Anonymous

Overview

This document presents illustrative, typed program examples for *Genesis*, a structural extension of the Turing machine. Each example demonstrates the interaction between program structure and the four primitive operators: $\Box$ (stability), $\triangle$ (generativity), $\circ$ (reflection), and $\ldots$ (open continuation). Typing judgments are provided to clarify the intended semantics.

1 Example 1: Stable Constant

```
let x =  42
```

$$\vdash 42 : \texttt{Nat}@\Box \qquad \vdash x : \Box\texttt{Nat}@\Box$$

The value `x` is invariant across all computation steps.

2 Example 2: Generative Structure

```
let model =  init_model()
```

$$\vdash \texttt{init\_model()} : \tau@\triangle \qquad \vdash \texttt{model} : \triangle\tau@\triangle$$

The internal structure of `model` may evolve during execution.

3 Example 3: Forbidden Coercion

```
let stable_model :  Model =  init_model()
```

$$\triangle\texttt{Model} \not\leq \Box\texttt{Model}$$

This program is rejected because generative values cannot be treated as stable.

4 Example 4: Mode-Sensitive Function

```
fun f(x : Nat @ ) : Nat @  =
    (x + 1)
```

$$f : (\texttt{Nat}@\Box) \to (\texttt{Nat}@\triangle)$$

Stable inputs may safely flow into generative contexts.

5 Example 5: Reflective Interpreter

```
let self_interp =  interpret(delta)
```

$$\vdash \texttt{self\_interp} : \circ^1 \tau @ \circ^1$$

The computation may inspect or modify its own transition rules.

6 Example 6: Stratified Reflection

```
let meta_interp =  self_interp
```

$$\vdash \texttt{meta\_interp} : \circ^2 \tau @ \circ^2$$

Each application of $\circ$ increases reflection depth, preventing paradoxical self-reference.

7 Example 7: Open-Ended Server

```
let server =
    loop {
      receive request;
      send response;
   }
```

$$\vdash \texttt{server} : \ldots\, \texttt{Unit}@\ldots$$

The computation does not halt but remains productive.

8 Example 8: Illegal Use of Open Value

```
let result : Nat = server
```

$$\ldots\texttt{Unit} \not\leq \texttt{Nat}$$

Open-ended computations cannot be coerced into completed values.

9 Example 9: Stable Configuration with Open Execution

```
let config =  load_config()

let service =
    run(config)
```

$$\vdash \texttt{config} : \Box\,\texttt{Config}@\Box \qquad \vdash \texttt{service} : \ldots\,\texttt{Unit}@\ldots$$

Stable data may parameterize open-ended execution.

10 Example 10: Full Structural Composition

```
let seed =  initial_state
let system =  evolve(seed)
let agent =  system
let world =  interact(agent)
```

$$\begin{gathered}\vdash \texttt{seed} : \Box\tau \\ \vdash \texttt{system} : \triangle\tau \\ \vdash \texttt{agent} : \circ^1\tau \\ \vdash \texttt{world} : \ldots\tau\end{gathered}$$

This example exercises all four structural operators in ascending order:

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

Conclusion

These examples demonstrate how the Genesis type system enforces structural discipline over computation. Programs are classified not only by what they compute, but by how they persist, evolve, reflect, and continue.