StarForth Ontology: Formal Conceptual Framework for Thermodynamically-Inspired Adaptive Virtual Machine

StarForth Project

Version 1.0 – December 13, 2025

Abstract

This document establishes a formal ontology, taxonomy, and lexicon for the StarForth adaptive runtime system. We provide precise mathematical definitions to eliminate ambiguity in academic discourse, explicitly distinguishing metaphorical concepts (thermodynamic analogies) from literal implementations (frequency counters, exponential decay functions). This framework enables reproducible research and rigorous peer review.

Contents

1 Introduction

The StarForth adaptive runtime employs concepts from thermodynamics, dynamical systems theory, and statistical inference. To avoid imprecise terminology, this ontology provides:

- **Formal definitions** with mathematical notation
- **Taxonomic hierarchy** of system components
- **Explicit scope limitations** (what is/isn't claimed)
- **Usage guidelines** for academic writing

2 Core Lexicon

2.1 Primary Measurable Quantities

Execution Frequency f

Count of times a dictionary entry has been executed, potentially adjusted by decay.

$$f(t) = f_0 \cdot e^{-\lambda t} \tag{1}$$

where f_0 is initial frequency and λ is decay coefficient.

Measurement: Unsigned 64-bit integer (`uint64_t execution_heat`).

Note: "Heat" is metaphorical naming; actual quantity is frequency.

Decay Coefficient λ

Rate parameter controlling exponential reduction in execution frequency.

$$\lambda \in \mathbb{R}^+, \quad \text{units: [1/time]} \tag{2}$$

Measurement: Derived via exponential regression; stored as Q48.16 fixed-point.

Window Size w

Number of recent execution events retained in rolling buffer.

$$w \in [w_{\min}, w_{\max}], \quad w \in \mathbb{N} \tag{3}$$

Measurement: Configurable parameter (default: 4096).

2.2 Derived Metrics

Transition Probability $P(B|A)$

Conditional probability that word B executes immediately after word A.

$$P(B|A) = \frac{\text{count}(A \to B)}{\text{count}(A)} \tag{4}$$

Coefficient of Variation CV

Normalized measure of dispersion.

$$CV = \frac{\sigma}{\mu} \tag{5}$$

where σ is standard deviation and μ is mean.

Convergence criterion: $CV \to 0$ as $t \to \infty$.

Variance Inflection Point w^*

Window size where variance begins to increase when window shrinks.

$$w^* = \arg \min_{w \in [w_{\min}, w_{\text{current}}]} \text{Var}(w) \tag{6}$$

3 Mathematical Formalism

3.1 Frequency Evolution Model

Discrete-time update:

$$f[t+1] = f[t] + \Delta_{\text{exec}}[t] - \Delta_{\text{decay}}[t] \tag{7}$$

where

$$\Delta_{\text{exec}}[t] = \begin{cases} 1 & \text{if word executed at tick } t \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

$$\Delta_{\text{decay}}[t] = \lambda \cdot f[t] \cdot \Delta t \tag{9}$$

Continuous-time model:

$$\frac{df}{dt} = r(t) - \lambda f(t) \tag{10}$$

where $r(t)$ is execution rate [executions/second].

Solution:

$$f(t) = e^{-\lambda t} \left[f_0 + \int_0^t r(\tau) e^{\lambda \tau} d\tau \right] \tag{11}$$

3.2 Hot-Words Cache Selection

Dictionary entry e is cached if and only if:

$$\operatorname{rank}(e) \leq K \tag{12}$$

where

$$\operatorname{rank}(e) = |\{e' \in \text{Dictionary} : f(e') > f(e)\}| + 1 \tag{13}$$

and K is cache size (constant).

3.3 Window Width Inference

Objective: Find optimal window size w^* minimizing variance.

Method: Binary search with Levene's test for variance homogeneity.

Algorithm 1 Variance Inflection Point Search

```
Initialize: $w_{\text{low}} \leftarrow w_{\min}, w_{\text{high}} \leftarrow w_{\text{current}}$
while $w_{\text{high}} - w_{\text{low}} > \epsilon$ do
    $w_{\text{mid}} \leftarrow \lfloor (w_{\text{low}} + w_{\text{high}})/2 \rfloor$
    Compute Var($w_{\text{mid}}$) from trajectory
    Perform Levene's test: $F \leftarrow$ Levene($w_{\text{mid}}, w_{\text{current}}$)
    if $p$-value($F$) $< \alpha$ then
        $w_{\text{low}} \leftarrow w_{\text{mid}}$                ▷ Variance increased, search right
    else
        $w_{\text{high}} \leftarrow w_{\text{mid}}$                ▷ Variance stable, search left
    end if
end while
return $w^* \leftarrow w_{\text{mid}}$
```

3.4 Decay Slope Inference

Given: Time series $\{(t_i, f_i)\}_{i=1}^{N}$ from rolling window.

Model: Exponential decay

$$f(t) = f_0 \cdot e^{-\lambda t} \tag{14}$$

Log-transform:

$$\ln(f) = \ln(f_0) - \lambda t \tag{15}$$

Least squares solution:

$$\lambda^* = \arg\min_{\lambda} \sum_{i=1}^{N} [\ln(f_i) - (\ln(f_0) - \lambda t_i)]^2 \tag{16}$$

Closed-form solution via linear regression on log-transformed data.

4 Taxonomy

4.1 Component Hierarchy

1. **Measurement Layer**
    - Execution Frequency Tracking
    - Temporal Recording (Rolling Window)
    - Transition Tracking (Word-to-Word)
2. **Transformation Layer**
    - Decay Models (Linear, Exponential)
    - Normalization (Ranking, Probability)
3. **Inference Layer**
    - Window Width Inference (Levene's Test)
    - Decay Slope Inference (Exponential Regression)
4. **Actuation Layer**
    - Hot-Words Cache (Frequency-Sorted)
    - Speculative Execution (Prefetch)
5. **Coordination Layer**
    - Heartbeat System (Time-Driven Ticks)
    - Loop Orchestration

4.2 Feedback Loop Classification

5 Convergence Theory

5.1 Deterministic Convergence

[Deterministic Convergence] A system exhibits deterministic convergence if, for all executions i, j of identical workloads:

$$\lim_{t\to\infty} \frac{|\text{metric}_i(t) - \text{metric}_j(t)|}{\sigma} < \epsilon \tag{17}$$

Loop	**Type**	**Mechanism**
#1 Execution Heat	Positive	$f \uparrow \Rightarrow$ rank $\uparrow \Rightarrow$ cache hit $\Rightarrow f \uparrow$
#2 Rolling Window	Neutral	Record history (monitoring only)
#3 Linear Decay	Negative	$f \uparrow \Rightarrow$ decay $\uparrow \Rightarrow f \downarrow$
#4 Pipelining	Positive	Transition $\uparrow \Rightarrow$ prediction $\uparrow$
#5 Window Inference	Negative	$\sigma^2 \uparrow \Rightarrow w \downarrow \Rightarrow \sigma^2 \downarrow$
#6 Slope Inference	Negative	Unstable $\Rightarrow \lambda \uparrow \Rightarrow$ stabilize
#7 Adaptive Heartbeat	Meta	Stable $\Rightarrow$ slower ticks

Table 1: Feedback loop classification by type and mechanism

where $\epsilon \to 0$ and σ is pooled standard deviation.

[0% Variance Convergence] Under continuous execution with fixed workload, the coefficient of variation of steady-state metrics approaches zero:

$$\lim_{t\to\infty} CV(t) = \lim_{t\to\infty} \frac{\sigma(t)}{\mu(t)} = 0 \tag{18}$$

Sketch.

1. Frequency evolution is governed by contractive mapping $T : f \mapsto f + r - \lambda f$.
2. Fixed point f^* exists by Banach fixed-point theorem when $\lambda > 0$.
3. Rolling window deterministically seeds metrics from identical initial conditions.
4. Levene's test-based adaptation reduces variance monotonically.
5. Therefore, all runs converge to same fixed point $\Rightarrow \sigma \to 0$.

□

5.2 Attractor Characterization

[Phase Space] The system state space is defined as:

$$\mathcal{S} = \{(w, \lambda, \sigma^2) \mid w \in \mathbb{N}, \lambda \in \mathbb{R}^+, \sigma^2 \in \mathbb{R}^+\} \tag{19}$$

[Attractor] A point $\mathbf{x}^* = (w^*, \lambda^*, \sigma^{*2})$ is an attractor if:

$$\mathbf{x}_{t+1} = F(\mathbf{x}_t) \quad \text{and} \quad F(\mathbf{x}^*) = \mathbf{x}^* \tag{20}$$

where F is the state transition function.

Empirically, StarForth exhibits a *stable fixed-point attractor* in (w, λ, σ^2) space, confirmed across 90 experimental runs.

6 Metaphorical Mappings

6.1 Thermodynamic Analogy

Thermodynamic Concept	**Execution Analog**
Thermal Energy	Execution Frequency (count)
Heat Dissipation	Exponential Decay ($e^{-\lambda t}$)
Temperature	Normalized Frequency Rank
Thermal Equilibrium	Steady-State Convergence
Cooling Rate	Decay Coefficient λ

Table 2: Thermodynamic metaphor mapping (conceptual only)

Important Caveat: These are *conceptual analogies*, not literal physical processes. No actual thermodynamic equations or hardware temperature measurements are involved.

7 Scope Limitations

7.1 What This Framework DOES Cover

- Execution frequency measurement and temporal decay
- Adaptive caching via frequency-based ranking
- Statistical inference for parameter tuning
- Dynamical systems characterization of convergence

7.2 What This Framework DOES NOT Cover

- Actual thermodynamic processes or physics simulations
- Machine learning or neural network training
- Quantum computing or quantum-inspired algorithms
- Biological neural systems or biomimicry

8 Usage Guidelines

8.1 Academic Writing Conventions

In Abstracts/Titles: Use precise, non-metaphorical terminology.

- ✓ “Thermodynamically-Inspired Adaptive Runtime”
- × “Physics-Based Virtual Machine”

In Technical Sections: Qualify metaphors explicitly.

> “We employ a *thermodynamic metaphor* where execution frequency is treated as ‘heat’ that dissipates exponentially over time, mathematically modeled as $f(t) = f_0 e^{-\lambda t}$.”

In Formalism: Use mathematical definitions, not analogies.

- ✓ “Frequency evolves as $df/dt = r(t) - \lambda f(t)$”
- × “Heat cools like Newton’s law of cooling”

9 Conclusion

This ontology provides a formal foundation for precise academic discourse on the StarForth adaptive runtime system. By explicitly distinguishing metaphorical concepts from literal implementations, we enable rigorous peer review and reproducible research.

A Notation Summary

References

[1] Strogatz, S. (2015). *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. Westview Press.

[2] Åström, K. J., & Murray, R. M. (2008). *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press.

[3] Casella, G., & Berger, R. L. (2002). *Statistical Inference*. Duxbury Press.

Symbol	**Definition**
f	Execution frequency
λ	Decay coefficient
w	Window size
K	Cache size
σ	Standard deviation
μ	Mean
CV	Coefficient of variation
$P(B\|A)$	Transition probability
w^*	Variance inflection point
$\mathcal{S}$	State space

Table 3: Mathematical notation used in this document

[4] Bolz, C. F., Cuni, A., Fijalkowski, M., & Rigo, A. (2009). Tracing the meta-level: PyPy's tracing JIT compiler. In *Proceedings of the 4th workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems* (pp. 18-25).

[5] Ertl, M. A. (1996). Stack caching for interpreters. In *ACM SIGPLAN Notices* (Vol. 30, No. 6, pp. 315-327).