Core Emotion Framework (CEF): TS 19 Appendix B — Reasoning Engine Pseudocode

Canonical Execution Logic for the CEF Semantic Inference Layer
Version 1.0 — Phase 4

Author: Jamel Bulgaria
ORCID: 0009-0007-5269-5739
Affiliation: OptimizeYourCapabilities.com
Contact: admin@optimizeyourcapabilities.com

Status: Canonical Appendix (TS-19)

---

0. Purpose and Canonical Position

Appendix B defines the **pseudocode implementation** of the CEF Reasoning Engine described in TS-19.

It provides:

- the canonical execution pipeline
- the internal data structures
- the inference loop
- the constraint-checking logic
- the output generation logic

This appendix introduces **no new emotional constructs**.
It operationalizes the inference rules defined in **TS-19 Appendix A**.

---

1. High-Level Engine Structure

```
function CEF_ReasoningEngine(inputData):

    validatedData = validateInput(inputData)
```

```
    normalizedState = normalize(validatedData)
    inferredState = infer(normalizedState)
    constrainedState = enforceConstraints(inferredState)
    output = generateOutput(constrainedState)
    return output
```

The engine always executes in this order:

1. **Validation** (TS-2 + Appendix D)
2. **Normalization** (TS-18)
3. **Inference** (TS-19 + Appendix A)
4. **Constraint Enforcement** (TS-1 → TS-17)
5. **Output Generation** (TS-19)

---

2. Input Validation Layer

```
function validateInput(data):
    if not conformsToJSONLD(data) and not conformsToRDF(data):
        raise ValidationError("Invalid ontology format")

    if violatesTS2Rules(data):
        raise ValidationError("TS-2 identity/structure violation")

    return data
```

Validation ensures:

- identity preservation
- no illegal transitions
- no illegal modulation
- no facet migration
- no center blending

---

3. Normalization Layer

```
function normalize(data):
    state = new State()

    state.operatorVector = extractOperators(data)
    state.facetVector = extractFacets(data)
    state.centerVector = extractCenters(data)
    state.modulationMatrix = buildModulationMatrix(data)
    state.transitionGraph = buildTransitionGraph(data)
    state.coherenceScalar = computeCoherence(data)

    return state
```

Normalization ensures:

- canonical ordering
- center fidelity
- no contamination
- consistent internal representation

---

4. Inference Layer

This is the core of the engine.

```
function infer(state):
    newState = copy(state)

    newState = inferIdentity(newState)
    newState = inferDirectionality(newState)
    newState = inferModulation(newState)
    newState = inferStability(newState)
    newState = inferPrediction(newState)
```

```
    newState = inferPlasticity(newState)
    newState = inferGovernance(newState)

    return newState
```

Each inference module corresponds to a rule family in Appendix A.

4.1 Identity Inference

```
function inferIdentity(state):
    for each facetPair (Fi, Fj):
        if Fi.precedes(Fj) and Fj.precedes(Fk):
            infer Fi.precedes(Fk)

    for each operator O:
        infer centerContains(O.center, O)

    return state
```

4.2 Directionality Inference

```
function inferDirectionality(state):
    for each transition (A -> B):
        for each transition (B -> C):
            infer A -> C

    return state
```

4.3 Modulation Inference

```
function inferModulation(state):
    for each modulation (A -> B):
```

```
    for each modulation (B -> C):
        infer A modulates C

  if elasticity(A,B) > threshold:
    infer B modulates A

  return state
```

4.4 Stability Inference

```
function inferStability(state):
  for each operator O:
    if O.activationLevel >= 0.8 * O.capacityLimit:
      mark O.stabilityRisk = HIGH

    if O.thresholdSpacing < minimumSpacing:
      mark O.instability = LIKELY

  return state
```

4.5 Predictive Inference

```
function inferPrediction(state):
  for each operator O:
    if O.driftVelocity > driftThreshold:
      infer predictsDrift(O)

  for each center C:
    if C.modulationDecayRate > collapseThreshold:
      infer predictsCollapse(C)
```

```
    return state
```

4.6 Plasticity Inference

```
function inferPlasticity(state):
    for each operator O:
        if cumulativeMicroAdjustments(O) > microShiftThreshold:
            infer longTermShift(O)

        if O.facetReorderingDelta > safeLimit:
            rejectFacetReordering(O)

    return state
```

4.7 Governance Inference

```
function inferGovernance(state):
    for each operator O:
        if O.stabilityRisk == HIGH and O.selfCorrectionGain >= correctionThreshold:
            infer selfCorrects(O)

    for each center C:
        if C.coherenceScalar < minimumCoherence and C.coherenceProtectionFactor >=
protectionThreshold:
            infer selfProtects(C)

    return state
```

---

5. Constraint Enforcement Layer

```
function enforceConstraints(state):
  if violatesIdentity(state):
    rollbackInference(state)

  if violatesDirectionality(state):
    rollbackInference(state)

  if violatesModulationRules(state):
    rollbackInference(state)

  if violatesFacetBoundaries(state):
    rollbackInference(state)

  if violatesGovernanceRules(state):
    rollbackInference(state)

  return state
```

This ensures:

- no illegal inferences
- no contamination
- no drift beyond canonical limits

---

6. Output Generation Layer

```
function generateOutput(state):
  output = {}

  output.inferredTransitions = state.transitionGraph
  output.inferredModulation = state.modulationMatrix
```

```
    output.stabilityAssessment = computeStabilitySummary(state)
    output.predictiveSignals = extractPredictions(state)
    output.governanceActions = extractGovernance(state)

    return output
```

Outputs must be:

- canonical
- identity-preserving
- center-bounded
- contamination-free

---

7. Canonical Status

Appendix B is the authoritative pseudocode specification for TS-19.
It defines the execution logic that implements the inference rules of Appendix A on the ontology defined in TS-18.

It is subordinate only to:

- Core Essence Document
- TS-1 → TS-19
