# VSAR Inference Engine: Unified Multi-Mode Control Semantics

This document specifies a **complete inference engine** for VSAR that exercises all supported reasoning modes using the encodings defined in **"VSAR: A Unified Encoding & Reasoning Framework (Clean Edition)"**.

VSAR's core idea:

> **One representation substrate (FHRR encodings) + multiple inference controllers (semantics).**

This engine is built as a **modular, pluggable control layer** over a shared memory of encoded objects.

---

## 0) Summary of reasoning modes and corresponding controllers

| Mode | Controller | Primary operations | Output type |
|---|---|---|---|
| Deductive / Horn | Forward/backward chaining | unify, apply rule, index | derived facts |
| DL (ALC core) | Tableau / completion | expand, clash check | satisfiable? / entailment |
| Abduction | Goal-driven inverse rules | unify head, hypothesize body | explanations |
| Induction | Program search / rule learning | generalize, score | learned rules |
| Analogical | Structure mapping | retrieve, align, map | mappings + transfers |
| Case-based | kNN retrieval + adaptation | similarity, mapping | adapted solution |
| Probabilistic | Weighted propagation | t-norms, noisy-OR | belief strengths |
| Paraconsistent | 4-valued / non-explosive | separate supports | ⟨supp(A), supp(¬A)⟩ |
| Non-monotonic/ default | Defeasible + defeat | priorities, exceptions | warranted set |
| Argumentation | AF/BAF solver | support/attack, propagation | acceptability scores |

| Mode | Controller | Primary operations | Output type |
|---|---|---|---|
| Epistemic | Multi-agent belief models | agent-indexed KBs | K/B entailments |

All controllers share:

- the same encodings and typed codebooks
- the same unification kernel
- the same retrieval/cleanup infrastructure

---

# 1) Data model and memory layout

## 1.1 Core stores

VSAR maintains these stores (conceptually; implementation may vary):

- **Symbol memories** (typed codebooks): `E, C, R, F, P, TAGS, OPS, ROLES...`
- **Fact store**: set of stored ground literals
- **Rule store**: stored rules (Horn, defeasible, DL axioms)
- **Graph store**: argument graphs (support/attack edges)
- **Case store**: cases for CBR
- **Map store**: analogy mappings

## 1.2 Item schema (unified)

Every stored item is:

```
Item = {
  vec: Complex[d],
  kind: enum {FACT, RULE, AXIOM, EDGE, CASE, MAP},
  weight: float or interval or mass,
  priority: float (optional),
  provenance: {source, timestamp, agent, trace},
  tags: set,
}
```

## 1.3 Paraconsistent belief record

For each literal `L`, maintain independent supports:

```
Belief[L] = (supp_pos, supp_neg)
```

- `supp_pos` applies to `L`

- `supp_neg` applies to `¬L`

This is the basis for 4-valued semantics.

---

# 2) Core primitives used by all controllers

## 2.1 Similarity, retrieval, and indexing

VSAR uses two *distinct* similarity mechanisms:

1) **Cleanup similarity (typed)**: nearest basis symbol in a codebook 2) **Item retrieval similarity (exploratory)**: nearest stored items to a query vector

### (A) Cleanup

Given $v$ and a typed codebook $S$:

$$cleanup_S(v) = \arg\max_{s \in S} \cos(v, s)$$

Cleanup is a **commitment** operation.

### (B) Retrieval

Given query $q$ and store $\mathcal{M}$:

$$retrieve_k(q) = \mathrm{TopK}_{x \in \mathcal{M}}\ \cos(q, x.vec)$$

Retrieval is **exploratory** and returns a set of candidates.

## 2.2 Unification kernel (slot-level)

Unification is performed by **structure-aware decoding** rather than whole-vector similarity.

### 2.2.1 Decode atom/term slots

For an atom encoding:

$$A = (P \otimes TAG_{ATOM}) \otimes \bigoplus_i (ARG_i \otimes enc(t_i))$$

Slot payload:

$$payload = A \oslash (P \otimes TAG_{ATOM})$$

Candidate slot value:

$$v_i = payload \oslash ARG_i$$

Then apply **typed cleanup**:

- constants: `cleanup_E(v_i)`
- terms: `cleanup_TERM(v_i)` (or recursively decode as term structure)

#### 2.2.2 Variable handling

Variables are not stored as basis symbols in the entity codebook. Instead, represent a variable token as:

- `VAR ⊗ TAG_META ⊗ (ID ⊗ var_id)`

Unification returns a substitution map:

```
θ = { x ↦ a, y ↦ f(b), ... }
```

Operationally, a variable position is decoded to a vector; the engine searches for bindings consistent with constraints.

### 2.3 Crisp query answering

For patterns like `p(a, ?)`:

1) retrieve candidate facts with matching predicate (index by predicate) 2) for each fact, decode the unknown slot via unbind 3) cleanup to a typed symbol 4) return all cleaned answers above threshold

This avoids “fuzzy whole-fact similarity” and produces crisp results.

---

## 3) Deductive reasoning (Horn / Datalog-style)

### 3.1 Rule representation

Rules are stored as:

$$R = TAG_{RULE} \otimes (HEAD \otimes enc(H) \oplus BODY \otimes enc(B_1 \wedge \cdots \wedge B_n))$$

### 3.2 Forward chaining algorithm (semi-naive)

**Inputs:** facts $F_0$, rules $\mathcal{R}$

**Loop:**

1. Maintain delta facts $\Delta$
2. For each rule, match body literals against $F \cup \Delta$ using unification

3. Produce new head instances and add to $\Delta'$
4. Stop when $\Delta'$ empty

**VSAR-specific optimization:**

- index facts by predicate vector `P_p`
- use slot decoding + cleanup for constant checks
- batch unbinding and cleanup in JAX

### 3.3 Backward chaining (goal-driven)

**Procedure prove(goal):**

- retrieve rules whose head predicate matches goal predicate
- unify goal with rule head
- recursively prove each instantiated body literal

Use memoization (tabling) to avoid loops.

---

## 4) Description Logic reasoning (ALC core)

VSAR supports DL by storing:

- TBox axioms: $C \sqsubseteq D$, $C \equiv D$
- ABox assertions: $C(a)$, $R(a, b)$

### 4.1 Completion / tableau state

Maintain a completion graph:

- node = individual
- label(node) = set of concept expressions
- edges = role assertions

### 4.2 Expansion rules (ALC)

- **⊓-rule:** if $C \sqcap D \in L(x)$ then add $C, D$ to $L(x)$
- **⊔-rule:** if $C \sqcup D \in L(x)$ then branch: add $C$ or add $D$
- **∃-rule:** if $\exists R.C \in L(x)$ then create y with edge R(x,y) and add C to L(y)
- **∀-rule:** if $\forall R.C \in L(x)$ and R(x,y) then add C to L(y)
- **clash:** if $C \in L(x)$ and $\neg C \in L(x)$ then clash

### 4.3 VSAR-specific machinery

- decode constructors by unbinding `LEFT/RIGHT/ROLE/FILLER`
- use typed cleanup to recognize operator tags
- retrieval is used to pull relevant axioms by matching LHS concept heads

### 4.4 Paraconsistent DL (optional)

If paraconsistent mode is enabled, a clash does **not** explode; it updates belief state and allows argumentation/defeat to resolve.

---

## 5) Paraconsistent inference (4-valued)

### 5.1 Belief update

When a fact `L` arrives with weight w:

- if L is positive: `supp(L) ← combine(supp(L), w)`
- if L is negative: `supp(¬L) ← combine(supp(¬L), w)`

Combine can be max, noisy-OR, or log-odds merge.

### 5.2 Query answering under paraconsistency

Return both supports:

```
query(L) -> (supp(L), supp(¬L))
```

Downstream controllers interpret this as true/false/both/neither.

---

## 6) Non-monotonic and default reasoning

Defaults are defeasible rules with priorities and exceptions.

### 6.1 Defeasible rule structure

`Rule = (vec, weight, priority, exceptions)`

### 6.2 Defeat relation

A derived conclusion is **defeated** if there exists an attacking rule/argument with:

- higher priority OR
- equal priority but higher weight OR
- more specific conditions

### 6.3 Warrant computation

Compute a fixpoint of warranted conclusions using an argumentation solver (Section 8).

---

# 7) Probabilistic / weighted inference

All derived items carry weights.

### 7.1 Body combination (AND)

$$w_{body} = T(w_1, \ldots, w_n)$$

where T is a pluggable t-norm (product, min, etc.).

### 7.2 Proof combination (OR)

$$w_{head} = S(w^{(1)}, \ldots, w^{(m)})$$

where S is a pluggable t-conorm (max, noisy-OR, etc.).

### 7.3 Integration with paraconsistency

Maintain weights independently for `L` and `¬L`.

---

# 8) Argumentation engine (AF/BAF)

### 8.1 Graph construction

- nodes: claims (literals, axioms, conclusions)
- edges: support and attack relations

Edge encodings:

$$enc(support(A, B)) = (EDGE \otimes SUPPORT) \otimes (SRC \otimes enc(A) \oplus TGT \otimes enc(B))$$

### 8.2 Semantics

Choose a semantics family:

- Dung AF (preferred extensions)
- bipolar AF (support + attack)
- weighted propagation (gradual semantics)

### 8.3 Gradual acceptability (example)

Let each node have base strength from evidence, then iteratively update:

$$str_{t+1}(x) = \sigma\Big(base(x) + \sum_{s \in Sup(x)} \alpha str_t(s) - \sum_{a \in Att(x)} \beta str_t(a)\Big)$$

where $\sigma$ is a squashing function.

Warranted conclusions are those with acceptability above threshold.

---

# 9) Epistemic reasoning (multi-agent)

## 9.1 Agent-indexed KBs

Maintain separate stores per agent:

- `KB[a].facts`
- `KB[a].rules`
- `KB[a].beliefs`

## 9.2 Modal operators

Knowledge/belief objects are encoded and can be nested:

- `enc(K_a φ)`
- `enc(B_a φ)`

## 9.3 Reasoning policies

- **K entailment** (factive, optional): if `K_a φ` then φ in world model
- **B entailment**: if `B_a φ` then φ in KB[a] with some weight

Updates:

- observation update to world
- communication update across agents (with trust weights)

Paraconsistency is supported per-agent.

---

# 10) Abduction engine

Given a goal literal G:

1) retrieve rules with head predicate matching G 2) unify G with rule head to obtain substitution θ 3) instantiate body under θ 4) identify which body literals are missing in current KB 5) propose missing as hypotheses

## 10.1 Explanation scoring

Score an explanation set H by:

$$Score(H) = \lambda_1 \sum_{h \in H} \log P(h) + \lambda_2 \sum_{r \in used} \log w(r) - \lambda_3 |H| - \lambda_4 Clash(H, KB)$$

Optionally include argumentation acceptability.

Return top-k explanations.

---

# 11) Analogical reasoning engine

## 11.1 Retrieval of candidate structures

- retrieve similar relational structures (facts/rules/cases) by exploratory similarity
- decode structures into slot-level components

## 11.2 Structure mapping

Compute a mapping M between symbols in domain A and domain B that maximizes preserved relations.

One objective:

$$M^* = \arg\max_M \sum_{(p(a,b) \in A)} \mathbf{1}[p(M(a), M(b)) \in B] \cdot w(p)$$

Use resonators / constraint satisfaction to find consistent M.

## 11.3 Transfer

Apply mapping to produce candidate hypotheses or solutions.

---

# 12) Case-based reasoning engine

Given a new problem P:

1) encode as `CASE(PROB ⊗ enc(P) ⊕ CONTEXT ⊗ enc(ctx))` 2) retrieve top-k similar cases by similarity of PROB slot 3) adapt solutions using analogical mapping engine 4) store new case with outcome and provenance

---

# 13) Induction (rule learning)

Induction is implemented as a search over rule templates.

### 13.1 Candidate generation

- anti-unification over sets of facts
- generalize constants into variables
- propose Horn clause templates

### 13.2 Scoring

Score candidate rules by:

- coverage of positive examples
- rejection of negatives
- simplicity
- weight/probability calibration

Store learned rules as defeasible rules with provenance.

---

## 14) Orchestration: a single engine that exercises all modes

### 14.1 Core loop

On each update (new facts/axioms/cases):

1) **Update belief state** (paraconsistent supports) 2) **Run deductive closure** (Horn) under current warranted set 3) **Run DL completion** for relevant individuals/concepts 4) **Recompute argument graph** and acceptability 5) **Apply defaults** (defeasible) subject to defeat

On each query:

- if it is a crisp logical query: use unbind→cleanup and deductive/proof controllers
- if it is an entailment query: consult DL + Horn + argumentation
- if it is an explanation query: invoke abduction
- if it is an analogy query: invoke mapping
- if it is a similar-case query: invoke CBR
- if it is agent-indexed: invoke epistemic controller

### 14.2 Guarantees and intended behavior

- **Soundness** is logic-dependent (DL/Horn when run classically)
- **Robustness** is provided by VSA similarity and noise tolerance
- **Non-explosiveness** is guaranteed by paraconsistent belief tracking
- **Extensibility** comes from adding new controllers without changing encodings

---

## 15) Engine-level test plan (high-level)

- deterministic bind/unbind invariants
- cleanup correctness under controlled bundling noise
- unification correctness on nested terms
- Horn closure matches symbolic baseline on toy KB
- ALC tableau results match a reference reasoner on toy ontologies
- paraconsistent non-explosion: derive does not trivialize under A and ¬A
- default defeat behaves by priority
- abductive explanations recover known hidden causes
- argumentation acceptability aligns with chosen semantics
- epistemic nesting decoding and agent separation
- analogy transfers preserve relational structure
- CBR retrieval/adaptation correctness

---

## 16) Implementation notes (VSAX-optimized)

- batch all unbind operations across candidate facts/rules
- use typed codebooks with matrix multiplication for cleanup
- maintain predicate indexes for O(1) candidate narrowing
- keep separate stores per agent for epistemic reasoning
- represent all proofs/derivations as provenance graphs for interpretability

---

## 17) Final view

VSAR's inference engine is a **portfolio of semantics** running over a single representational layer.

You can start with a minimal subset (Horn + cleanup) and progressively add controllers without changing encodings.