Vehicle Machine Language (VML)

Formal Specification Draft Version 0.1

Working Draft

July 2026

Abstract

Vehicle Machine Language (VML) is a machine-native behavioral language for autonomous vehicles and advanced driver assistance systems. The central premise is that natural language belongs to design time, not runtime. Human requirements are authored in natural language and translated into VML, while runtime systems execute and verify only VML.

This version 0.1 resets numbering and consolidates prior drafts into a single implementation-oriented specification. It preserves the strongest ideas from earlier iterations, removes duplicated concepts, and restructures the layered model with explicit syntax, static checks, operational semantics, and reference examples. The goal is practical: a reader should be able to implement a parser, type checker, monitor, and a planner interface from this document.

1 Why VML Exists

Natural language is powerful for people but problematic for machine execution in safety-critical systems. It is ambiguous, context dependent, and open-world by design. Runtime safety logic requires deterministic interpretation, traceability, and formal verification.

The VML pipeline is therefore:

Natural Language Requirements → Semantic Compiler (human + AI tooling) → VML → Runtime Verification

AI models may assist translation and review at design time, but no safety-critical runtime decision depends on natural-language interpretation.

2 Design Principles

VML is founded on the following core principles:

1. **No Runtime Natural Language:** Safety-critical decisions shall never depend on the interpretation of natural language.
2. **Explicit & Unambiguous Semantics:** Every VML statement must have one and only one interpretation.
3. **Formal Verifiability:** Every VML specification must be suitable for formal analysis and automated machine checking.
4. **Composability:** Complex behaviors shall be constructed from the combination of simpler, primitive behaviors.
5. **Human Readability:** While machine-oriented, VML must remain understandable to engineers to facilitate authoring, review, and debugging.
6. **Technology Independence:** VML semantics shall not depend on any specific perception system, planning algorithm, or neural network architecture.

3 Conformance Levels

VML 0.1 defines two profiles:

- **Core Profile (required for 0.1 conformance):** logic, temporal operators, metric comparisons, goals, obligations, and monitor semantics.
- **Extended Profile (optional in 0.1):** probabilistic expressions, ontology declarations, utility optimization objectives, contrary-to-duty deontics, and assumptions.

4 System Model

4.1 Execution Trace

A run is a discrete state trace

$$\sigma = S_0, S_1, S_2, \ldots$$

with sample period $\Delta t > 0$ (fixed or piecewise fixed by implementation configuration).

4.2 State Shape

Each state S_k contains:

- Boolean atoms (facts), for example `Collision = FALSE`.
- Numeric variables, for example `Ego.Velocity = 7.3`.
- Optional probability fields, for example `P(IsPedestrian(Obj1)) = 0.92`.
- Optional symbolic class memberships, for example `Obj1 ISA Pedestrian`.

5 Layered Language Model

VML is structured as a composition of logical layers, where each layer addresses a fundamental aspect of the driving task. This unified model integrates the core requirements of autonomous behavior into a single representation.

$$VML = L \oplus T \oplus M \oplus P \oplus G \oplus D \oplus O \oplus U$$

where layers are integrated but can be implemented incrementally.

Layer	Name	Question
L	Logical (Predicate Logic)	What facts hold? What exists and what is true?
T	Temporal Logic (LTL)	When must it happen? When must they hold?
M	Metric	Arithmetic Constraints: How much? (e.g., distance, velocity) What num
P	Probabilistic (optional)	How certain is perception/state? How certain are we?
G	Goal (Planning Objectives)	What do we desire to achieve? What future states are desired?
D	Deontic	What is obligatory, permitted, forbidden?
O	Ontology (optional)	What classes and inheritance apply?
U	Utility (optional)	How are soft objectives ranked?

6 Concrete Syntax (EBNF)

The following grammar is normative for parser behavior in 0.1.

```
program ::= declaration* statement*

declaration ::= class_decl | binding_decl
class_decl ::= "CLASS" IDENT "ISA" IDENT ("HAS" "{" attr_list "}")? ";"
attr_list ::= attr ("," attr)*
attr ::= IDENT ":" TYPE
binding_decl ::= IDENT "=" literal ";"

statement ::= goal_stmt
                  | hard_rule
                  | deontic_stmt
                  | require_stmt
                  | optimize_stmt
                  | assume_stmt

goal_stmt ::= "GOAL" expr ";"

hard_rule ::= "ALWAYS" expr ";"
                  | "EVENTUALLY" expr ";"
                  | "NEXT" expr ";"
                  | expr "UNTIL" expr ";"
                  | expr ";"

require_stmt ::= "REQUIRE" "(" expr ")" "IMPLIES" statement

optimize_stmt ::= "OPTIMIZE" "PRIORITY" "(" INT ")"
                     ("MINIMIZE"|"MAXIMIZE") expr ";"

assume_stmt ::= "ASSUME" expr ";"

deontic_stmt ::= deontic_op expr ("WHEN" expr)?
                     ("OTHERWISE" deontic_stmt)? ";"
deontic_op ::= "OBLIGATORY" | "PERMITTED" | "PROHIBITED"

expr ::= impl_expr
impl_expr ::= or_expr ("IMPLIES" or_expr)*
or_expr ::= and_expr ("OR" and_expr)*
and_expr ::= unary_expr ("AND" unary_expr)*
unary_expr ::= "NOT" unary_expr
                  | temporal_expr

temporal_expr ::= "ALWAYS" unary_expr
                  | "EVENTUALLY" unary_expr
                  | "NEXT" unary_expr
                  | primary_expr ("UNTIL" primary_expr)?

primary_expr ::= "(" expr ")"
                  | predicate
                  | comparison
                  | prob_expr
                  | isa_expr

predicate ::= IDENT ("(" arg_list ")")?
arg_list ::= expr ("," expr)*
comparison ::= value comparator value
```

```
comparator ::= "<" | "<=" | "=" | ">=" | ">"
value ::= IDENT | NUMBER | STRING | BOOLEAN
isa_expr ::= IDENT "ISA" IDENT

prob_expr ::= "P" "(" expr ")"
                  | "EXPECTED" "(" expr ")"
                  | "CDF" "(" expr ")"
                  | "EXPECTED_RISK" "(" arg_list ")"

literal ::= NUMBER | STRING | BOOLEAN
TYPE ::= "BOOLEAN" | "REAL" | "INT" | "STRING"
IDENT ::= /[A-Za-z_][A-Za-z0-9_.]*/
NUMBER ::= /-?[0-9]+(\.[0-9]+)?/
STRING ::= /"([^"\\]|\\.)*"/
BOOLEAN ::= "TRUE" | "FALSE"
INT ::= /[0-9]+/
```

7 Static Semantics (Type and Consistency Rules)

An implementation must reject programs that violate these checks.

7.1 Core checks

1. Every referenced identifier has a declaration source (state schema, binding, or class member).
2. Boolean operators consume Boolean subexpressions.
3. Numeric comparators consume numeric terms.
4. `UNTIL` has Boolean left and right operands.
5. `GOAL` expression is Boolean.
6. Deontic content (for `OBLIGATORY`/`PROHIBITED`/`PERMITTED`) is Boolean.

7.2 Optional checks

1. `P(...)` and `CDF(...)` evaluate to values in $[0, 1]$.
2. `PRIORITY(n)` uses positive integer levels; duplicate level declarations are allowed and form a set at that level.
3. `ISA` relations are acyclic.

8 Dynamic Semantics

8.1 Boolean and Temporal Semantics

Let $\sigma, k \models \varphi$ mean formula φ holds at state index k.

$$
\begin{aligned}
\sigma, k \models \texttt{NOT}\ \varphi &\iff \neg(\sigma, k \models \varphi) \\
\sigma, k \models \varphi\ \texttt{AND}\ \psi &\iff (\sigma, k \models \varphi) \wedge (\sigma, k \models \psi) \\
\sigma, k \models \varphi\ \texttt{OR}\ \psi &\iff (\sigma, k \models \varphi) \vee (\sigma, k \models \psi) \\
\sigma, k \models \varphi\ \texttt{IMPLIES}\ \psi &\iff \neg(\sigma, k \models \varphi) \vee (\sigma, k \models \psi) \\
\sigma, k \models \texttt{NEXT}\ \varphi &\iff \sigma, k+1 \models \varphi \\
\sigma, k \models \texttt{ALWAYS}\ \varphi &\iff \forall j \geq k :\ \sigma, j \models \varphi \\
\sigma, k \models \texttt{EVENTUALLY}\ \varphi &\iff \exists j \geq k :\ \sigma, j \models \varphi \\
\sigma, k \models \varphi\ \texttt{UNTIL}\ \psi &\iff \exists j \geq k :\ \sigma, j \models \psi\ \wedge\ \forall i \in [k, j) : \sigma, i \models \varphi
\end{aligned}
$$

8.2 Deontic Semantics in 0.1

For monitor semantics:

- `OBLIGATORY e`: violation when e is false in a state where it applies.
- `PROHIBITED e`: violation when e is true in a state where it applies.
- `PERMITTED e`: no direct violation; used as policy metadata and planner tie-break signal.

A `WHEN c` guard limits applicability to states where c is true.

8.3 Contrary-to-Duty (optional)

For

```
OBLIGATORY a OTHERWISE OBLIGATORY b;
```

if a is violated in an applicable state, then obligation b becomes active immediately in the same state and remains active until satisfied or superseded by implementation policy.

8.4 Probabilistic Functions (optional)

- `P(phi)` evaluates to confidence estimate that proposition ϕ is true in the current state.
- `EXPECTED(x)` returns $\mathbb{E}[x]$ from the estimator distribution.
- `CDF(x < c)` returns $\Pr(x < c)$.
- `EXPECTED_RISK(...)` returns expected weighted harm over modeled outcomes.

An implementation must document estimator source and update frequency for these values.

9 Layer-by-Layer Implementation Guidance

9.1 L: Logical Layer

Purpose: define the vocabulary of facts and relations.

Implementation:

- Maintain symbol table for atoms and predicates.
- Map sensor/perception/planner outputs to canonical atom updates.
- Use stable naming conventions (for example `Ego.Velocity`, `Obj12.IsPedestrian`).

Example:

```
PedestrianInCrosswalk;
InLane(Ego, Lane_2);
```

9.2 T: Temporal Layer

Purpose: encode persistence, liveness, and ordered behavior.

Implementation:

- Compile each temporal formula to a monitor automaton or progression state.
- For online monitoring, maintain per-formula runtime memory (for open obligations and pending eventualities).

Example:

```
ALWAYS (NOT Collision);
Braking UNTIL CrossingClear;
```

9.3 M: Metric Layer

Purpose: encode numeric safety envelopes and dynamic limits.

Implementation:

- Normalize units at ingestion (for example SI base units).
- Evaluate comparisons after unit conversion and signal validity checks.

Example:

```
ALWAYS (DistanceToPedestrian > 2.0);
ALWAYS (Ego.LateralAcceleration < 2.5);
```

9.4 P: Probabilistic Layer (optional)

Purpose: preserve uncertainty into decision logic instead of premature hard thresholding.

Implementation:

- Bind probability functions to estimator APIs.
- Set deterministic fallback when probabilistic source is missing (for example fail-safe low speed policy).

Example:

```
REQUIRE (P(IsPedestrian(Obj1)) > 0.95)
IMPLIES OBLIGATORY (Ego.Velocity = 0);
```

9.4.1 Probabilistic Representation: Bridging the Perception-Planning Gap ($\mathcal{P}$)

Perception systems (cameras, radar, LiDAR processed through neural networks) output probabilistic classifications and spatial coordinate state estimations containing sensor noise and epistemic uncertainty. Converting these continuous probability distributions into arbitrary binary decisions before path planning (e.g., "there is 51% probability of a pedestrian, so I must act as if there is 100% or 0% probability") leads to either dangerous planning gaps or overly conservative, jerky driving behavior (such as "phantom braking").

VML formally bridges this perception-planning gap by representing world entities as **stochastic states** and introducing operators that reason directly over probability density functions (PDFs), cumulative distribution functions (CDFs), and mathematical expectations.

9.4.2 Formal Probabilistic Attributes

Every dynamic entity e tracked by the system has an associated probabilistic state vector. VML 0.4 allows direct query access to these statistical properties:

- $P(e \text{ ISA } C)$: The categorical probability that entity e belongs to class C (e.g., $P(\text{Obstacle}_1 \text{ ISA Pedestrian}) = 0.88$).
- $e.\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$: The spatial state vector representing position, velocity, and heading, modeled as a probability distribution (e.g., Gaussian or Gaussian Mixture Model).

9.5 Advanced Statistical Operators

To express policies in safety-critical probabilistic terms, VML 0.4 introduces three key operators:

1. `EXPECTED[f(x)]`: Calculates the mathematical expectation $\mathbb{E}[f(x)] = \int f(x)p(x)dx$.
2. `CDF(Variable < Limit)`: Calculates the cumulative probability $P(\text{Variable} < \text{Limit})$.
3. `EXPECTED_RISK(Collision)`: Computes the deontic risk, defined as:

$$\mathcal{R}(\text{Collision}) = \sum_{c \in \text{Classes}} P(e \text{ ISA } c) \times \int_{\Omega_{\text{col}}} p(\mathbf{x}_{\text{ego}}) \cdot p(\mathbf{x}_e \mid c)\, d\mathbf{x} \times \text{Severity}(c)$$

   where $\text{Severity}(c)$ is a parameterized ethical severity metric (e.g., $\text{Severity}(\text{Pedestrian}) = 1.0$, whereas $\text{Severity}(\text{CardboardBox}) = 0.001$).

9.5.1 Stochastic Rules Syntax

This allows us to write elegant, deterministic threshold rules over stochastic states:

```
-- Stop rule based on collision probability and entity class severity
ALWAYS REQUIRE (EXPECTED_RISK(Collision, Ego, Obstacle_1) > 10^-5)
IMPLIES
  OBLIGATORY (Ego.Velocity = 0 UNTIL EXPECTED_RISK(Collision, Ego, Obstacle_1) <= 10^-6)
      ;

-- Speed adjustment based on confidence of lane boundaries
ALWAYS REQUIRE (CDF(DistanceTo(Ego, LeftLaneBoundary) < 0.2) > 0.05)
IMPLIES
  OBLIGATORY (Ego.Velocity < 10.0);
```

9.6 G: Goal Layer

Purpose: express desired outcomes that are pursued while respecting hard constraints.

Implementation:

- Keep goal status machine: inactive, active, reached, blocked.
- Do not allow goal pursuit to override hard safety violations.

Example:

```
GOAL At(Ego, Destination);
```

9.7 D: Deontic Layer

Purpose: encode obligations, permissions, prohibitions, and fallback duties.

Implementation:

- Evaluate applicability via `WHEN` guard.
- Emit structured violation records with timestamp, rule id, and witness signals.

Example:

```
OBLIGATORY YieldToPedestrian WHEN PedestrianInCrosswalk;
PROHIBITED CrossSolidLane;
```

In autonomous driving, critical situations occur where primary safety obligations cannot be fulfilled without violating another rule (for example, stepping over a double yellow line is prohibited, but it is necessary to bypass a broken-down vehicle blocking the lane). Traditional modal logic fails in these scenarios due to logical contradictions.

VML resolves this by utilizing **Contrary-to-Duty (CTD) logic**, which formalizes sub-ideal obligations. If a primary obligation $O(\alpha)$ is violated (or is physically impossible to meet), a secondary conditional obligation $O(\beta \mid \neg\alpha)$ is triggered. This provides a formal basis for the design of **Minimal Risk Maneuvers (MRM)**.

Mathematical syntax:

$$\textbf{OBLIGATORY}\ \alpha\ \textbf{OTHERWISE OBLIGATORY}\ \beta$$

```
-- Primary: Avoid collision while staying in lane.
-- Secondary (CTD): If a collision is physically unavoidable within current lane,
-- steer into empty shoulder (violating lane rules) to avoid high-severity collision.
ALWAYS OBLIGATORY (
  ALWAYS NOT (Ego.IsInCollision) AND ALWAYS (Ego.Lane = CurrentLane)
)
OTHERWISE OBLIGATORY (
  Ego.Maneuver = MinimalRiskManeuver AND Ego.TargetLocation = SafeShoulder
);
```

9.7.1 Spatial-Topological Logic

Relying strictly on continuous, absolute coordinate mathematics in rule specifications is highly brittle. VML introduces qualitative spatial reasoning using a formal topological logic inspired by Region Connection Calculus (RCC-8).

We define standard spatial topological predicates over two-dimensional spatial regions of entities:

- `Spatial.DC(A, B)`: Disconnected (A and B do not touch).
- `Spatial.EC(A, B)`: Externally Connected (A and B touch only at their boundaries).
- `Spatial.PO(A, B)`: Partial Overlap (A and B share interior points).
- `Spatial.TPP(A, B)`: Tangential Proper Part (A is completely inside B, touching its boundary).

```
-- Formalizing safe keeping using topological relations
ALWAYS PROHIBITED (Spatial.PO(Ego.BoundingBox, Vehicle_1.SafetyEnvelope));
```

9.7.2 Multi-Agent Game-Theoretic Assumptions

Safety guarantees depend on how we assume other agents will behave. Overly optimistic assumptions (such as "other drivers will always obey stop signs") are unsafe, while overly pessimistic assumptions (such as "other drivers may accelerate into me at any second") lead to vehicle freezing.

VML formalizes these interaction models via conditional `ASSUME` statements, defining game-theoretic boundaries of cooperation:

```
-- Assume cooperative play under bounded rationality
ASSUME ALWAYS (EXPECTED_RISK(Collision, OtherVehicle, ego) < 10^-2)
IMPLIES
  ALWAYS OBLIGATORY (MaintainSmoothTrajectory(Ego));
```

9.8 O: Ontology Layer (optional)

Purpose: reduce rule duplication by class inheritance.

Implementation:

- Build directed acyclic class graph.
- Expand class-targeted rules to instance-level obligations at runtime.

Example:

```
CLASS VulnerableRoadUser ISA DynamicObstacle;
CLASS Pedestrian ISA VulnerableRoadUser;
OBLIGATORY YieldTo(target) WHEN (target ISA VulnerableRoadUser);
```

To prevent VML files from becoming bloated with repetitive rules for every possible object type, VML integrates a formal taxonomic **Ontology Layer** ($\mathcal{O}$). This layer is formalized using Description Logic (DL) and supports semantic subclass inheritance and property attribution.

9.8.1 The Driving Taxonomy Hierarchy

We define a standardized hierarchical structure for the road ecosystem:

$$\text{Entity} \sqsubseteq \text{StaticObstacle} \sqcup \text{DynamicObstacle} \sqcup \text{Infrastructure}$$

$$\text{DynamicObstacle} \sqsubseteq \text{Vehicle} \sqcup \text{VulnerableRoadUser (VRU)}$$

$$\text{VRU} \sqsubseteq \text{Pedestrian} \sqcup \text{Cyclist} \sqcup \text{Animal}$$

$$\text{Vehicle} \sqsubseteq \text{StandardVehicle} \sqcup \text{EmergencyVehicle}$$

9.8.2 Ontological Axioms in VML

VML specifications can define taxonomic classes and relations directly, using the `ISA` and `HAS` syntax:

```
-- Ontological Definitions
CLASS VulnerableRoadUser ISA DynamicObstacle;
CLASS Pedestrian ISA VulnerableRoadUser;
CLASS Cyclist ISA VulnerableRoadUser;
CLASS EmergencyVehicle ISA Vehicle HAS { SirensActive: BOOLEAN, PriorityRightOfWay:
    BOOLEAN };

-- Generic Deontic Rule written for VulnerableRoadUsers
ALWAYS OBLIGATORY (YieldTo(Ego, target)) WHEN (target ISA VulnerableRoadUser AND
    Spatial.Overlap(target.SafetyZone, Ego.Path));
```

Behavioral Inheritance Benefit: Because `Pedestrian ISA VulnerableRoadUser` is declared in the ontology, any dynamic entity detected as a `Pedestrian` automatically inherits the yield obligation rule without requiring additional code. This significantly simplifies verification and decreases the semantic compile size.

9.9 U: Utility Layer (optional) - Optimization Objectives

Purpose: represent soft objectives under strict priority ordering.

Implementation:

- First enforce all hard constraints.
- Then optimize lexicographically by ascending priority number.

Autonomous driving is inherently a multi-objective optimization problem. An autonomous vehicle must simultaneously balance safety, regulatory compliance, passenger comfort, and operational progress. Standard temporal logics treat all constraints as binary (either fully satisfied or violated), which is highly impractical for real-world trajectory generation.

To resolve this, VML introduces the **Utility Layer** ($\mathcal{U}$), enabling the specification of soft constraints and mathematical optimization objectives alongside hard temporal requirements.

9.9.1 Lexicographical Priority Levels

Instead of relying on arbitrary scalar weights in a monolithic cost function—which often leads to fragile and unverifiable edge-case behaviors—VML defines optimization objectives using strict **Lexicographical Priority Levels**.

A higher-priority level constraint can never be compromised to satisfy a lower-priority objective. If multiple trajectories satisfy the higher-priority constraints equally, the lower-priority objectives are evaluated to break the tie.

9.9.2 Syntax and Mathematical Formulation

Let $J_i(x, u)$ be a continuous cost function mapping states x and control inputs u to a real-valued cost over a planning horizon H. The optimization objective is declared using the following syntax:

```
OPTIMIZE PRIORITY(1) MINIMIZE EXPECTED_RISK(Collision);
OPTIMIZE PRIORITY(2) MINIMIZE Ego.LateralJerk + Ego.LongitudinalJerk; -- Passenger
    Comfort
OPTIMIZE PRIORITY(3) MAXIMIZE Ego.Velocity; -- Progress
```

Mathematically, the trajectory generator solves:

$$\min_{u \in \mathcal{U}} \left(J_1(x,u), J_2(x,u), \ldots, J_n(x,u)\right) \quad \text{subject to } \Phi_{\text{hard}} \models \text{True}$$

where Φ_{hard} is the set of hard physical and safety temporal constraints (e.g., stopping at red lights).

Example:

```
OPTIMIZE PRIORITY(1) MINIMIZE EXPECTED_RISK(Collision);
OPTIMIZE PRIORITY(2) MINIMIZE Ego.LateralJerk + Ego.LongitudinalJerk;
OPTIMIZE PRIORITY(3) MAXIMIZE Ego.Velocity;
```

10 Reference Execution Pipeline

A minimal 0.1 implementation is:

$$\text{Lexer} \rightarrow \text{Parser} \rightarrow \text{AST} \rightarrow \text{Type Checker} \rightarrow \text{Rule Engine/Monitor}$$

A planner-integrated implementation is:

$$\text{Perception State} \rightarrow \text{VML Evaluator} \rightarrow \text{Constraint Filter} \rightarrow \text{Trajectory Optimizer} \rightarrow \text{Controller}$$

10.1 Online monitor loop

```
for each incoming state S_k:
  update atom table and numeric store
  evaluate guards and activate applicable deontic rules
  progress temporal monitors
  evaluate hard constraints and record violations
  update goals and optimization context
  publish verdict/event stream
```

11 Normative Runtime Outputs

A conforming runtime monitor must expose at least:

- `rule_id`
- `timestamp` (state index and wall time if available)
- `status` in {satisfied, violated, pending}
- `evidence` (minimal set of signal values used in decision)

12 Unified Complex Use-Case Scenario: Unprotected Left Turn

To demonstrate the expressive power of VML 0.4, we formalize a highly complex scenario: an **Unprotected Left Turn across oncoming traffic with high-uncertainty pedestrian presence on the crosswalk**.

```
-- Unprotected Left Turn VML Specification (Version 0.4)

-- 1. Ontological Mapping
CLASS OncomingVehicle ISA Vehicle;
CLASS CrosswalkPedestrian ISA Pedestrian;

-- 2. Utility Layer Optimization Goals
OPTIMIZE PRIORITY(1) MINIMIZE EXPECTED_RISK(Collision);
OPTIMIZE PRIORITY(2) MINIMIZE Ego.LateralJerk; -- Comfort
OPTIMIZE PRIORITY(3) MAXIMIZE Ego.Velocity; -- Flow

-- 3. Behavioral Rules
-- Rule A: Hard obligation to stop for pedestrian if risk is high
ALWAYS REQUIRE (EXPECTED_RISK(Collision, Ego, target) > 10^-5 AND target ISA Pedestrian
    )
IMPLIES
  OBLIGATORY (Ego.Velocity = 0.0)
  OTHERWISE OBLIGATORY (
```

```
    Ego.Maneuver = SafeBrakingToShoulder -- CTD Fallback if stop fails
  );

-- Rule B: Execute unprotected left turn only when gaps are statistically safe
ALWAYS OBLIGATORY (
  Ego.Maneuver = ExecuteLeftTurn
)
WHEN (
  EXPECTED_RISK(Collision, Ego, OncomingVehicle) < 10^-6 AND
  CDF(OncomingVehicle.Velocity > 25.0) < 0.02
)
OTHERWISE OBLIGATORY (
  Ego.Velocity = 0.0 AND Ego.Maneuver = YieldAtIntersection
);

-- Rule C: Respect spatial safety boundaries topologically
ALWAYS PROHIBITED (
  Spatial.PO(Ego.BoundingBox, Crosswalk.Boundary)
) WHEN (
  EXPECTED_RISK(Collision, Ego, target) > 10^-6 AND target ISA VulnerableRoadUser
);
```

13 Natural Language to VML Translation Workflow

Recommended process:

1. Parse requirement sentence into actor, condition, action, and timing.
2. Map domain terms to canonical ontology symbols.
3. Select hard constraints versus goals.
4. Introduce numeric thresholds and units explicitly.
5. Add fallback deontic obligations for infeasible primary duties.
6. Run type checks and simulation traces; revise until stable.

14 What Is Intentionally Out of Scope in 0.1

- Full denotational semantics for every probabilistic operator variant.
- Standardized ontology exchange format between organizations.
- Certification argument templates.
- Code generation for any specific vehicle platform.

15 Conclusion

VML 0.1 defines a practical formal foundation: natural language at design time, formal language at runtime. The specification is immediately implementable as a parser and monitor and extensible toward probabilistic planning, ontology-based reasoning, and lexicographic multi-objective optimization.

$$\text{Natural Language} \neq \text{Runtime Language}$$

Natural Language → VML → Safe Executable Behavior