ICS 03.220.20
CCS R04

group standard

T/ITS 0293.2-2025

Autonomous Transportation System Traffic Semantic Representation Language
Part 2: Grammar Rules

Released on 2025-11-26 Implementation on 2025-11-26

China Intelligent Transportation Industry Alliance released

table of contents

preface

This document is drafted in accordance with GB/T 1.1—2020 "Guidelines for Standardization Work Part 1: Structure and Drafting Rules of Standardization Documents".

Please note that some content in this document may involve patents. The publisher of this document is not responsible for identifying patents.

This document is proposed and overseen by the China Intelligent Transportation Industry Alliance (C-ITS).

The primary drafting institutions for this document include: Beijing Jiaotong University, Beijing University of Posts and Telecommunications, the Ministry of Transport's Highway Research Institute, Hualu Yiyun Technology Co., Ltd., China Unicom Smart Network Technology Co., Ltd., Beihang University, Zhuzhou CRRC Times Electric Co., Ltd., and the Ministry of Transport's Water Transport Research Institute.

The primary drafters of this document are: Dong Honghui, Niu Yajie, Wang Jiajia, Yuan Quan, Li Jinglin, Ren Yilong, Yu Haiyang, Li Zhenhua, She Hongyan, Xin Liang, Yu Chaoyang, Wang Quandong, Zhou Yucheng, Chen Yi, Ou Fan, Pan Xiaoxi, Li Wei, Wu Hao, Liu Junlan, Ni Jiaying, Feng Jiarui, Jiang Peiyuan, Hong Yipeng, Zhou Weijie, Lan Zhengxing, Gu Huinan, Ma Panke, and Lin Jun.

Autonomous Transportation System Traffic Semantic Representation Language Part 2: Grammar Specification

1 scope

This document defines the grammatical specifications for semantic representation of traffic knowledge in autonomous transportation systems, and supports semantic interactions among multiple traffic entities.

This document applies to traffic knowledge representation in autonomous transportation systems, including but not limited to traffic rules, traffic signs, traffic signals, and vehicle behavior.

2 Normative reference documents

The content of the following documents forms the essential provisions of this document through their normative references. For dated references, only the version corresponding to that date applies to this document; for undated references, their latest version (including all amendments) applies to this document.

GB/T 1.1—2020 Standardization Work Guidelines Part 1: Structure and drafting rules of standardization documents

ISO/IEC 13211-1-1995 Information technology-Programming language Prolog-Part 1: General core

ISO/IEC 14882—2020 Information technology: C++ programming language

T/ITS 0292-2025 Model of Interoperability Mechanism for Autonomous Transport Systems

T/ITS 0293.1-2025 Autonomous Traffic Systems Traffic Semantics Representation Language Part 1: General Definitions

3 Terms and Definitions

The following terms and definitions defined in T/ITS 0292-2025, T/ITS 0293.1-2025 and below apply to this document.

3.1

autonomous transportation system

It is a highly intelligent and highly autonomous traffic system characterized by autonomous perception, autonomous decision-making and autonomous execution.

[Source: T/ITS 0292-2025]

3.2

Traffic Semantic Representation Language

Traffic semantic representation language is a formal language to describe traffic content accurately, which has the abilities of semantic representation, semantic understanding, semantic interaction, logical reasoning and interoperability.

[Source: T/ITS 0293.1-2025]

3.3

Autonomous traffic agent

Autonomous traffic agent is a traffic intelligent agent which can realize the closed loop of perception, cognition, decision-making and control in the complex traffic environment and achieve the predetermined traffic task.

[Source: T/ITS 0293.1-2025]

3.4

fact fact

A basic statement describing the attributes of a transport entity, defined by the keyword "fact".

3.5

relation relation

A statement describing the relationship between traffic entities, defined by the keyword "relation".

3.6

rule rule

A conditional statement describing traffic logic constraints, defined by the keyword "rule".

4 abbreviation

RLRL: Traffic Semantic Representation Language

5 Lexical specifications for a traffic semantic representation language

5.1 Character Set and Encoding

This traffic semantic representation language uses the Unicode character set, supports Chinese and English traffic terms, and the source file is encoded in UTF-8.

5.2 ID

Identifiers in TSRL serve as symbolic carriers for naming core language elements. Their naming conventions directly impact the readability, consistency, and comprehension efficiency of semantic descriptions during multi-party interactions. To ensure uniform identification of the same language element across different developers and systems, the following clearly defines the composition rules and naming standards for identifiers:

In TSRL, identifiers are used to name modules, categories, classes, attributes, and predicates, com posed of letters, numbers, and underscores.

— Module names, class names, and entity names must start with a capital letter, e.g., TrafficLib, Vehicle;

— Attribute names and predicate names must start with a capital letter, such as HasSpeed and Distance.

— Constants must start with a capital letter and may include numbers and underscores, such as Vehicle_1 or Lane5.

—Variables and functions must start with a lowercase letter or an underscore. If only an undersc ore is present, it is treated as an anonymous variable, such as _ or speed(Car1).

5.3 keywords

Keywords are the basic symbols used in TSRL to construct core syntax structures. They define t he organizational forms of terms, entity attributes, relational connections, and operational logic through specific identifiers, serving as the core elements for building a standardized traffic semantic descripti on system. Specific keywords and their explanations are listed in Table 1.

Table 1 Keywords

keywords	explain
module	Define the terminology module
category	Define the category of the transportation system
class	Define the traffic entity class
fact	Define entity attribute facts
relation	Define entity relationships
action	Define physical action
rule	Define logical rules
let	Define the control operations for traffic entities.
select	Define knowledge base query operations
asserta	Insert before defining the knowledge base
assertz	Insert operation after defining the knowledge base
retract	Define knowledge base deletion operations
ask	Information Inquiry
tell	Transfer/Notify Information
print	Define print statements
fun	Define custom functions
false	indicates logical false value
true	indicates logical true value
nil	Indicates empty values or no entity
and	Logic and Operators
or	Logical or operator
not	Logical NOT operator
forall	universal quantifier ∀
exists	existential quantifier ∃

5.4 elementary data type

5.4.1 Numeric constants

Numerical constants are fundamental data types in TSRL (Traffic State and Resource Library) for quantifying traffic entity attributes, with their types and definitions as follows:

—Integer: An int type is a 32-bit signed integer, such as 1, 20, or 333.

—Float type: float is a representation of floating-point numbers, a 32-bit single-precision floating-point number.

5.4.2 Boolean value

A status qualifier type with a boolean value, where true means true and false means false.

5.4.3 null value

In TSRL, nil is used to represent null values.

5.4.4 alphabetic string

Text enclosed in double quotes (e.g., "crossroads", "green") supports escaping characters (line bre aks), \\" (double quotes).

5.5 Composite data type

5.5.1 outside

Tables are variable container-type entity types in TSRL language, used to organize traffic entity sets. They support dynamic management of multi-type traffic semantic elements and adapt to dynamically changing entity group requirements in traffic scenarios, such as vehicle queues or event sequences in a lane. The general form of table entity is:

$$l = [x_1, x_2, \dots x_n];$$

— Symbol explanation: [] is the table x_i delimiter; (i=1,2,...,n) are table entries representing traffic entities or semantic elements.

—Special form: An empty table without any elements is called an empty table, denoted as []. It can be used to initialize a set of traffic entities for dynamic filling.

5.5.2 dictionary

In TSRL, a dictionary is a variable container entity type that establishes traffic semantic key-value mappings. It uniquely associates traffic entities or attribute values, fulfilling the 'identifier-semantic' mapping requirements in traffic scenarios (e.g., signal phase-duration mapping, vehicle ID-status mapping). The general form of dictionary entity is:

$$d = \{key_1: value_1, key_2: value_2, \dots key_n: value_n\};$$

— Symbol explanation: {} is the dictionary key_n: $value_n$delimiter; is a key-value pair, with ":" a s the separator between keys and values, and commas separating key-value pairs.

—Key constraint: The key uniquely identifies traffic semantic terms, with values being associated entities or attributes.

5.6 Operators and Delimiters

Operators and separators are the core syntactic elements in TSRL for traffic data computation, logical judgment, and statement structure definition. Operators enable functional expressions such as numerical operations and logical reasoning, while separators delineate the boundaries of statement components (see Table 6). Together, they ensure the rigor and parseability of semantic descriptions. Operators include:

—Comparison operator: A symbol used to determine whether values are greater than, equal to, or less than each other (see Table 2).

— Arithmetic operators: symbols for performing numerical operations such as addition, subtraction, and multiplication (see Table 3);

—Quantifier operator: A marker indicating the logical concept of 'all' or 'existence' (see Table 4).

— Logical operators: Symbols or keyword combinations used to perform logical judgments and support traffic semantic logic reasoning (see Table 5).

Table 2 Comparison Operators

symbol	explain
>	Check if the left value is greater than the right value
<	Check if the left value is less than the right value
>=	Check if the left value is greater than or equal to the right value
<=	Check if the left value is less than or equal to the right value
==	Check if both sides are equal (supports numbers, strings, and booles).
!=	Check if the two sides are unequal

Table 3 Arithmetic Operators

symbol	explain
+	Addition of numbers/String concatenation
-	Subtract numbers / represent negative values
*	Multiplication of numbers
/	Floating-point division with decimal result
**	Power operation
//	Integer division, round the result to the nearest integer
%	complementation

Table 4: Quantifier Operators

equivalent logical identifier	explain	form	instance
forall	The logical symbol ∀ corresponds to the universal quantifier.	∀x P(x)	forall xP(x)
exists	Corresponding to the logical symbol ∃ , it represents the existential quantifier.	∃x P(x)	exists xP(x)

Table 5 Logical Operators

equivalent logical identifier	explain	priority	instance
not	Corresponds to the logical symbol ¬, indicating negation	1	not P(x)
and / &	Corresponds to the logical symbol ∧, representing a conjunctive.	2	P(x) and Q(x)
			P(x) & Q(x)

or / |	Corresponds to the logical symbol ∨, representing disjunction.	3	P(x) or Q(x)
			P(x) | Q(x)
=>	Corresponding logical symbol ⇒, indicating implication	4	Q(x) => P(x)
<=>	The logical symbol ↔ represents equivalence	5	P(x) <=> Q(x)

Table 6 Separator

symbol	explain
( )	Wrap the argument list of functions and predicates / change the expression priority
[ ]	Package table element list/table index access
{ }	Key-value pairs of code blocks or wrapper dictionaries for packages, categories, and classes
;	statement terminator
,	Separator parameter/separator element/separator key-value pair
:	Key and value in a partitioned dictionary

5.7 explanatory note

Comments in the RLRL language start with "#" and end at the end of the physical line. Exampl e: # Define vehicle class attributes.

6 Syntax specifications for the Traffic Semantic Representation Language

6.1 TSRL syntactic structure

6.1.1 fact

The syntax of fact is used to describe the specific information and state in the traffic system, and it is stipulated that the predicate and constant are used to represent these facts. There are two kinds of syntax of fact statement:

a) Multi-sentence structure

```
<fact>
        <predicate name> (<field list>);
        <predicate name> (<field list>);
        ......
</fact>
```

b) single statement structure

```
fact <predicate name> (<list>)
```

— Predicates are divided into entity, category, attribute and relation.

— Entity: Vehicle (ID). Example: Vehicle(x) denotes the vehicle entity with ID x.

— Category: Is predicate (parameter 1, parameter 2,..., parameter n). Example: IsVehicle(x), indicating x is a vehicle.

— Attribute: Has predicate (parameter 1, parameter 2,..., parameter n). Example: HasSpeed(Vehicle, Speed) indicates that the vehicle Vehicle has speed Speed.

—Relation: A predicate (parameter 1, parameter 2,..., parameter n). Example: On(Vehicle, Road) indicates a vehicle on a road.

6.1.2 rule

The syntactic structure of rules, used to describe constraints and behavioral norms in traffic systems, includes conditions and conclusions, represented by logical operators and quantifiers. There are two types of syntactic structures for rule statements:

a) Multi-sentence structure

```
<rule>
    <predicate name> (<list>): - <predicate name> (<list>) {, <predicate name> (<list>)};
    <predicate name> (<list>): - <predicate name> (<list>) {, <predicate name> (<list>)};
    ......
</rule>
```

b) single statement structure

rule <predicate name> (<list>): - <predicate name> (<list>) {, <predicate name> (<list>)};

The colon (":") denotes "if" (or simply "if"). The left predicate is the rule's conclusion (calle d the head), while the right predicate is the rule's premise (called the body).

— "{}" indicates zero or multiple repetitions, and commas denote "and" (logical AND).

6.1.3 ask about

The syntax of a query is used to retrieve information from the system. It specifies how to represent query conditions and results using predicates and variables. There are two types of syntax for a query statement:

a) Multi-sentence structure

```
<ask>
        <predicate name> (<list>){, <predicate name> (<list>)};
        <predicate name> (<list>){, <predicate name> (<list>)};
                              ......
</ask>
```

b) single statement structure

ask <predicate name> (<list>){, <predicate name> (<list>)};

6.1.4 inform

The syntax of a statement is used to convey information in a system, specifying the conditions and results of the statement through predicates and variables. There are two types of syntax for a statement:

a) Multi-sentence structure

```
<tell>
        <predicate name> (<list>){, <predicate name> (<list>)};
        <predicate name> (<list>){, <predicate name> (<list>)};
                              ......
</tell>
```

b) single statement structure

tell <predicate name> (<list>) {, <predicate name> (<list>)};

6.1.5 operate

The syntax structure of operation is used to describe the specific behavior of the entity in the traffic system, and it is divided into control, knowledge base operation, assignment operation and print operation.

a) Control: A specific action command for an object. The let keyword can trigger a behavior, as shown below:

let <predicate name> (<table>) {, <predicate name> (<table>)};

b) Knowledge base operations: Operations on the knowledge base, a dynamic data structure in memory composed of facts, with four predefined dynamic database operation predicates as follows:

select(<predicate name>)

asserta(<fact>);

assertz(<fact>);

retract(<fact>);

-- <fact> is a predicate expression that corresponds to a fact.

—select(<predicate name>) query statement. The program returns the predicate expression co ntaining the corresponding predicate name in the dynamic knowledge base.

—asserta(<fact>) inserts the fact into the current dynamic database before the predicate with the same name.

—assertz(<fact>) inserts the fact into the current dynamic database after the predicate of the same name.

—retract (<fact>) removes the fact from the current dynamic database.

c) Assign value: Assigns the value of an expression to another object and returns the assigned value. There are two forms:

<term> = <term>;

<term> is <term>;

When assigning with the "=" operator, a new entity is created for reference if the entity bel ongs to a basic entity type.

— Using the assignment symbol 'is' will make two entities point to the same address.

d) Print operation: The print statement prints the value of an expression.

print Expression;

If the expression is a calculable expression, print the result directly.

If the expression is a predicate expression, print the predicate expression directly without ver ifying its truth value.

6.2 class

a) The Concept and Characteristics of Class in RRL

— Category is the abstraction of a collective concept in the physical world.

—A class with certain attributes, where these attributes are instances of a specific type.

— Classes contain certain relationships, represented by predicate/function expressions. When creating a class, it comes with some built-in predicate/function expressions for entity typ e checking, attribute value assignment, and attribute validation. Users can also define th eir own predicate/function expressions for the class.

— A class contains certain rules, which are represented by rule statements.

— Class inheritance, where the inheritance relationship satisfies the predicate/function express ion Subset (subclass, superclass), and the derived class also contains predicate/function e xpressions for entity type validation, attribute value assignment, and attribute validation.

b) Construction of Class in RRL

The basic representation of a class is:

```
class classname(superclass) {
    ClasstypeA propertyA;
    PredicateNameA(entityp,1, entityp,2, ...);
    FunctionNameA(entityf,1, entityf,2, ...) = entityf,x;
    RuleNameA(entityr,1, entityr,2, ...) :- PredicateName1(entityrp,1, entityrp,2, ...), ...
};
```

c) instantiation of RL class

Instance statement:

classname(entityname);

— Entity instantiation: To create an entity Car1 of the Vehicle class, the TSRL statement is: Vehicle(Car1).

d) The Related Predicates and Function Word Expressions of RLRL Class

In the construction of classes, TSRL automatically generates predicates and function expressions in addition to those built by users.

predicate expression:

—Isclassname (entity), checks if the entity belongs to the class classname.

—PropertyA (entity, value), indicating that the attribute PropertyA in entity entity has value value.

Expression:

—PropertyA (classentity) = entityPropertyA, a property of the entity class entityPropertyA.

Regarding classes and entities, the TSRL's built-in predicate/function expressions are as follows:

—Member (entity, class name) checks whether the entity belongs to the class.

—PropertyOf(entity, classname) determines whether an entity is a property of the specified cl ass.

—PropertyOf(entity1, entity2) is used to verify whether entity entity1 is an attribute of entity entity2.

—TypeOf(entity1) = classname is used to retrieve the classname (classname) of the entity en tity1.

7 Traffic Semantic Representation Language Usage Specifications

7.1 General Principles of Use

To ensure the consistent application of TSRL across diverse traffic scenarios and user interactions, the following core principles must be adhered to:

a) Semantic consistency principle: The naming and description of traffic entities, attributes, and relationships must comply with Chapter 3 'Terms and Definitions' and Chapter 5 'Lexical Specifications' of this document. For example, the vehicle entity should be uniformly labeled as IsVehicle (entity), and the speed attribute should be uniformly described as HasSpeed (entity, value), to avoid homonymy or synonymy.

b) Logical rigor principle: Rule descriptions must be based on first-order logic representations, clearly define the logical relationship between conditions and conclusions, use logical operators to build inference chains, and prohibit logical contradictions or missing conditions.

c) Interaction standardization principle: Multi-party information exchange must be conducted through the ask and tell keywords, with interaction content including identity identifiers, information types, and timestamps to ensure traceability.

d) Data adaptability principle: Choose appropriate data types based on traffic scenario requirements. For example, use numeric constants for quantitative attributes and tables for entity sets.

7.2 Traffic Knowledge Expression Standards

7.2.1 Scene Description Specification

Scene description defines traffic application scenarios holistically, specifying the subject, boundarie s, and objectives to establish a framework for subsequent state descriptions and rule formulation. The specific specifications are as follows:

a) Define the scene subject by using IsXXX (entity) predicates to identify the subject type, e.g., IsVehicle (v) or IsTrafficLight (tl).

b) Define boundary conditions for the scene, including spatial relationships, environmental states, and device states. Example: HasTrafficLightState(id, state).

7.2.2 state description specification

State descriptions focus on real-time attributes and operational parameters of traffic entities, serving as the primary data source for rule evaluation and logical reasoning. They must be standardized according to entity types to ensure data accuracy and usability. For example, vehicle status should include the identifier HasVehicleID(v, id) and the movement attribute HasSpeed(v, s).

7.2.3 Rule Description Specification

Rules must be defined using the `rule` keyword, structured as "Conclusion: -Premise" (see 6.1.2). The premise section combines predicates with logical operators, while the conclusion section specifies the output result, such as warning trigger or action recommendation.

7.2.4 Information Interchange Specification

Information exchange is the key process for sharing TSRL (Traffic Safety Risk Level) description s among multiple traffic entities. It is essential to establish clear standards for initiating, transmitting, and managing data to ensure efficient and accurate information flow between entities.

a) Interaction initiation: The entity's identity must first be verified via IsEntityType (entity, type), followed by declaring the interaction request using the ask keyword.

b) Information transmission: The message must be encapsulated using the tell keyword, with the message body containing the information type identifier, semantic data, and timestamp.

c) Knowledge base operations: Use select, asserta, assertz, or retract keywords to manage dynamic data, ensuring accurate and up-to-date knowledge base information after interactions.

7.2.5 logical reasoning specification

a) Verify the truth of basic predicates before reasoning to exclude invalid entities or incorrect states.

b) The reasoning process calculates key parameters, and the parameter calculation must comply with operator rules.

c) After reasoning, the system matches the rule library. If the rule conditions are met, it derives the conclusion. The reasoning result must be printed with the print statement or passed to the target entity with the tell statement.

Appendix A
（ informative annex ）
Knowledge representation of road traffic scenarios

A.1 Scene Description

Focusing on traffic light-controlled intersections, this system involves three key elements: vehicles, traffic lights, and the environment. It eliminates interference from extreme weather or sudden obstacl es, with the core objective of assessing red-light running risks through logical reasoning and triggerin g alerts. The scenario is described as follows:

（1）Scene subject:

—IsVehicle (v): where 'v' denotes a vehicle.

—IsTrafficLight (tl): tl stands for traffic light.

—IsEnv (env): 'env' stands for environment.

（2）Scene boundary:

—IsInSameIntersection(v, tl): The vehicle v and traffic light tl are at the same intersection.

—HasRoadCondition(环境变量env, "good"): The road condition in environment env is good.

—HasWeatherCondition("env", "sunny"): The environment env has sunny weather.

—HasTrafficCongestion (env, "no"): The environment env is free of traffic congestion.

—HasTrafficLightState (env, "normal"): The traffic light (tl) is in normal operation.

A.2 Status Description

（1）vehicle status predicate

—HasVehicleID(v, id): The unique identifier for vehicle v is id.

—HasSpeed(v, s): The vehicle v travels at speed s (km/h).

—HasDistanceToStopLine(v, d): The distance between vehicle v and the stop line at the intersect ion, measured in meters.

（2）signal state predicate

—HasTrafficLightID (tl, id): The unique identifier for traffic light tl is id.

—HasTrafficLightColor(id, lane, color): Identifies the traffic light with id, where lane color is col or, with values of

red/green/yellow。

—HasPhaseRemainTime(tl, t): The remaining phase time of signal tl, in seconds

—HasTrafficLightState (tl, state): The traffic light tl's operational status is indicated by state, wit h values being normal or abnormal.

(3) Environmental state predicates

—HasRoadCondition (env, rs): The road condition of environment env is rs, with values good or bad.

—HasWeatherCondition(环境变量env,天气状态w),where w can be sunny, rainy, foggy, or similar.

—HasTrafficCongestion (env, c): The congestion level (c) in environment env, with values no/lo w/medium/high.

A.3 Rule Description

a) Rule 1 for triggering red light warning: Green or yellow light scenario

natural language description ：

For any vehicle v, traffic light tl, and environmental conditions env, if the following conditions a re satisfied:

1) The same intersection

2) The signal light is green or yellow

3) The sum of the remaining green and yellow light times is greater than 0

4) Time for the vehicle to reach the stop line at a constant speed> remaining green light time + re maining yellow light time

5) Good road condition

6) The traffic light is working properly.

7) No congestion

8) sunny weather

The vehicle v triggers a red-light violation alert when it arrives at a signal that has already turne d red, indicating a potential risk of running a red light.

TSRL description ：

```
#Red Light Alert Trigger Rule 1: The current signal is green or yellow
rule RedLightWarnTriggered1(v) :-
    IsVehicle(v) & IsTrafficLight(tl) & IsEnv(env) & InSameIntersection(v, tl)
    & (HasTrafficLightColor(tl, "green") | HasTrafficLightColor(tl, "yellow"))
    & HasGreenRemainTime(tl, t_g) & HasYellowRemainTime(tl, t_y) & ((t_g + t_y) > 0)
    & HasDistanceToStopLine(v, d) & HasSpeed(v, s) & ((d/(s*1000/3600)) > (t_g + t_y))
    & HasRoadCondition(env, "good") & HasTrafficLightState(tl, "normal")
    & HasTrafficCongestion(env, "no") & HasWeatherCondition(env, "sunny");
```

b) Rule 2 for triggering red light warning: The traffic light is red

natural language description ：

For any vehicle v, traffic light tl, and environmental conditions env, if the following conditions are satisfied:

1）The same intersection

2）The traffic light is red.

3）Red light remaining time> 0

4）Time for the vehicle to reach the stop line at a constant speed <remaining time of the red light

5）Good road condition

6）The traffic light is working properly.

7）No congestion

8）sunny weather

The vehicle v triggers a red-light violation alert when it arrives at a red light, indicating a potential risk of running a red light.

TSRL description ：

```
#Red Light Alert Trigger Rule 2: The current signal is red
rule RedLightWarnTriggered2(v) :-
    IsVehicle(v) & IsTrafficLight(tl) & IsEnv(env) & IsInSameIntersection(v, tl)
    & HasTrafficLightColor(tl, "red") & HasRedRemainTime(tl, t_r) & (t_r > 0)
    & HasDistanceToStopLine(v, d) & HasSpeed(v, s) & ((d/(s*1000/3600)) < t_r))
    & HasRoadCondition(env, "good") & HasTrafficLight(tl, "normal")
    & HasTrafficCongestion(env, "no") & HasWeatherCondition(env, "sunny");
```

c) Early Warning Association and Driving Recommendation Rules

natural language description ：

For vehicle v that triggers a red light warning, if the vehicle instance is valid, the system recom mends executing the 'Brake' action and outputs the corresponding deceleration parameter decel_val.

TSRL realize ：

```
#Driving Advisory Rules
rule RecommendAction(v, "Brake") :-
    (RedLightWarnTriggered1(v) | RedLightWarnTriggered2(v)) & IsVehicle(v);
rule RecommendDecel(v, decel_val) :-
    (RedLightWarnTriggered1(v) | RedLightWarnTriggered2(v)) & IsVehicle(v);
```

Appendix B
（ informative annex ）
Knowledge representation in rail transit scenarios

B.1 Scene Description

Taking the scenario of freight train dispatching and passenger train priority in railway section sta tions as an example, there are three trains, multiple track sections, and a control center responsible fo r dispatching decisions. To accurately describe their relationships and behaviors, the semantic expressio n of the traffic interaction scenario is as follows:

（1）Scene subject:

—IsTrack (track): indicates that 'track' refers to a railway track.

—IsPassengerTrain (train1): indicates that train1 is a passenger train.

—IsFreightTrain (train2): indicates that train2 is a freight train.

—IsControlCentre (cc): indicates that cc is the dispatch control center.

—IsDispatcher (dp): This indicates that dp is a dispatcher.

（2）Scene boundary:

—HasTrackStatus (track, status): Indicates the status of the track (e.g., Occupied or Free).

—HasPlannedRoute (train, track): indicates that the train's planned route is on the track.

—HasTrackAvailable (track): indicates whether the track is available.

—HasRightSwitchPosition (track): Indicates the correct position of the switch leading to track.

—HasTrackPassable (train, track): Indicates that the train can pass through the track.

—OnTrack (train, track): indicates a train is moving along the track.

—HasConflictCondition (track, "no"): indicates no train conflicts on track track.

—HasConflict (train1, train2, track): indicates a conflict between train1 and train2 on track.

—HasPriority (train, p): indicates that train train has priority level p.

B.2 Status Description

—HasTrainSpeed(x, speed): indicates that train x has speed;

—HasTrainAcceleration(x, acceleration): indicates that train x has an acceleration of acceleration.

—HasTrainCapacity(x, capacity): indicates that train x has a capacity of capacity.

—HasTrainStop(x, stop): indicates that train x has stop as its terminal station.

B.3 Rule Description

a) collision detection rule

natural language description :

A conflict occurs when FreightTrain1 occupies TrackA, while PassengerTrain's planned route is al so on TrackA and has the highest priority.

TSRL description :

```
Conflict Detection Rules
rule Conflict(PassengerTrain, FreightTrain1, TrackA) :-
    OnTrack(FreightTrain1, TrackA)
    & PlannedRoute(PassengerTrain, TrackA)
    & HasPriority(PassengerTrain, 1);
```

b) scheduling transfer rule

natural language description :

If a conflict occurs between the passenger train PassengerTrain and freight train FreightTrain1, and TrackC is available with no conflicts and the switch is correct, the dispatch center will generate a transfer instruction.

TSRL description :

```
#Shift Rules
rule Control(FreightTrain1, TransferTo(TrackC)) :-
    HasConflict(PassengerTrain, FreightTrain1, TrackA)
    & HasTrackAvailable(TrackC) ∧ HasConflictCondition(TrackC, ″no″ )
    & HasRightSwitchPosition(TrackC);
```

Appendix C

(informative annex)

Knowledge representation of water transportation scenarios

C.1 Scene Description

The ship collision warning system provides early warning to the driver about the impending colli sion through effective data processing and alarm system, thereby reducing the risk of frontal collisions. To accurately describe the relationship and behavior between the ships in traffic interaction scenarios, the following semantic description is provided:

（1）Scene subject:

—IsVessel (v): indicates that the individual v is a vessel;

—IIsMariner(m): indicates that individual m is a crew member;

—IsPort(x): indicates that x is a port;

—IsDock(x): indicates that x is a dock;

（2）Scene boundary:

—IsClosing(x, y): indicates that x and y are approaching each other.

—HasCollisionRisk(x, y): indicates a collision risk between x and y;

—HasAlertType(x,Alert): Indicates that x is an Alert type.

—HasWaterTrafficRule(x, rule) indicates that x complies with the water traffic rule specified as r ule. —HasWaterTrafficRule(x, rule) indicates that x complies with the water traffic rule specified as rule.

—HasSegmentType(w, type) indicates that the waterway segment w is of type type. —HasSeg mentType(w, type) indicates that the waterway segment w is of type type.

C.2 Status Description

—HasVesselPosition(v, position): indicates that vessel v's position is position;

—HasVesselSpeed (v, speed): indicates that the vessel's speed is speed;

—HasShipAcceleration(x, a): indicates that ship x has an acceleration of a;

—HasVesselCourse (v, course): indicates that vessel v is on course;

—HasVisibility(x, level): Indicates visibility level (e.g., good, moderate, poor);

—HasSeaCondition(x, condition): Indicates the sea condition is condition (e.g., calm, moderate waves, or rough seas).

C.3 Rule Description

rules of collision risk for ships

natural language description ：

This rule indicates that if vessel X and vessel Y are in a head-on situation, the distance between the two vessels is less than the safe distance, and the relative speed suggests they are approaching, there is a collision risk. Vessel X will receive a head-on collision warning.

TSRL description ：

```
#Rules for collision risks between ships
rule CollisionRisk(x, y) :-
    IsVessel(x) & IsVessel(y) & Distance(x, y, d) & IsHeadOnSituation(x, y)
    & IsClosing(x, y) & GreaterThan(safe_distance, d);
```

China Intelligent Transportation Industry Alliance
Standard
Self-organizing traffic system Traffic semantics representation language Part 2: Grammar Rules
T/ITS 0293.2-2025

8 Xitucheng Road, Haidian District, Beijing (100088)
China Intelligent Transportation Industry Alliance Printing
Website: http://www.c-its.org.cn

2025 11 First edition 2025 11 First printing