Linköping Electronic Articles in
Computer and Information Science
Vol. 5(2000): nr 14

The LMW Traffic World in the Fluent Calculus

Andreas Henschel and Michael Thielscher

Linköping University Electronic Press
Linköping, Sweden

http://www.ep.liu.se/ea/cis/2000/014/

Posted on December 18, 1999,
formally published on December 5, 2000
by Linköping University Electronic Press
581 83 Linköping, Sweden

Linköping Electronic Articles in
Computer and Information Science
ISSN 1401-9841
Series editor: Erik Sandewall

Typeset by the author using LaTeX
Formatted using étendu *style*

Recommended citation:
<Author>. <Title>. Linköping Electronic Articles in
Computer and Information Science, Vol. 5(2000): nr 14.
http://www.ep.liu.se/ea/cis/2000/014/. *December 5, 2000.*

This URL will also contain a link to the author's home page.

Abstract

An axiomatization of the Traffic World scenario as proposed in [1] is presented using the Fluent Calculus [4]. The most challenging property of the domain is its involving such diverse features and continuous change, concurrency, non-determinism, and ramifications, i.e., indirect effects. We show that the Fluent Calculus along with its augmentations is well-suited to meet this challenge.

Authors' address

Artificial Intelligence Institute
Dresden University of Technology
Dresden, Germany
E-mails: {andreas,mit}@pikas.inf.tu-dresden.de

1 Introduction

1.1 The Fluent Calculus

We use a Fluent Calculus which simultaneously deals with non-determinism, concurrency, ramification, continuous change and natural actions. For an introduction to the Fluent Calculus and its augmentations regarding

- Ramification
- Concurrent actions
- Continuous change

the reader is strongly encouraged to read [4], [5], and [6], respectively. However, the occurrence of the various aspects in a single model requires a sound combination of the various approaches; see the appendix for the complete collection of standard elements of the signature and foundational axioms.

1.2 Informal description of the model

In each state cars are moving at different positions. States are updated by both deliberative and natural actions, possibly concurrently, and their indirect effects (sections 3 and 5). For junctions something special was created: Each outgoing segment is furnished with a virtual waiting area to house the waiting loop for that segment (see also figure 1). This avoids the need for effect axioms which distinguish whether or not the road is free up to the required safety distance. This especially pays off in case of several cars concurrently approaching a certain segment: In accordance with a given priority ordering among segments at nodes (i.e., the right-of-way regulation), all cars will inevitably be moved into the waiting area in the right order.

Our way of modeling cars waiting for a road to become free, is inspired by these fancy slides in swimming pools: Once the person in front of you is far enough, a traffic light will be triggered and you are entitled to go. Admittedly, the approach seems to some extent unorthodox. However, the axiomatization is straightforward and who knows, if it proves to work out fine, maybe one day this will revolutionize the way of building junctions in the real world. Altogether the model should tackle situations of—informally speaking—“heavy traffic”.

Additionally, the following features are incorporated:

- Distinction between ‘non-deliberative’ and ‘deliberative’ cars, that is, cars which are not and which are, respectively, under our control.
- Traffic jam simulation via ramification.
- Waiting queue update via ramification.
- Cars are at any time either moving, waiting at the virtual entry of a segment, or, if deliberative, waiting at a node.
- Deliberative cars may speed up and slow down at any time.

2 Declarations

2.1 Domain sorts

Apart from the standard sorts (see appendix A), we introduce the sorts $\textsc{Segm}$ (road segments), $\textsc{Car}$ (cars, both deliberative and non-deliberative), $\textsc{Node}$ (nodes), $\mathbf{N}$ (the natural numbers), $\textsc{Dist}$ (distances, a non-negative real value in km), and $\textsc{Vel}$ (velocity of moving objects, a non-negative real value in $kmph$). We will use the variables $segm$, car, $node$, n, $distance$ (or d), and vel, resp., possibly with sub- or superscripts, for these sorts.

2.2 The Fluents

$$\begin{array}{rl} \mathit{Movement}: & \textsc{Car} \times \textsc{Dist} \times \textsc{Vel} \times \textsc{Time} \times \textsc{Segm} \times \textsc{Node} \mapsto \textsc{Fluent} \\ \mathit{AtNode}: & \textsc{Car} \times \textsc{Node} \times \textsc{Segm} \mapsto \textsc{Fluent} \\ \mathit{Blocked}: & \textsc{Node} \times \textsc{Segm} \mapsto \textsc{Fluent} \\ \mathit{Waiting}: & \textsc{Car} \times \textsc{Node} \times \textsc{Segm} \times \mathbf{N} \mapsto \textsc{Fluent} \\ \mathit{Counter}: & \textsc{Node} \times \textsc{Segm} \times \mathbf{N} \mapsto \textsc{Fluent} \end{array}$$

1. The 'process' fluent *Movement* is the central fluent, which models the continuous change of a car's position. An example is

   $$\mathit{Movement}(OD\_KA\_174, 2.5km, 70kmph, 9{:}30{:}00, segm_{14}, node_8)$$

   The parameters of the fluent are:
   - the name of the car (e.g., its number plate);
   - the distance the car had absolved on the current segment at the time of the last change of velocity;
   - the current speed;
   - the time of the last change of velocity;
   - the segment the car is driving on;
   - the node it is heading for.

   Hence, if the actual time is t, then the total distance on that particular segment can be calculated as follows:

   $$total = distance + vel(t - time)$$

2. The fluent $AtNode(car, node, segm)$ is true if the deliberative car has arrived at $node$, coming from segment $segm$.
3. The fluent $Blocked(node, segm)$ is true if segment $segm$ (outgoing from $node$) is blocked by some car which has not yet traveled far enough to allow the next car to leave the waiting loop.
4. The fluent $Waiting(car, node, segm, n)$ is true if car is in the waiting loop for $segm$ at $node$ in waiting position n.
5. The fluent $Counter(node, segm, n)$ is true if at $node$ there are n cars waiting to continue on segment $segm$.

2.3 Auxiliary predicates

We use two auxiliary predicates to describe properties of the domain which are not subject to change:

$$\begin{aligned} \mathit{Connects}: &\quad \textsc{Segm} \times \textsc{Node} \times \textsc{Node} \\ \mathit{Del}: &\quad \textsc{Car} \end{aligned}$$

where $\mathit{Connects}(\mathit{segm}, \mathit{node}_1, \mathit{node}_2)$ says that node_1 is connected to node_2 by segm while $\mathit{Del}(\mathit{car})$ says that the argument is a car under our control (we call it a 'deliberative' car).

2.4 Auxiliary Functions

$$\begin{aligned} \mathit{Length} &: \textsc{Segm} \mapsto \mathbf{R} \\ \mathit{Limit} &: \textsc{Segm} \mapsto \textsc{Vel} \\ \mathit{Topspeed} &: \textsc{Car} \mapsto \textsc{Vel} \\ \mathit{Min} &: \mathbf{R} \times \mathbf{R} \mapsto \mathbf{R} \\ \mathit{Priority} &: \textsc{Segm} \times \textsc{Node} \mapsto \mathbf{N} \end{aligned}$$

They should be self-explanatory, except maybe for the last one, which states the priority cars have at a node coming from a certain segment. (A car coming from a segment with higher priority has the right-of-way if turning into the same segment.)

2.5 Actions

2.5.1 Natural Actions

$$\begin{aligned} \mathit{ArriveAt} &: \textsc{Car} \times \textsc{Segm} \times \textsc{Node} \times \textsc{Segm} \times \textsc{Time} \mapsto \textsc{Action} \\ \mathit{ArriveAt}_D &: \textsc{Car} \times \textsc{Node} \times \textsc{Time} \mapsto \textsc{Action} \\ \mathit{Approach} &: \textsc{Car} \times \textsc{Car} \times \textsc{Time} \mapsto \textsc{Action} \\ \mathit{Unblock} &: \textsc{Node} \times \textsc{Segm} \times \textsc{Time} \mapsto \textsc{Action} \end{aligned}$$

1. The action $\mathit{ArriveAt}(\mathit{car}, \mathit{segm}_1, \mathit{node}, \mathit{segm}_2, t)$ denotes the arrival of the non-deliberative *car* coming from segm_1 at *node* and its immediate turn into segm_2 at time t.
2. The action $\mathit{ArriveAt}_D(\mathit{car}, \mathit{node}, t)$ denotes the arrival of the deliberative *car* at *node* at time t. (Since this car is under our control, it does not turn immediately. A subsequent deliberative *Turn* action may follow.)
3. The action $\mathit{Approach}(\mathit{car}_1, \mathit{car}_2, t)$ denotes car_1 reaching the safety distance of the car in front, car_2. (The effect will be to adjust its speed.)
4. The action $\mathit{Unblock}(\mathit{node}, \mathit{segm}, t)$ denotes segment *segm* becoming unblocked, outgoing from *node*. This happens whenever a car leaving *node* on *segm* has reached the safety distance ς.

2.5.2 Deliberative Actions

$$\begin{aligned} \mathit{Turn} &: \textsc{Car} \times \textsc{Segm} \times \textsc{Node} \times \textsc{Segm} \times \textsc{Time} \mapsto \textsc{Action} \\ \mathit{ChangeVel} &: \textsc{Car} \times \textsc{Vel} \times \textsc{Time} \mapsto \textsc{Action} \end{aligned}$$

Deliberative cars are privileged to control their turning and velocity: Action $\mathit{Turn}(car, segm_1, node, segm_2, t)$ means that coming from $segm_1$, car is at node $node$ and moves into the waiting area of $segm_2$ at time t, and action $\mathit{ChangeVel}(car, vel, t)$ denotes that car changes its velocity to vel at time t.

3 Specifications of Actions

3.1 The Axiom for Natural Actions

The four natural actions are summarized in a definition of the standard predicate $Natural$:

$$\begin{aligned} Natural(a) \leftrightarrow\ & (\exists)\, a = \mathit{ArriveAt}(car, segm_1, n, segm_2, t) \vee \\ & (\exists)\, a = \mathit{ArriveAt}_D(car, n, t) \vee \\ & (\exists)\, a = \mathit{Approach}(car_1, car_2, t) \vee \\ & (\exists)\, a = \mathit{Unblock}(node, segm, t) \end{aligned}$$

3.2 Non-deliberative cars arriving at a node

A car is expected to arrive at a node when it has absolved the full length of the segment. It non-deterministically chooses exactly one new segment to continue its journey. Non-deterministic choice of arguments of natural actions requires two implicational axioms for defining the preconditions.

$$\begin{aligned} & \mathit{Poss}(\mathit{ArriveAt}(car, segm_1, node, segm_2, t), z) \rightarrow \\ & \quad (\exists)\ \mathit{HoldsIn}(\mathit{Movement}(car, d, vel, t_0, segm_1, node), z) \wedge \\ & \qquad d + vel(t - t_0) = \mathit{Length}(segm) \wedge \neg \mathit{Del}(car) \wedge \mathit{After}(t, z) \wedge \\ & \qquad \mathit{Connects}(segm_2, node_1, node_2) \wedge segm_1 \neq segm_2 \end{aligned}$$

$$\begin{aligned} & \mathit{HoldsIn}(\mathit{Movement}(car, d, vel, t_0, segm, node), z) \wedge \\ & d + vel(t - t_0) = \mathit{Length}(segm) \wedge \neg \mathit{Del}(car) \wedge \mathit{After}(t, z) \rightarrow \\ & \quad \exists!\, segm_2.\ \mathit{Poss}(\mathit{ArriveAt}(car, segm, node, segm_2, t), z) \end{aligned}$$

The state update axiom for $\mathit{ArriveAt}$ terminates the current movement and places the car on the next free spot of the waiting area of the segment it has chosen above. The waiting loop uses the fluent $Counter$ counting the cars. If the parking space is empty and no other car is currently on $segm_2$ within ς-reach, the causal relationships (see Section 5.3) will guarantee that the car continues immediately as an indirect effect of its turn. To summarize, the negative direct effects are termination of movement, termination of current value of counter, and termination of starting time of the state; the positive direct effects are initiation of waiting in the waiting area, a new value of the counter, and a new starting time:[1]

$$
\begin{array}{l}
Poss(ArriveAt(car, segm_1, node, segm_2, t) \circ c, z) \wedge \\
\neg Cancels(c, ArriveAt(car, segm_1, node, segm_2, t), z) \rightarrow \\
\quad (\exists)\ DSucc(ArriveAt(car, segm_1, node, segm_2, t) \circ c, z) \\
\qquad \circ\, Movement(car, d, vel, t_0', segm_1, node) \\
\qquad\ \circ\, Counter(node, segm_2, n) \circ StartTime(t_0) \\
\qquad = DSucc(c, z) \circ Waiting(car, node, segm_2, n) \\
\qquad\quad \circ\, Counter(node, segm_2, n+1) \circ StartTime(t)
\end{array}
$$

Cancellation takes place whenever there is a car (either deliberative or not) which attempts to turn onto the same segment but has a higher priority. The key here is that only those decompositions of concurrent action c occur without cancellation where all turns into the same segment are inferred in the order of preference.

$$
\begin{array}{l}
Cancels(c, ArriveAt(car_1, segm_1, node, segm_2, t), z) \leftrightarrow \\
\quad (\exists)\ Priority(segm_1', node) > Priority(segm_1, node) \wedge \\
\qquad [\, HoldsIn(Turn(car_2, segm_1', node, segm_2, t), c) \vee \\
\qquad\ HoldsIn(ArriveAt(car_2, segm_1', node, segm_2, t), c)\,]
\end{array}
$$

3.3 Deliberative cars arriving at a node

As the arrival at a node of a car is inevitably a natural action, the action is split up into two parts: Arrival at a node (natural action) and choosing a new segment (deliberative action; see section 3.4).

$$
\begin{array}{l}
Poss(ArriveAt_D(car, node, t), z) \leftrightarrow \\
\quad HoldsIn(Movement(car, d, vel, t_0, segm, node), z) \wedge \\
\quad d + vel(t - t_0) = Length(segm) \wedge Del(car) \wedge After(t, z)
\end{array}
$$

$$
\begin{array}{l}
Poss(ArriveAt_D(car, node, t) \circ c, z) \rightarrow \wedge \\
\quad (\exists)\ DSucc(ArriveAt_D(car, node, t) \circ c, z) \circ StartTime(t_0) \\
\qquad \circ\, Movement(car, d, vel, t_0', segm, node) \\
\qquad = DSucc(c, z) \circ StartTime(t) \circ AtNode(car, node, segm_1)
\end{array}
$$

3.4 Deliberative cars turning

$$
\begin{array}{l}
Poss(Turn(car, segm_1, node, segm_2, t), z) \leftrightarrow \\
\quad HoldsIn(AtNode(car, node, segm_1), z) \wedge \\
\quad \exists node'.\ Connects(segm_2, node, node')) \wedge After(t, z)
\end{array}
$$

[1]The state update axioms used here and in the following are designed for continuous change, concurrency, and ramification; see the appendix.

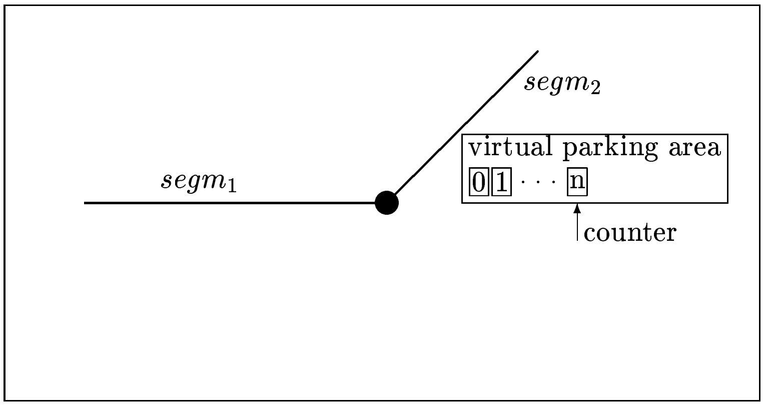


Figure 1: Design of junctions

$$
\begin{aligned}
& Poss(Turn(car, segm_1, node, segm_2, t) \circ c, z) \land \\
& \neg Cancels(c, Turn(car, segm_1, node, segm_2, t), z) \rightarrow \\
& \quad (\exists)\ DSucc(Turn(car, segm_1, node, segm_2, t) \circ c, z) \\
& \qquad \circ AtNode(car, node, segm_1) \\
& \qquad\ \circ Counter(node_1, segm_2, n) \circ StartTime(t_0) \\
& \qquad = DSucc(c, z) \circ Waiting(car, node, segm_2, n) \\
& \qquad\quad \circ Counter(node, segm_2, n+1) \circ StartTime(t)
\end{aligned}
$$

$$
\begin{aligned}
& Cancels(c, Turn(car_1, segm_1, node, segm_2, t), z) \leftrightarrow \\
& \quad (\exists)\ Priority(segm_1', node) > Priority(segm_1, node) \land \\
& \qquad [\, HoldsIn(Turn(car_2, segm_1', node, segm_2, t), c) \lor \\
& \qquad\ HoldsIn(ArriveAt(car_2, segm_1', node, segm_2, t), c)\,]
\end{aligned}
$$

3.5 Approaching another car

$$
\begin{aligned}
& Poss(Approach(car_1, car_2, t), z) \leftrightarrow \\
& \quad (\exists)\ HoldsIn(Movement(car_1, d_1, vel_1, t_1, segm, node), z) \land \\
& \qquad HoldsIn(Movement(car_2, d_2, vel_2, t_2, segm, node), z) \land \\
& \qquad d_1 + vel(t - t_1) + \varsigma = d_2 + vel(t - t_2) \land After(t, z)
\end{aligned}
$$

Reaching the safety distance to another car leads to the effects that the old movement is terminated and a new one is initiated with the speed of the hindering car and the new starting point (in order to keep track of the position).

$$
\begin{aligned}
& Poss(Approach(car_1, car_2, t) \circ c, z) \land \\
& HoldsIn(Movement(car_2, d_2, vel_2, t_2, segm, node), z) \rightarrow \\
& \quad (\exists)\ DSucc(Approach(car_1, car_2, t) \circ c, z) \\
& \qquad \circ Movement(car_1, d_1, vel_1, t_1, segm, node) \\
& \qquad = DSucc(c, z) \circ StartTime(t) \\
& \qquad\quad \circ Movement(car_1, d_1 + vel_1 \cdot (t - t_1), vel_2, t, segm, node)
\end{aligned}
$$

The reader may note that this may influence other cars behind. This is taken care of by ramification (see section 5.1).

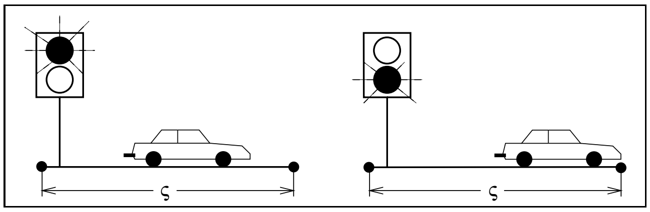


Figure 2: Triggering the traffic light

3.6 Triggering the Traffic Light—the Unblock Action

The situation is depicted in figure 2.

$$
\begin{aligned}
&Poss(Unblock(node, segm, t), z) \leftrightarrow \\
&\quad (\exists)\ HoldsIn(Movement(car, d, vel, t_0, segm, node'), z) \wedge \\
&\qquad d + vel \cdot (t - t_0) = \varsigma \wedge HoldsIn(Blocked(node, segm), z) \wedge \\
&\qquad Connects(segm, node, node')
\end{aligned}
$$

$$
\begin{aligned}
&Poss(Unblock(node, segm, t) \circ c, z) \rightarrow \\
&\quad (\exists)\ DSucc(Unblock(node, segm, t) \circ c, z) \\
&\qquad\ \circ Blocked(node, segm) \circ\ StartTime(t_0) \\
&\qquad = DSucc(c, z) \circ StartTime(t)
\end{aligned}
$$

3.7 Changing the velocity

Whenever a deliberative car is in motion, it is entitled to change its velocity.

$$
\begin{aligned}
&Poss(ChangeVel(car, vel, t), z) \leftrightarrow \\
&\quad (\exists)\ HoldsIn(Movement(car, d, vel', t_0, segm, node), z) \wedge \\
&\qquad Del(car) \wedge After(t, z)
\end{aligned}
$$

As effect, the old movement terminates and a new one is initialized with new initial position, velocity, and time.

$$
\begin{aligned}
&Poss(ChangeVel(car, vel, t) \circ c, z) \rightarrow \\
&\quad (\exists)\ DSucc(ChangeVel(car, vel, t) \circ c, z) \\
&\qquad\ \circ Movement(car, d, vel_{old}, t', segm, node) \\
&\qquad\ \circ StartTime(t_0) \\
&\qquad = DSucc(c, z) \circ StartTime(t) \\
&\qquad\quad \circ Movement(car, d + vel_{old}(t - t'), vel, t, segm, node)
\end{aligned}
$$

3.8 The General Precondition Axiom for Concurrency

A concurrent action is possible iff so is each single action and if deliberative cars do not perform two conflicting actions.

$$
\begin{aligned}
Poss(c,z) \leftrightarrow\ & (\forall a)\,(HoldsIn(a,c) \rightarrow Poss(a,z)) \wedge \\
& (\forall)\,(HoldsIn(Turn(car, segm_1, node, segm_2, t) \\
& \qquad\qquad \circ\ Turn(car', segm'_1, node', segm'_2, t), c) \\
& \quad \rightarrow car \neq car') \wedge \\
& (\forall)\,(HoldsIn(ChangeVel(car, vel, t) \\
& \qquad\qquad \circ\ ChangeVel(car', vel', t), c) \\
& \quad \rightarrow car \neq car')
\end{aligned}
$$

4 State Constraints

The states in all situations must obey the following constraints. Note that satisfying these constraints often enforces indirect effects of actions via ramification (see section 5).

4.1 Only one Status per Car

Cars can only be in one of three possible states at any time.

$$
\begin{aligned}
& Holds(Movement(car, d, vel, t, segm, node), s) \rightarrow \\
& \quad \neg(\exists) Holds(Waiting(car, node, segm, n), s) \wedge \\
& \quad \neg(\exists) Holds(AtNode(car, node, segm), s) \wedge \\
& \quad \forall[Holds(Movement(car, d', vel', t', segm', node'), s) \rightarrow \\
& \qquad d = d' \wedge vel = vel' \wedge t = t' \wedge segm = segm' \wedge node = node'] \\
& \wedge \\
& Holds(Waiting(car, node, segm, n), s) \rightarrow \\
& \quad \neg(\exists) Holds(Movement(car, d, vel, t, segm, node), s) \wedge \\
& \quad \neg(\exists) Holds(AtNode(car, node, segm), s) \wedge \\
& \quad \forall[Holds(Waiting(car, node', segm', n'), s) \rightarrow \\
& \qquad segm = segm' \wedge node = node' \wedge n = n'] \\
& \wedge \\
& Holds(AtNode(car, node, segm), s) \rightarrow \\
& \quad \neg(\exists) Holds(Waiting(car, node, segm, n), s) \wedge \\
& \quad \neg(\exists) Holds(Movement(car, d, vel, t, segm, node), s) \wedge \\
& \quad \forall[Holds(AtNode(car, node', segm'), s) \rightarrow \\
& \qquad segm = segm' \wedge node = node']
\end{aligned}
$$

4.2 Safety Distance

Cars must keep the global safety distance.

$$
\begin{aligned}
&Holds(Movement(car_1, d_1, vel_1, t_1, segm, node), s) \wedge \\
&Holds(Movement(car_2, d_2, vel_2, t_2, segm, node), s) \wedge \\
&Start(s) = t \rightarrow car_1 \neq car_2 \\
&\quad d_1 + vel_1(t - t_1) - (d_2 + vel_2(t - t_2)) \geq \varsigma \wedge \\
&\quad [d_1 + vel_1(t - t_1) - (d_2 + vel_2(t - t_2)) = \varsigma \rightarrow \\
&\qquad vel_2 \leq vel_1]
\end{aligned}
$$

4.3 Pacing up

We assume that non-deliberative cars go with the maximal possible speed, which is either the car's top speed, the segment's speed limit, or the velocity of a slower car in front.

$$
\begin{aligned}
&Holds(Movement(car_1, d_1, vel_1, t_1, segm, node), s) \wedge \neg Del(car) \wedge \\
&Start(s) = t \wedge vel < Min(Topspeed(car), Limit(segm)) \rightarrow \\
&\quad (\exists)\ Holds(Movement(car_2, d_2, vel_2, t_2, segm, node), s) \wedge \\
&\qquad d_1 + vel(t - t_1) - d_2 + vel(t - t_2) = \varsigma
\end{aligned}
$$

(This state constraint could be optional.)

$$
\begin{aligned}
&Poss(ChangeVel(car, vel, t) \circ c, z) \ \rightarrow \\
&\quad (\exists)\ DSucc(ChangeVel(car, vel, t) \circ c, z) \\
&\qquad \circ\, Movement(car, d, vel_{old}, t', segm, node) \\
&\qquad \circ\, StartTime(t_0) \\
&\qquad = \ DSucc(c, z) \circ StartTime(t) \\
&\qquad\quad \circ\, Movement(car, d + vel_{old}(t - t'), vel, t, segm, node)
\end{aligned}
$$

4.4 Waiting for Access to a Segment

If a car is waiting for access to a segment, then only because the traffic light is red (we are about to rule out those daydreamers...)

$$
\begin{aligned}
&Holds(Waiting(car, node, segm, 0), s) \rightarrow \\
&\ Holds(Blocked(node, segm), s)
\end{aligned}
$$

4.5 Occupied Segments are Blocked

A segment is blocked iff a car is within ς-reach on that segment.

$$
\begin{aligned}
&Holds(Blocked(node, segm), s) \leftrightarrow \\
&\ Holds(Movement(car, d, vel, t_0, segm, node), s) \wedge \\
&\ Start(s) = t \wedge d + vel(t - t_0) < \varsigma
\end{aligned}
$$

4.6 Controlled Cars

Only controlled cars are entitled to be at a node (taking time to decide which segment to turn onto).

$$Holds(AtNode(car, node, segm), s) \rightarrow Del(car)$$

4.7 No Holes in Waiting Loop

$$\begin{aligned}&Holds(Waiting(car, node, segm, n+1), s) \rightarrow \\ &\quad Holds(Waiting(car, node, segm, n), s)\end{aligned}$$

4.8 Driving sensible

A car cannot exceed the speed limit either given by the specification of the car's topspeed or the speed limit of the current segment.

$$\begin{aligned}&Holds(Movement(car, d, vel, t_0, segm, node), s) \rightarrow \\ &\quad vel \leq Min(Limit(segm), Topspeed(car))\end{aligned}$$

4.9 Unique Counter

$$Holds(Counter(node, segm, n_1), s) \wedge Holds(Counter(node, segm, n_2), s) \\ \rightarrow n_1 = n_2$$

5 Causal Relationships

The Traffic World allows for a simplified variant of the Fluent Calculus approach to the Ramification Problem: Since the occurrence of an indirect effect always depends solely on the current intermediate state, the effects by which this state has been reached need not be recorded (see also appendix B.3).

5.1 Traffic Jam

In order to maintain the safety distance between cars, the natural action *Approach* describes how to slow down in the event of catching up with a slower car. For a whole convoy to slow down, this is not sufficient, though. The first car behind the slower car manages to adapt via the state update axiom in section 3.5, but the second car (in ς-distance) does not and thus violates the traffic regulations. This is ruled out by the causal relationship given below. Ramification at its best: In a manner of falling dominoes the effect of deceleration is propagated through an entire convoy! Conversely, speeding up the whole convoy, e.g., in case a deliberative car accelerates, is captured by the same causal relationship as well. So here we go:

$$
\begin{array}{l}
Causes(z_1, z_2) \leftrightarrow \\
\quad HoldsIn(Movement(car_1, dist_1, vel_{new}, t_1, segm, node), z_1) \wedge \\
\quad HoldsIn(Movement(car_2, dist_2, vel_{old}, t_2, segm, node), z_1) \wedge \\
\quad \varsigma = dist_1 - vel_{new} \cdot t_1 - (dist_2 - vel_{old} \cdot t_2) \wedge \\
\quad [\, vel_{new} < vel_{old} \vee (vel_{new} > vel_{old} \wedge \neg Del(car_2))\,] \wedge \\
\quad z_2 \circ Movement(car_2, dist_2, vel_{old}, t_2, segm, node) = \\
\qquad z_1 \circ Movement(car_2, dist_2 + vel_{old} \cdot (t_1 - t_2), \\
\qquad\qquad\qquad Min(Topspeed(car), vel_{new}), t_2, segm, node) \\
\quad \vee \ldots
\end{array}
$$

(The ellipses indicates that more causal relationships are defined below.)

The 'obsolete' state: In z_1, car_1 is moving and car_2 is behind in ς-distance, possibly restricted in speed by the car in front. The first car, car_1, has a different velocity, which may be due to its just having changed speed.

The updated state: In z_2, car_2 has changed its velocity. It had to slow down if the car in front slowed down, and it had to speed up if it is a non-deliberative car and the car in front enhanced speed.

Notice that if a car is deliberative and the car in front accelerates, then the former does not adapt automatically. Rather it may speed up using the deliberative action *ChangeVel*.

5.2 Non-deliberative Car Speeds Up

A non-deliberative car speeds up automatically if a slower car in front has arrived at a junction, thus 'disappearing' from the road.

$$
\begin{array}{l}
Causes(z_1, z_2) \leftrightarrow \\
\quad \ldots \vee \\
\quad HoldsIn(Movement(car_1, dist_1, vel_1, t_1, segm, node), z_1) \wedge \\
\quad \neg(\exists)\,[\, HoldsIn(Movement(car_2, dist_2, vel_2, t_2, segm, node), z_1) \wedge \\
\qquad\quad \varsigma = dist_1 - vel_1 \cdot t_1 - (dist_2 - vel_2 \cdot t_2)\,] \wedge \\
\quad vel_1 < Min(Topspeed(car_1), Limit(segm)) \wedge \neg Del(car) \wedge \\
\quad HoldsIn(StartTime(t_0), z_1) \wedge \\
\quad z_2 \circ Movement(car_1, dist_1, vel_1, t_1, segm, node) = \\
\qquad z_1 \circ Movement(car_1, dist_1 + vel_1 \cdot (t_0 - t_1), \\
\qquad\qquad\qquad Min(Topspeed(car), Limit(segm)), t_0, segm, node) \\
\quad \vee \ldots
\end{array}
$$

The 'obsolete' state: In z_1, non-deliberative car_1 is moving slower than it could although no car in front is in ς-distance.

The updated state: In z_2, car_1 has changed its velocity.

5.3 First Car Gets Released

$$
\begin{array}{l}
Causes(z_1, z_2) \leftrightarrow \\
\quad \ldots \vee \\
\quad \neg HoldsIn(Blocked(node, segm), z_1) \wedge \\
\quad HoldsIn(Waiting(car, segm, node, 0), z_1) \wedge \\
\quad HoldsIn(StartTime(t), z_1) \wedge \\
\quad z_2 \circ Counter(node, segm, n) \circ Waiting(car, node, segm, 0) = \\
\qquad z_1 \circ Blocked(node, segm) \circ Counter(node, segm, n-1) \circ \\
\qquad\quad Movement(car, 0, Min(Topspeed(car), Limit(segm)), \\
\hfill t, segm, node_2) \\
\quad \vee \ldots
\end{array}
$$

The 'obsolete' state: In z_1, a segment is not blocked but there is a car in waiting position 0 for that segment.

The updated state: In z_2, the car is sent on its journey, i.e., a new movement is initiated. The new speed is either the topspeed of the car or the speed limit of the road, whichever is less. (If there is a car in ς-distance and goes even slower, then the causal relationship of section 5.1 applies subsequently.) At the same time, the road gets blocked, the car at waiting position 0 is not there anymore, and the counter is reduced.

5.4 Waiting Queue Gets Updated

The happy event of reducing the waiting queue is implemented by propagating the empty spot, thus updating the waiting queue.

$$
\begin{array}{l}
Causes(z_1, z_2) \leftrightarrow \\
\quad \ldots \vee \\
\quad HoldsIn(Waiting(car_2, node, segm, n+1), z_1) \wedge \\
\quad \neg HoldsIn(Waiting(car_1, node, segm, n), Z_1) \wedge \\
\quad z_2 \circ Waiting(car_2, node, segm, n+1) = \\
\qquad z_1 \circ Waiting(car_2, node, segm, n)
\end{array}
$$

The 'obsolete' state: In z_1, a car is waiting at the $n+1$st position and space no. n is available.

The updated state: In z_2, the car has moved from position $n+1$ to n.

Notice that should one driver decide not to wait any longer in the queue (for whatever reason[2]), then the relationship above will automatically "fill" this hole, too. [3]

[2] E.g., an agent facing a long waiting time could activate some replanning strategy, and turn around as a consequence.

[3] Alternatively one could design a virtual waiting road where incoming cars are placed at the very end of the queue but already respecting the ς-distance. In this case the traffic jam rules could take care of the catching-up procedure. This approach seems less realistic and less ecological, though. Furthermore, placing a slow car behind a long fast convoy would yield an unpleasant gap.

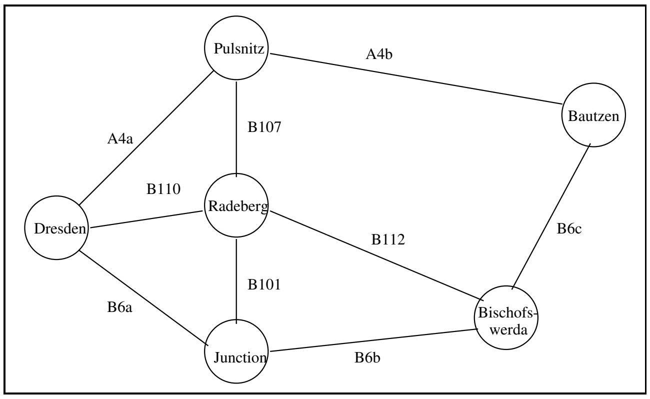
6 Scenario Worlds

6.1 Unique Name Assumptions

As cars, segments and nodes should have unique names, it is necessary to introduce the respective axioms (using the notation of [2]):

$$UNA[car_1, ...car_n]$$
$$UNA[segm_1, ...segm_o]$$
$$UNA[node_1, ...node_p]$$

6.2 Static Scenario Knowledge



The road segments specification: The road segments connect the nodes, and they have both a certain length and a certain speed limit, expressed with the following predicates and functions, respectively:

Connections	Length	Limit
$Connects(A4_a, Dr, Pu)$	$Length(A4_a) = 24$	$Limit(A4_a) = 110$
$Connects(A4_b, Pu, Ba)$	$Length(A4_b) = 41$	$Limit(A4_b) = 130$
$Connects(B6_a, Dr, Ju)$	$Length(B6_a) = 18$	$Limit(B6_a) = 60$
$Connects(B6_b, Ju, Bi)$	$Length(B6_b) = 22$	$Limit(B6_b) = 100$
$Connects(B6_c, Bi, Ba)$	$Length(B6_c) = 20$	$Limit(B6_c) = 100$
$Connects(B110, Dr, Rb)$	$Length(B110) = 12$	$Limit(B110) = 50$
$Connects(B112, Rb, Bi)$	$Length(B112) = 35$	$Limit(B112) = 80$
$Connects(B101, Pu, Rb)$	$Length(B101) = 7$	$Limit(B101) = 80$
$Connects(B107, Rb, Ju)$	$Length(B107) = 10$	$Limit(B107) = 80$

Furthermore the following rule expresses symmetry:

$$Connects(segm, node_1, node_2) \leftrightarrow Connects(segm, node_2, node_1)$$

The cars specification:

$$\begin{aligned} &Del(car) \leftrightarrow \\ &\qquad car = OD\_KA\_174 \vee \\ &\qquad car = DD\_OM\_362 \vee \\ &\qquad car = Tractor_1 \end{aligned}$$

$$
\begin{aligned}
&\mathit{Topspeed}(OD\_KA\_174) = 125 \\
&\mathit{Topspeed}(DD\_OM\_362) = 160 \\
&\mathit{Topspeed}(Tractor_1) = 40 \\
&\mathit{Topspeed}(RoboCar_1) = 100 \\
&\mathit{Topspeed}(RoboCar_2) = 80 \\
&\mathit{Topspeed}(RoboCar_3) = 60
\end{aligned}
$$

Furthermore, a value for the safety distance ς needs to be specified:[4]

$$\varsigma = 50m$$

7 Example situations

7.1 Heavy Traffic

A number of convoys approaching a node from different directions, all keen on the same road segment. They all will neatly be placed in the parking area in the order they arrived at the node—a fairness you will hardly see in real life.

7.2 A construction area

The above axiomatization has the potential for several extensions, e.g. the simulation of an obstacle, which arose all of a sudden. An easy hack would be to place a movement with velocity 0 somewhere outside anyones safety distance. Likewise, a car failure/break-down can be treated similarly.

7.3 Disappearing cars

Some of car drivers happiest moments are those, where tractors finally turn into the field, and thus should be modelled here, too. This would just be a matter of removing the concerning *Movement*-fluent, since the speed up of the following cars is already taken care of (cf. section 5.2).

A Standard Elements of the Signature

The Fluent Calculus is an order-sorted second-order language with equality. The standard sorts are FLUENT, STATE, ACTION, CONCURRENT, SIT, and TIME, where FLUENT is a sub-sort of STATE and ACTION is a sub-sort of CONCURRENT. We use the variables f, z, a, c, s, and t, resp., possibly with sub- or superscripts, for these sorts. The elements of sort TIME are the real numbers along with their standard interpretation.

[4] In a more detailed model the distance should be dynamic, i.e. depending on the car's velocity.

The Fluent Calculus uses the following standard functions:

$$
\begin{array}{rl}
\emptyset : & \mapsto \textsc{State} \\
\circ : & \textsc{State} \times \textsc{State} \mapsto \textsc{State} \\
\mathit{State} : & \textsc{Sit} \mapsto \textsc{State} \\
\epsilon : & \mapsto \textsc{Concurrent} \\
\circ : & \textsc{Concurrent} \times \textsc{Concurrent} \mapsto \textsc{Concurrent} \\
\mathit{Do} : & \textsc{Concurrent} \times \textsc{Sit} \mapsto \textsc{Sit} \\
\mathit{DSucc} : & \textsc{Concurrent} \times \textsc{State} \mapsto \textsc{State} \\
\mathit{Succ} : & \textsc{Concurrent} \times \textsc{State} \mapsto \textsc{State} \\
\mathit{Time} : & \textsc{Concurrent} \mapsto \textsc{Time} \\
\mathit{StartTime} : & \textsc{Time} \mapsto \textsc{Fluent}
\end{array}
$$

and the following standard predicates:

$$
\begin{array}{rl}
\mathit{Poss} : & \textsc{Concurrent} \times \textsc{State} \\
\mathit{Cancels} : & \textsc{Concurrent} \times \textsc{Concurrent} \times \textsc{State} \\
\mathit{Causes} : & \textsc{State} \times \textsc{State} \\
\mathit{Ramify} : & \textsc{State} \times \textsc{State} \\
\mathit{Natural} : & \textsc{Action} \\
\mathit{Trajectory} : & \textsc{State} \times \textsc{State}
\end{array}
$$

There are three differences to the reference articles [4], [5], and [6] in these definitions:

- Function $DSucc(c, z)$ (read: "direct successor state") is a new predicate by which is combined continuous change with ramifications: It denotes the state resulting from the direct effects of the (concurrent) action c in state z. The overall successor, $Succ(c, z)$, is obtained via ramification.
- Predicates $Causes$ and $Ramify$ are just binary because the Traffic World requires only a simplified solution to the Ramification Problem.
- Most functions and predicates carry the sort CONCURRENT instead of ACTION, to accommodate concurrency, and STATE instead of SIT, to accommodate continuous change and natural actions.

B Foundational Axioms

B.1 The Equational Foundation

$$State(s) \neq f \circ f \circ z$$

$$
\begin{array}{rcl}
(z_1 \circ z_2) \circ z_3 & = & z_1 \circ (z_2 \circ z_3) \\
z_1 \circ z_2 & = & z_2 \circ z_1 \\
z \circ \emptyset & = & z
\end{array}
$$

$$z = f \rightarrow z \neq \emptyset \wedge [\, z = z' \circ z'' \rightarrow z' = \emptyset \vee z'' = \emptyset \,]$$

$$
\begin{array}{l}
z_1 \circ z_2 = z_3 \circ z_4 \rightarrow \\
\quad \exists z_a, z_b, z_c, z_d.\ z_1 = z_a \circ z_b \wedge z_2 = z_c \circ z_d \wedge z_3 = z_a \circ z_c \wedge z_4 = z_b \circ z_d
\end{array}
$$

(Note: As opposed to the axioms of AC1-unification completeness, used in previous versions of the Fluent Calculus to enable conclusions regarding inequality of states, the last two axioms allow to incorporate domain dependent equalities as well as assumptions of unique names [3].)

B.2 Concurrency

$$
\begin{array}{rcl}
(c_1 \circ c_2) \circ c_3 & = & c_1 \circ (c_2 \circ c_3) \\
c_1 \circ c_2 & = & c_2 \circ c_1 \\
c \circ \emptyset & = & c
\end{array}
$$

$$c = a \rightarrow c \neq \emptyset \wedge [\, c = c' \circ c'' \rightarrow c' = \epsilon \vee c'' = \epsilon \,]$$

$$
\begin{array}{l}
c_1 \circ c_2 = c_3 \circ c_4 \rightarrow \\
\quad \exists c_a, c_b, c_c, c_d.\ c_1 = c_a \circ c_b \wedge c_2 = c_c \circ c_d \wedge c_3 = c_a \circ c_c \wedge c_4 = c_b \circ c_d
\end{array}
$$

$$\mathit{State}(\mathit{Do}(\epsilon, s)) = \mathit{State}(s)$$

B.3 Ramification

$$
\mathit{Ramify}(z, z') \leftrightarrow \forall \Pi \left\{
\begin{array}{c}
\forall z_1.\, \Pi(z_1, z_1) \\
\wedge \\
\left[\begin{array}{l} \forall z_1, z_2, z_3. \\ \quad \Pi(z_1, z_2) \wedge \mathit{Causes}(z_2, z_3) \rightarrow \Pi(z_1, z_3) \end{array} \right] \\
\rightarrow \\
\Pi(z, z')
\end{array}
\right\}
$$

$$\mathit{Poss}(c, z) \rightarrow \mathit{Ramify}(\mathit{DSucc}(c, z), \mathit{Succ}(c, z))$$

B.4 Continuous Change and Natural Actions

$$\mathit{TimeUniform}(c) \rightarrow \mathit{HoldsIn}(a, c) \rightarrow \mathit{Time}(c) = \mathit{Time}(a)$$

$$
\begin{array}{l}
\neg(\exists)(\mathit{HoldsIn}(\mathit{StartTime}(t_1) \circ \mathit{StartTime}(t_2), z)) \rightarrow \\
\qquad \mathit{HoldsIn}(\mathit{StartTime}(t), z) \rightarrow \mathit{Start}(z) = t
\end{array}
$$

$$Start(s) = Start(State(s))$$

$$Holds(StartTime(t_1), s) \wedge Holds(StartTime(t_2), s) \\ \rightarrow t_1 = t_2$$

C Macros

$$HoldsIn(z_1, z) \stackrel{\text{def}}{=} \exists\, z'.z = z_1 \circ z'$$

$$Holds(f, s) \stackrel{\text{def}}{=} HoldsIn(f, State(s))$$

$$HoldsIn(c_1, c) \stackrel{\text{def}}{=} \exists\, c'.c = c_1 \circ c'$$

$$TimeUniform(c) \stackrel{\text{def}}{=} (\forall a, a')(HoldsIn(a, c) \wedge HoldsIn(a', c) \\ \rightarrow Time(a) = Time(a'))$$

$$ActualState(s, t, z) \stackrel{\text{def}}{=} Trajectory(State(s), z) \wedge t > Start(z) \wedge \\ (\forall a)(Natural(a) \wedge Poss(a, z) \rightarrow Time(a) \geq t)$$

$$ExpectedNatAct(z, t, c) \stackrel{\text{def}}{=} (\forall a)(HoldsIn(a, c) \leftrightarrow Poss(a, z) \wedge Natural(a) \wedge \\ Time(a) = t) \wedge (\forall z')c \neq a \circ a \circ z'$$

$$Poss(c, s) \stackrel{\text{def}}{=} (\forall a)(HoldsIn(a, c) \rightarrow \neg Natural(a)) \wedge TimeUniform(c) \\ (\forall z, c')(ActualState(s, Time(c), z) \\ \rightarrow ExpectedNatAct(z, Time(c), c') \\ \rightarrow Poss(c \circ c', z))$$

D State Update Axioms

State Update axioms for continuous change, concurrency, and ramification have the following form:

$$
\begin{aligned}
&Poss(A(\vec{x}) \circ c, z) \,\wedge\, \Delta(\vec{x}, c, z) \rightarrow \\
&\quad (\exists)\ DSucc(A(\vec{x}) \circ c, z) \circ \vartheta^- \\
&\qquad = DSucc(c, z) \circ \vartheta^+
\end{aligned}
$$

where ϑ^+ and ϑ^- are the positive and negative, resp., direct effects of action A in state z under condition Δ.

References

[1] `http://www.ida.liu.se/ext/etai/lmw/`.

[2] A. B. Baker. A simple solution to the Yale Shooting problem. In R. Brachman, H. J. Levesque, and R. Reiter, editors, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 11–20, Toronto, Kanada, 1989. Morgan Kaufmann.

[3] Hans-Peter Störr and Michael Thielscher. A new equational foundation for the fluent calculus. 1999. (Submitted).

[4] Michael Thielscher. Introduction to the Fluent Calculus. *Electronic Transactions on Artificial Intelligence*, 2(3–4):179–192, 1998. URL: `http://www.ep.liu.se/ea/cis/1998/014/`.

[5] Michael Thielscher. Fluent Calculus planning with continuous change. *Electronic Transactions on Artificial Intelligence*, 1999. (Submitted.) URL: `http://www.ep.liu.se/ea/cis/1999/011/`.

[6] Michael Thielscher. Solving the inferential frame problem for concurrent actions. 1999. (Submitted.) `http://pikas.inf.tu-dresden.de/~mit/publications/conferences/1-KR00.ps`.