A Representation of the Traffic World in the Language of the Causal Calculator

Varol Akman
Department of Computer Engineering
Bilkent University
akman@cs.bilkent.edu.tr

Selim T. Erdoğan, Joohyung Lee and Vladimir Lifschitz
Department of Computer Sciences
University of Texas at Austin
{selim,appsmurf,vl}@cs.utexas.edu

Abstract

The Traffic World is an action domain proposed by Erik Sandewall as part of his Logic Modelling Workshop. We show how to represent this domain in the input language of the Causal Calculator.

1 Introduction

The Traffic World is an action domain proposed by Erik Sandewall as part of his Logic Modelling Workshop[1]. We show how to represent this domain in the input language of the Causal Calculator (CCALC)[2]. A representation of another Workshop domain—the Zoo World—in the same language is discussed in a companion paper [Lee *et al.*, 2001]. We assume that the reader is familiar with the review of CCALC and of the closely related action language $\mathcal{C}+$ in Sections 3—5 of that paper.

The next section contains an extensive quote from the Logic Modelling Workshop description of the Traffic World. Then we discuss the possibility of describing continuous motion using integer arithmetic (Section 3) and the use of action languages for representing change in the absence of actions (Section 4). Our formalization of the Traffic World is presented in Section 5 and examples of how we tested it are shown in Section 6. Finally, Section 7 compares the formalization with related work.

2 Sandewall's Description of the Traffic World

Here is the Logic Modelling Workshop description that we want to formalize:

The TRAFFIC scenario world is intended to capture simple hybrid phenomena: vehicles moving continuously with well defined velocities along roads with well defined lengths, respecting speed limits and other restrictions on the vehicle's behaviors.

The landscape in the TRAFFIC Scenario World uses the following two types:

- *Nodes*, which can be thought of as road crossings without any particular structure (no lanes, etc)
- *Segments*, which can be thought of as road segments each of which connects two nodes.

The set of nodes and the set of segments are both considered as fully known, and all nodes and segments can be assigned individual names.

Each segment has exactly one *start node* and exactly one *end node*. It also has a *length*, which is a real number (or rational number, if preferred). This is all the structure there is.

...The activity structure in the TRAFFIC world uses only one sort:

- *Cars*, which are intuitively thought of as driving along the arcs in the TRAFFIC landscape structure.

Each car has a *position* at each point in time. The position is indicated as a pair consisting of the segment where the car is located, and the distance travelled along the segment. The distance travelled is a number between 0 and the segment's length.

Each car has a *top speed*, and each road segment has a *speed limit*. The actual velocity of a car at each point in time is the maximum velocity allowed by the following three conditions:

- The speed limit of the road segment where it is driving
- Its own top speed
- Surrounding traffic restrictions

Cars drive at piecewise constant velocity, and can change velocity discontinuously. (A more refined variant, TRAFFIC2, will require cars to change their velocity continuously, and assumes piecewise constant acceleration/deceleration). When a car arrives at a node ("intersection") then it may continue on any segment that connects to that node, except the one it is arriving at.

Cars can drive in both "directions" along a segment, that is, they can move both from the start node to the end node, and vice versa.

Cars can not overtake — if two cars go in the same direction on the same road segment, and one catches up with the other, then it has to stay behind at least until they arrive to the next node, where possibly the second car can choose another direction onwards. Cars going in opposite directions on the same segment can meet without difficulty, however.

[1] http://www.ida.liu.se/ext/etai/lmw/ .
[2] http://www.cs.utexas.edu/users/tag/cc/ .

The surrounding traffic restriction says that a car is never allowed to be closer than a fixed safety distance *varsigma* to the car in front of it, and it may never get itself into a situation where that could happen. This means, first of all, that when it gets to a distance of *varsigma* to a car moving in front of it on the same segment and in the same direction, then it must reduce speed to match the speed of the car in front of it. Also when getting close to a node (= an intersection), a car must reduce its speed in a way that takes into consideration all other cars that are just approaching or leaving the same node.

3 Continuous Motion and Integer Arithmetic

Under some special circumstances, questions about continuous motion can be discussed using integers, without ever mentioning reals or even rational numbers. Such special circumstances are assumed in our formalization of the Traffic World. Specifically, we assume that

- the lengths of all road segments and the safety distance *varsigma* are expressed by integers,
- the top speeds of all cars and the speed limits on all road segments are expressed by integers,
- in the scenarios under consideration, the times when cars leave and reach endpoints of road segments, and the times at which the distance between two cars traveling on the same segment in the same direction becomes *varsigma* are expressed by integers.

These constraints are similar to those adopted in the description of the spacecraft Integer in [Lee and Lifschitz, 2001]:

> Far away from stars and planets, the Integer is not affected by any external forces. As its proud name suggests, the mass of the spacecraft is an integer. For every integer t, the coordinates and all three components of the Integer's velocity vector at time t are integers; the forces applied to the spacecraft by its jet engines over the interval $(t, t+1)$, for any integer t, are constant vectors whose components are integers as well. If the crew of the Integer attempts to violate any of these conditions, the jets will fail to operate!

The spacecraft Integer is actually close to the refined version of the Traffic World mentioned in Section 2 above, in the sense that its velocity changes continuously, and its acceleration is piecewise constant.

For query answering on the basis of our formalization of the Traffic World, it is essential that all time instants mentioned in the queries be expressed by integers.

Assumptions like these make it easier to describe motion in the action languages whose semantics is defined in the framework of transition systems [Gelfond and Lifschitz, 1998]. The use of these languages for describing motion without such simplifying assumptions is a topic for future research.

4 Change in the Absence of Actions

The only actions performed in the Traffic World are those related to selecting a new road segment when a car reaches an intersection. In our formalization, these actions are denoted by $\mathit{ChooseSegment}(c, sg)$—the driver of car c chooses to turn into road segment sg. Between intersections every car is assumed to be moving at the maximum speed allowed by the road conditions, so that the drivers are not permitted to perform any actions affecting the speeds of their cars.

The speeds of cars may change many times, however, during a time interval that does not include the execution of actions. Consider, for instance, a long road segment with a high speed limit, and two cars moving along it in the same direction. If the top speed of the car in front is lower than the top speed of the second car then the latter will slow down at some point to match the speed of the slower car. If there are several cars on the road moving in the same direction then the surrounding traffic restriction may force the last of them to reduce its speed several times, although no actions will be executed. (An example of such a scenario is shown in Section 6.)

A similar phenomenon will be observed when several cars have simultaneously (or almost simultaneously) approached the same node from different directions, with the intention to turn into the same road. The cars will have to leave the intersection one by one, after the intervals that will guarantee the safety distance between them. In our formalization, there are no rules determining the order in which the cars are going to depart. This nondeterministic sequence of events may take a long time, and it does not involve the execution of actions.

Change in the absence of actions, so essential in the Traffic World, can be easily described in language $\mathcal{C}+$. (In this respect $\mathcal{C}+$ is similar to its predecessor $\mathcal{C}$ defined in [Giunchiglia and Lifschitz, 1998].) Recall that a "dynamic law" is an expression of the form

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H \tag{1}$$

where F, G and H are formulas (see [Giunchiglia *et al.*, 2001, Section 4] or [Lee *et al.*, 2001, Section 3]). The logical constant $\top$ often plays the role of G, in which case the part **if** G can be dropped. Formula H is allowed to include action symbols, and in many cases it does; then (1) describes direct effects of some actions. For instance, our formalization of the Traffic World describes the effect of action $\mathit{ChooseSegment}(c, sg)$ by the expression

$$\mathit{ChooseSegment}(c, sg) \textbf{ causes } \mathit{NextSegment}(c) = sg$$

which is shorthand for the dynamic law

$$\textbf{caused } \mathit{NextSegment}(c) = sg \textbf{ after } \mathit{ChooseSegment}(c, sg)\,.$$

But dynamic laws containing no action symbols play an important role also. The most widely used kind of dynamic law without action symbols is

$$\textbf{inertial } F$$

which stands for

$$\textbf{caused } F \textbf{ if } F \textbf{ after } F\,.$$

```
:- include 'arithmetic'.
```

(A standard file.)

```
:- sorts
  node;
% The set of objects of the sort segmentOrNone consists of all segments and one
% auxiliary symbol.  This sort is used to declare partial segment-valued fluents
  segmentOrNone >> segment;
  car.

:- variables
  Nd,Nd1                          :: node;
  Sg,Sg1                          :: segment;
  C,C1,C2                         :: car;
  Ds,Ds1,L                        :: integer;
  Sp,Sp1                          :: integer;
  X,Y,Z                           :: computed.
```

(Sort `integer` represents an initial segment of the nonnegative integers and is defined in the standard file `arithmetic`. A variable is declared `computed` if its values are going to be computed by CCALC in the process of grounding.)

Table 1: Formalization

It solves the frame problem by asserting that F "tends to remain true." Dynamic laws that contain no action symbols can also tell us how the world changes when no actions are performed. For instance, the dynamic law

$$\begin{aligned}&\textbf{caused } \mathit{Distance}(c) = ds + sp\\ &\qquad\qquad\textbf{after } \neg\, \mathit{WillLeave}(c)\\ &\qquad\qquad\qquad \wedge \mathit{Distance}(c) = ds \wedge \mathit{Speed}(c) = sp\end{aligned}$$

says that if (i) car c is going to remain on the same road segment during the next time interval, (ii) the distance travelled by c along that segment so far is ds, and (iii) the speed of c during that time interval is going to be equal to sp, then by the end of the time interval the distance travelled by c along the current road segment will become $ds + sp$.

5 Formalization

Our formalization of the Traffic World is shown in Tables 1–6 and is available online[3].

We distinguish between the general assumptions about the Traffic World quoted in Section 2 above and specific details, such as the number and positions of road segments, the numerical values of their speed limits, the number of cars and their top speeds. Here we formalize only the general assumptions, and leave such details unspecified, as in the formalization of the Zoo World in [Lee *et al.*, 2001, Section 5]. A description of all the specifics has to be added to our formalization to get an input file accepted by CCALC.[4]

The annotation `(lmw)` found in many comments below refers to the Logic Modelling Workshop description of the Traffic World (Section 2).

[3] `http://www.cs.utexas.edu/users/tag/cc/traffic.html` .

[4] The computational experiments that we have performed use Version 1.9 of CCALC. Some propositions in this formalization had to be replaced by equivalent, but less concise, propositions that are easier for CCALC to handle.

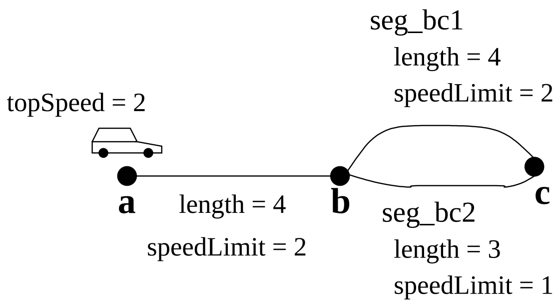

Figure 1: Initial state and landscape for Example 1

As mentioned in the introduction, we assume that the reader is familiar with the features of CCALC reviewed in [Lee *et al.*, 2001].

6 Examples

We tested certain aspects of our formalization by giving CCALC queries about several scenarios and checking that its answers matched our expectations. Here are three examples:

1. Consider 3 roads with the lengths and speed limits shown in Figure 1. The top speed of `car1` *is 2 and it is initially located at node* `a`*. (Note that segment* `seg_bc1` *is long and has a high speed limit whereas segment* `seg_bc2` *is short but its speed limit is low.) Which way must the car go in order to reach* `c` *from* `a` *as soon as possible?*

This planning problem was described as follows:

```
facts ::
  0: position(car1, seg_ab, 0),
  0: positiveOrientation(car1);
goals ::
  1..10: atNode(car1, c).
```

```
:- constants
  none                        :: segmentOrNone;

% If a car is pointing toward the end node of the segment on which it currently is,
%  positiveOrientation will be true
  positiveOrientation(car)   ::  inertialFluent;

% Each car has a position at each point in time. The position is indicated as a pair
%  consisting of the segment where the car is located, and the distance travelled
%  along the segment (lmw)
  segment(car)                :: inertialFluent(segment);
  distance(car)               :: fluent(integer).

:- macros
  position(#1,#2,#3) -> (segment(#1) eq #2 && distance(#1) eq #3).

:- constants
% Each segment has exactly one start node and exactly one end node (lmw)
  startNode(segment)          :: fluent(node);
  endNode(segment)            :: fluent(node);

% Each segment has a length, which is a real number (or rational number,
%  if preferred). (lmw)  We assume it to be an integer
  length(segment)             :: fluent(integer);

% Each road segment has a speed limit (lmw)
  speedLimit(segment)         :: fluent(integer);

% Each car has a top speed (lmw)
  topSpeed(car)               :: fluent(integer);

% Actual speed of a car during the next time interval
  speed(car)                  :: fluent(integer);

% The new segment a car will continue on
  nextSegment(car)            :: inertialFluent(segmentOrNone);

% A car will leave the segment on which it is currently travelling
  willLeave(car)              :: exogenousFluent.
```

(By declaring a fluent "exogenous" we allow it to change its values arbitrarily.)

```
% The landscape and limitations don't change
constant startNode(Sg) eq Nd.
constant endNode(Sg) eq Nd.
constant length(Sg) eq Ds.
constant speedLimit(Sg) eq Sp.
constant topSpeed(C) eq Sp.

% A car will have a positive orientation after leaving the start node of a segment
caused positiveOrientation(C)
   after willLeave(C) && atNode(C,Nd) && nextSegment(C) eq Sg && startNode(Sg) eq Nd.

% A car will have a negative orientation after leaving the end node of a segment
caused -positiveOrientation(C)
   after willLeave(C) && atNode(C,Nd) && nextSegment(C) eq Sg && endNode(Sg) eq Nd.

% Proceed to the next segment if the car was about to leave
caused segment(C) eq Sg after willLeave(C) && nextSegment(C) eq Sg.
```

Table 2: Formalization continued

```
% The distance covered by a car which remained on the same segment
caused distance(C) eq X
   after -willLeave(C) && distance(C) eq Ds && speed(C) eq Sp && sum(X,Ds,Sp).

% The distance covered by a car right after it changed to a new segment
caused distance(C) eq Sp after willLeave(C) && speed(C) eq Sp.

% The time when a car reaches a node is assumed to be an integer
never position(C,Sg,Ds) && length(Sg) eq L && Ds>L.

% No two cars on the same segment and having the same orientation can be closer than
%  varsigma (lmw)
never (positiveOrientation(C)<->positiveOrientation(C1))
      && position(C,Sg,Ds) && position(C1,Sg,Ds1) && -(C=C1)
      && Ds1>=Ds && diff(X,Ds1,Ds) && X<varsigma.

% If a car is waiting at a node and there is a car too close on the next segment
%  it will travel on, the time at which the car in front will reach a distance of
%  varsigma from the node is assumed to be an integer
caused false if position(C1,Sg,Ds1) && Ds1>varsigma
   after position(C1,Sg,Ds) && Ds<varsigma && nextSegment(C) eq Sg
         && (modifiedOrientation(C) <-> positiveOrientation(C1)).
```

(In the language of CCALC, `<->` is equivalence.)

```
:- constants
% A car is at a node
  atNode(car,node)           :: fluent.

atNode(C,Nd) defined as
 (\/Sg:
   ((positiveOrientation(C) && position(C,Sg,0) && startNode(Sg) eq Nd) ++
    (-positiveOrientation(C) && position(C,Sg,0) && endNode(Sg) eq Nd) ++
    (\/L: ((positiveOrientation(C) &&  position(C,Sg,L)
               && length(Sg) eq L && endNode(Sg) eq Nd) ++
           (-positiveOrientation(C) && position(C,Sg,L)
               && length(Sg) eq L && startNode(Sg) eq Nd))))).
```

(This is shorthand for a pair of static laws that define `atNode(C,Nd)` to be equivalent to the formula that follows `defined as`.)

```
:- constants
% For any car in the middle of a segment, its modifiedOrientation has the same value
%  as its positiveOrientation.  If a car has selected a new segment, then
%  modifiedOrientation has the value that positiveOrientation would have if the car
%  were at the beginning of the new segment
  modifiedOrientation(car)   :: fluent.

modifiedOrientation(C) defined as
     (\/Nd: \/Sg:
       ((atNode(C,Nd) && nextSegment(C) eq Sg && startNode(Sg) eq Nd)
        ++ (nextSegment(C) eq none && positiveOrientation(C)))).

:- constants
% The relation between modifiedDistance and distance is similar
  modifiedDistance(car)      :: fluent(integer).

caused modifiedDistance(C) eq Ds if nextSegment(C) eq none && distance(C) eq Ds.

caused modifiedDistance(C) eq 0 if -(nextSegment(C) eq none).
```

Table 3: Formalization continued

```
:- constants
% The relation between modifiedSegment and segment is similar
  modifiedSegment(car)        :: fluent(segment).

caused modifiedSegment(C) eq Sg if nextSegment(C) eq none && segment(C) eq Sg.

caused modifiedSegment(C) eq Sg if nextSegment(C) eq Sg.

:- constants
% Maximum speed allowed by the top speed of a car and the speed limit of the segment
%  on which the car will be travelling
   maxSpeed(car)             :: fluent(integer).

caused maxSpeed(C) eq X
   if topSpeed(C) eq Sp && nextSegment(C) eq none && segment(C) eq Sg
   && speedLimit(Sg) eq Sp1 && X is min(Sp,Sp1).

caused maxSpeed(C) eq X
   if topSpeed(C) eq Sp && nextSegment(C) eq Sg
   && speedLimit(Sg) eq Sp1 && X is min(Sp,Sp1).

:- constants
% Maximum distance a car can cover on the segment on which it will be travelling
  maxDistance(car)           :: fluent(integer).

caused maxDistance(C) eq X
   if modifiedDistance(C) eq Ds && maxSpeed(C) eq Sp && sum(X,Ds,Sp).

:- constants
% The first car will be ahead of the second
  ahead(car,car)             :: fluent.

ahead(C1,C) defined as
  -(C=C1) && (\/Sg:
             ((modifiedOrientation(C)<->positiveOrientation(C1))
                && modifiedSegment(C) eq Sg && segment(C1) eq Sg
                && (\/Ds: \/Ds1:
                     (modifiedDistance(C) eq Ds && distance(C1) eq Ds1 && Ds=<Ds1)))).

% No overtaking
caused false if ahead(C,C1) after ahead(C1,C).

:- constants
% The first car will be ahead of the second car and not farther than varsigma from it
  varsigmaAhead(car,car)           :: fluent.

varsigmaAhead(C1,C) defined as
    ahead(C1,C) && (\/Ds: \/Ds1: (modifiedDistance(C) eq Ds && distance(C1) eq Ds1
                                  && diff(X,Ds1,Ds) && X=<varsigma)).

% The actual velocity of a car at each point in time is the maximum velocity allowed
%  by the following three conditions:
%   - The speed limit of the road segment where it is driving
%   - Its own top speed
%   - Surrounding traffic restrictions (lmw)

% If a car is in the middle of a segment and there is no other car which is varsigma
%  ahead of the car and which will not leave, then it will travel at its maximum speed
caused speed(C) eq Sp
   if nextSegment(C) eq none && maxSpeed(C) eq Sp
   && (/\C1: (varsigmaAhead(C1,C) ->> willLeave(C1))).
```

Table 4: Formalization continued

```
% If a car is in the middle of a segment and there is a car varsigma ahead of it
%  which will not leave, then its speed will be the smaller of its maximum speed and
%  the speed of the car in front
caused speed(C) eq X
   if nextSegment(C) eq none && maxSpeed(C) eq Sp
   && varsigmaAhead(C1,C) && -willLeave(C1) && speed(C1) eq Sp1 && X is min(Sp,Sp1).

% If a car is at the end of a segment and will not leave then it will stay where it is
caused speed(C) eq 0 if -(nextSegment(C) eq none) && -willLeave(C).

% If a car is at the end of a segment and will enter a new segment where there is
%  no car within varsigma, then it will travel at its maximum speed
caused speed(C) eq Sp
   if nextSegment(C) eq Sg && willLeave(C) && maxSpeed(C) eq Sp
   && (/\C1: -varsigmaAhead(C1,C)).

% If a car is at the end of a segment and will enter a new segment where
%  there is a car within varsigma, its speed will be the smaller of its maximum speed
%  and the speed of the car in front
caused speed(C) eq X
   if -(nextSegment(C) eq none) && willLeave(C)
   && varsigmaAhead(C1,C) && maxSpeed(C) eq Sp && speed(C1) eq Sp1 && X is min(Sp,Sp1).

:- constants
% Choose a new segment for a car to proceed
  chooseSegment(car,segment) :: action.

% Direct effect of choosing a new segment
chooseSegment(C,Sg) causes nextSegment(C) eq Sg.

% Cannot choose the segment that is already chosen
nonexecutable chooseSegment(C,Sg) if nextSegment(C) eq Sg.

% When a car arrives at a node ("intersection") then it may continue on any segment
%  that connects to that node... (lmw)
caused false if atNode(C,Nd)
   after o(chooseSegment(C,Sg)) && -(startNode(Sg) eq Nd ++ endNode(Sg) eq Nd).

% ...except the one it is arriving at (lmw)
caused false if nextSegment(C) eq Sg after segment(C) eq Sg.

% A car will choose a new segment to proceed on concurrently with arriving at a node
caused false if atNode(C,Nd)
   after (/\Sg: -o(chooseSegment(C,Sg))) && (/\Nd: -atNode(C,Nd)).

% A car can't have a next segment unless it has travelled to the end of its current
%  segment
caused nextSegment(C) eq none after willLeave(C).
always (/\Nd: -atNode(C,Nd)) ->> nextSegment(C) eq none.
always atNode(C,Nd) && segment(C) eq Sg && startNode(Sg) eq Nd
       && positiveOrientation(C) ->> nextSegment(C) eq none.
always atNode(C,Nd) && segment(C) eq Sg && endNode(Sg) eq Nd
       && -positiveOrientation(C) ->> nextSegment(C) eq none.

% Only cars which have selected a new segment can leave
never willLeave(C) && nextSegment(C) eq none.

% At most one car will leave a node and enter a new segment at each time
never -(C=C1) && nextSegment(C) eq Sg && nextSegment(C1) eq Sg
      && atNode(C,Nd) && atNode(C1,Nd) && willLeave(C) && willLeave(C1).
```

Table 5: Formalization continued

```
% If there is a car at a node which has selected a new segment, and
%  there are no cars within varsigma from the node, then there should be a car
%  which will leave the node (i.e. no unnecessary waiting is allowed)

:- constants
% The following two fluents are used in formalizing the above and they only have
% meaning when a car is at a node. Their values don't affect anything at other times

% The first car is closer than varsigma to the node at which the second car is waiting
  car_within_varsigma(car,car) :: fluent;

% If no cars are within varsigma, then varsigma units of the road in front are free
  varsigma_free(car)           :: fluent.

car_within_varsigma(C1,C) defined as
   (\/Ds1: (varsigmaAhead(C1,C) && distance(C1) eq Ds1  && Ds1<varsigma)).

varsigma_free(C) defined as
   (/\C1: -car_within_varsigma(C1,C)).

always (atNode(C,Nd) && nextSegment(C) eq Sg && varsigma_free(C))
    ->> (\/C2: (atNode(C2,Nd) && nextSegment(C2) eq Sg && willLeave(C2))).

% A car will not leave its current segment if there is another car within varsigma
always -varsigma_free(C) ->> -willLeave(C).
```

Table 6: The end of the formalization

The notation `1..10` tells CCALC to try to find a plan of length 1 and if there is no such a plan to increase the plan length until it finds a plan or the length exceeds 10.

CCALC found the following plan:

```
0:  atNode(car1,a)  nextSegment(car1) eq none
    speed(car1) eq 2  distance(car1) eq 0
    segment(car1) eq seg_ab

1:  nextSegment(car1) eq none  speed(car1) eq 2
    distance(car1) eq 2 segment(car1) eq seg_ab

ACTIONS:  chooseSegment(car1,seg_bc1)

2:  atNode(car1,b)  nextSegment(car1) eq seg_bc1
    speed(car1) eq 2  distance(car1) eq 4
    segment(car1) eq seg_ab

3:  nextSegment(car1) eq none  speed(car1) eq 2
    distance(car1) eq 2  segment(car1) eq seg_bc1

ACTIONS:  chooseSegment(car1,seg_bc2)

4:  atNode(car1,c)  nextSegment(car1) eq seg_bc2
    speed(car1) eq 1  distance(car1) eq 4
    segment(car1) eq seg_bc1
```

2. The top speeds of 3 cars and their initial positions are as shown at the top diagram of Figure 2. Cars `car1`, `car2` *and* `car3` *have top speeds of 3, 2 and 1, respectively. Assuming the speed limit is 3 and the safety distance is 1, how will the cars move during the next 3 time instants?*

The facts in this case were:

```
time  :: 3;
facts ::
  0: position(car1, seg_ab, 0),
  0: positiveOrientation(car1),
  0: position(car2, seg_ab, 2),
  0: positiveOrientation(car2),
  0: position(car3, seg_ab, 5),
  0: positiveOrientation(car3).
```

CCALC found a scenario consistent with these facts (Figure 2):

```
0:  distance(car1) eq 0  distance(car2) eq 2
    distance(car3) eq 5  speed(car1) eq 3
    speed(car2) eq 2  speed(car3) eq 1

1:  distance(car1) eq 3  distance(car2) eq 4
    distance(car3) eq 6  speed(car1) eq 2
    speed(car2) eq 2  speed(car3) eq 1

2:  distance(car1) eq 5  distance(car2) eq 6
    distance(car3) eq 7  speed(car1) eq 1
    speed(car2) eq 1  speed(car3) eq 1

3:  distance(car1) eq 6  distance(car2) eq 7
    distance(car3) eq 8  speed(car1) eq 1
    speed(car2) eq 1  speed(car3) eq 1
```

3. The landscape and the initial positions of two cars are shown at the top of Figure 3. The top speeds of both cars and the speed limits for all segments are 1. The safety distance is 2. What are the plans in which both cars will be on segment `seg_cd` *at time 5?*

The facts and goals for this problem were:

```
facts ::
 0: position(car1,seg_ac,1),
 0: position(car2,seg_bc,1),
 0: (/\C: positiveOrientation(C));
goals ::
```

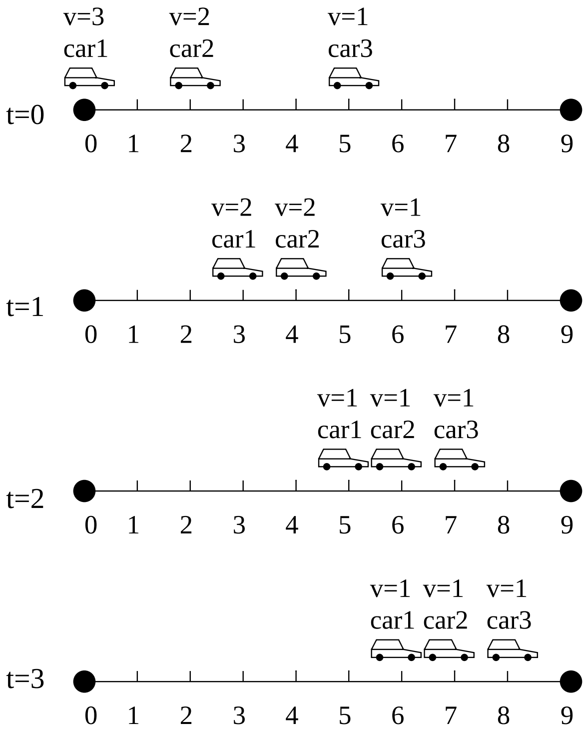

Figure 2: Example 2

```
5: (/\C: segment(C) eq seg_cd).
```

For this example we instructed CCALC to find all possible plans and it returned two plans. We show only the first plan here (Figure 3):

```
Plan 1:


0:  nextSegment(car1) eq none
    nextSegment(car2) eq none
    speed(car1) eq 1 speed(car2) eq 1
    distance(car1) eq 1  distance(car2) eq 1
    segment(car1) eq seg_ac  segment(car2) eq seg_bc

1:  nextSegment(car1) eq none
    nextSegment(car2) eq none
    speed(car1) eq 1 speed(car2) eq 1
    distance(car1) eq 2  distance(car2) eq 2
    segment(car1) eq seg_ac  segment(car2) eq seg_bc

ACTIONS:  chooseSegment(car1,seg_cd)
          chooseSegment(car2,seg_cd)

2:  atNode(car1,c)  atNode(car2,c)  willLeave(car2)
    nextSegment(car1) eq seg_cd
    nextSegment(car2) eq seg_cd
    speed(car1) eq 0  speed(car2) eq 1
    distance(car1) eq 3  distance(car2) eq 3
    segment(car1) eq seg_ac segment(car2) eq seg_bc

3:  atNode(car1,c)  nextSegment(car1) eq seg_cd
    nextSegment(car2) eq none speed(car1) eq 0
    speed(car2) eq 1  distance(car1) eq 3
    distance(car2) eq 1  segment(car1) eq seg_ac
    segment(car2) eq seg_cd

4:  atNode(car1,c)  willLeave(car1)
    nextSegment(car1) eq seg_cd
    nextSegment(car2) eq none  speed(car1) eq 1
    speed(car2) eq 1  distance(car1) eq 3
    distance(car2) eq 2  segment(car1) eq seg_ac
    segment(car2) eq seg_cd

5:  nextSegment(car1) eq none
    nextSegment(car2) eq none  speed(car1) eq 1
    speed(car2) eq 1  distance(car1) eq 1
    distance(car2) eq 3  segment(car1) eq seg_cd
    segment(car2) eq seg_cd
```

In the second plan the states at times 0 and 1 and the actions performed between times 1 and 2 are the same. For times 2 through 5, everything is the same except that `car1` is replaced by `car2` everywhere, and vice versa.

The fact that the same actions lead to different states illustrates the nondeterminism in the transition system to which the formalization corresponds. This nondeterminism arises when more than one car is waiting to enter a new segment. In such situations the car which will

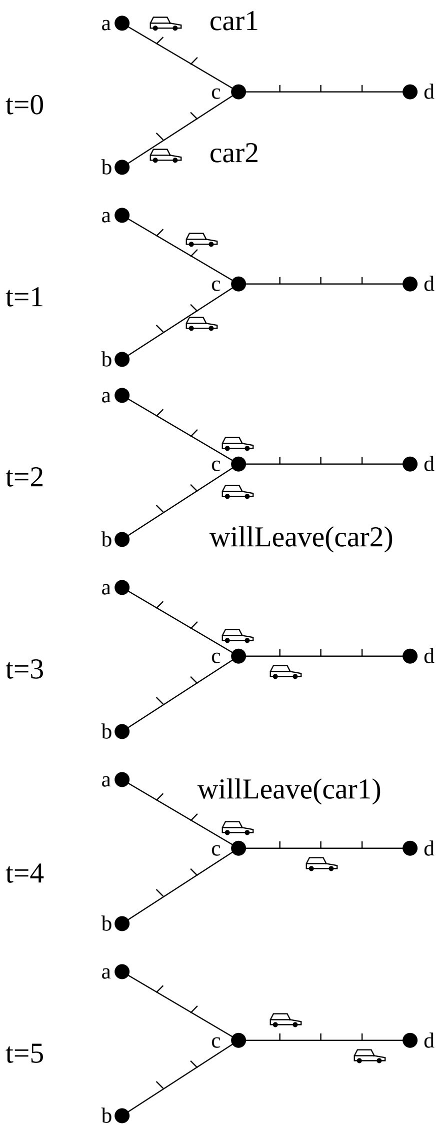

Figure 3: Example 3, plan 1

leave the node is chosen nondeterministically so, as in this example, there may be multiple scenarios in which the order of cars leaving the node is different.

7 Related Work

Henschel and Thielscher [1999] showed how to formalize the Traffic World in the fluent calculus [Thielscher, 1998]. They do not assume that speeds and lengths are expressed by integers, as we do (Section 3). Other than that, their representation differs from ours in the following ways:

- Instead of having all cars obey the same rules, cars are divided into two groups: "deliberative" cars and "non-deliberative" cars. Drivers of deliberative cars can set their own speeds (using an action to change the speed of a car) instead of having to go at the maximum speed possible, and are allowed to wait at nodes. Non-deliberative cars are just like cars in our paper, as described in the Logic Modelling Workshop specification (Section 2).
- There is a "waiting area" associated with each node-segment pair. In this area, the cars that are going to enter the segment wait in line until there are no cars within *varsigma* from the node. When the new segment becomes free, the first car in the waiting area is allowed to leave.
- At any time, a car is in exactly one of three states: moving on a segment, at a node, or in a waiting area. Cars which are at a node or in a waiting area are not considered to be on segments. So after a car arrives at a node, the cars following it are allowed to arrive at the node too, even if the car remains there.
- For each node, the segments leading to it are assigned priorities. When several drivers simultaneously decide to turn into the same segment, their cars are placed in the waiting area in the order determined by the priorities of the segments they arrived from.

If a car is considered to be on some road segment at all times, as in our formalization, the number of cars that can approach a node is limited by the number of roads leading to that node. The view adopted in [Henschel and Thielscher, 1999], on the other hand, makes a car at a node exempt from the surrounding traffic restriction, so that the number of cars that can gather at a node (or in a waiting area) is unlimited. Perhaps we model streets in a city, and Henschel and Thielscher model roads between towns. It would not be difficult to modify our formalization to include deliberative cars and waiting areas; see [Erdoğan, 2000, Section 5.3] for a related discussion.

Acknowledgements

The work of the fourth author was partially supported by National Science Foundation under grant IIS-9732744.

References

[Erdoğan, 2000] Selim T. Erdoğan. Formalization of the TRAFFIC world in the $\mathcal{C}$ action language.[5] Master's thesis, Bilkent University, 2000.

[Gelfond and Lifschitz, 1998] Michael Gelfond and Vladimir Lifschitz. Action languages.[6] *Electronic Transactions on AI*, 3:195–210, 1998.

[Giunchiglia and Lifschitz, 1998] Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal explanation: Preliminary report. In *Proc. AAAI-98*, pages 623–630. AAAI Press, 1998.

[Giunchiglia *et al.*, 2001] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, and Hudson Turner. Causal laws and multi-valued fluents.[7] Unpublished draft, 2001.

[Henschel and Thielscher, 1999] Andreas Henschel and Michael Thielscher. The LMW Traffic world in the fluent calculus.[8] 1999.

[Lee and Lifschitz, 2001] Joohyung Lee and Vladimir Lifschitz. Additive fluents.[9] In *Working Notes of the AAAI Spring Symposium on Answer Set Programming*, 2001.

[Lee *et al.*, 2001] Joohyung Lee, Vladimir Lifschitz, and Hudson Turner. A representation of the zoo world in the language of the causal calculator.[10] To appear in *Common Sense 2001*.

[Thielscher, 1998] Michael Thielscher. Introduction to the fluent calculus.[11] *Electronic Transactions on AI*, 3, 1998.

[5] `http://www.cs.utexas.edu/users/selim/publications/tezdeneme.ps.gz` .

[6] `http://www.ep.liu.se/ea/cis/1998/016/` .

[7] `http://www.cs.utexas.edu/users/vl/mypapers/clmvf-long.ps` .

[8] `http://www.ida.liu.se/ext/epa/cis/lmw/001/paper.ps` .

[9] `http://www.cs.utexas.edu/users/vl/mypapers/additive.ps` .

[10] `http://www.cs.utexas.edu/users/vl/mypapers/zoo.ps` .

[11] `http://www.ep.liu.se/ea/cis/1998/014/` .