Natural Language Texts for a Cognitive Vision System

Michael Arens and **Artur Ottlik** and **Hans-Hellmut Nagel** [1]

Abstract. Text-to-logic conversion is studied in a system approach which extracts a conceptual representation of temporal developments within a road traffic scene recorded by a video camera. A Fuzzy Metric Temporal Horn Logic (FMTHL) facilitates a schematic representation of road vehicle behavior at intersections. Geometric results of a model-based vehicle detection and tracking subsystem are used to interpret this generic conceptual FMTHL representation in order to obtain a conceptual description of the specific developments in the recorded traffic scene.

One task accepts a natural language (NL) English text formulation of the generic conceptual knowledge and converts this into the internal FMTHL representation. This requires to distinguish algorithmically between discourse-related and scheme-related NL statements. A separate second task creates a synthetic video sequence from a NL text, using as much as possible the same apparatus as the first one. A comparison between genuine road traffic video sequences and those generated synthetically allows to test both the text-to-logic transformation and the use of geometric knowledge represented within the entire system for image sequence evaluation and subsequent interpretation. Results obtained by the execution of both tasks are discussed.

1 INTRODUCTION

An (artificial) *cognitive vision system* does not only evaluate image sequences up to the level of numerical results, it should also derive a *conceptual* description of temporal developments in the recorded scene [14]. Such a task requires schematic knowledge about the domain of discourse, i. e. about the – stationary and time-dependent – scene geometry and about the temporal evolution within the recorded scene. It will be an advantage if the user of such a system can provide the required knowledge *without help by a systems specialist*. We thus proceed on the hypothesis that a user prefers to provide the required knowledge *in his own words* – see, for example, [16].

We thus need an automatic transformation process deriving a machine–usable representation of knowledge from a natural language text input. In our system approach, the knowledge about the behavior of agents is represented in form of *situation graph trees* (SGTs) as described in [17]. These graphs consist of *situation schemes* (nodes), which describe the state of an agent and its environment at one discrete point of time – corresponding, e. g., to a (half-) frame of an image sequence – and the action the agent is supposed to carry out in that state. These schemes are connected by *prediction edges* leading from one scheme to (potential) successor schemes for the next time step, building a directed *situation graph*. In addition to these prediction edges, *specialization edges* can connect a situation scheme with another situation graph, which means that the situation is temporally or conceptually described by this additional graph in a more detailed way. These specialization edges thus establish a tree–like structure with more detailed schematic descriptions at the leaf nodes. For an automatic exploitation of these *behavior schemes*, the SGTs are transformed into a logic program of a *fuzzy metric temporal Horn logic* (FMTHL [17, 8]). An automatic transformation of a natural language description of schematic behavior knowledge therefore aims at this representation of an SGT described in a logic program as a result.

It turns out to be desirable for testing purposes to be able to *invert* this process, namely to generate a synthetic image sequence by analyzing a text: an *original image sequence* (OIS) is evaluated resulting in a natural language textual description, based on *schematic* knowledge derived from natural language texts. The result of this process is visualized in the form of a *synthetic image sequence* (SIS), which can be compared with the OIS. In this way, contradictions and shortcomings of the machine vision system, the knowledge base and the visualization process should be recognized much easier.

For both tasks – the automatic acquisition of behavioral knowledge from natural language texts and the analysis of natural language texts for generating synthetic image sequences – we use the same system approach and the same intermediate representations in form of FMTHL–programs.

2 RELATED WORK

A survey of system approaches creating high–level descriptions from image sequences can be found in [9]. Several different knowledge representation formalisms are used in the systems cited there – both for knowledge about *static* and *time-varying* scene components. [4] propose a machine–learning approach to acquire knowledge about events in the observed scene: sequences of primitive object–relations are automatically extracted and clustered during a learning phase. These clustered sequences are used to classify new sequences of object–relations. [1] also apply a machine–learning approach, on the basis of *Conceptual Spaces* (see [5]) and time–delay neural networks. As pointed out in [4], one could criticize the machine–learning approach in two points: first, a *sufficient* training set has to be provided covering everything the system is meant to learn, and secondly, it is not always clear how the training method influences the resulting knowledge obtained from the training set. In addition, the association of learned *events* with linguistic concepts has to be established by the human operator or user again [4]. In [1], only the association of basic concepts with occurrencies in the scene is learned by the system. A knowledge base (*'Linguistic Area'*), which describes linkages between basic concepts has to be provided explicitly by the user, too.

For the visualization of texts several approaches can be found. The project *WordsEye* (see [2]) visualizes texts which contain descriptions of static as well as dynamic aspects of a scene. The main focus, however, lies on the static relations, i.e. the look of objects

[1] Institut für Algorithmen und Kognitive Systeme, Fakultät für Informatik, Universität Karlsruhe (TH), Postfach 6980, 76128 Karlsruhe, Germany

(e.g., color, size) and spatial relations between objects. The *Kairai* virtual actor system (see [18]) is designed in order to understand natural language instructions from verbal and written input whereby the user can control software robots within a given virtual world. In the project *Ulysse* [7] the user can utter instructions in order to navigate in a virtual world. The *Virtual Director* project (see [13]) analyses both static and dynamic aspects of a scene in a park domain described in the text. The system *CarSim* (see [3]) generates a 3D simulation from a text describing a car accident.

3 TEXT TRANSFORMATION

The process which transforms a natural language input text into an FMTHL–representation of the knowledge given by the text is depicted in Figure 1. Each (intermediate) representation of the discourse entered as a text is shown as a rectangle, while process steps which transform one representation into another one are depicted as numbered hexagons. Additional knowledge used to support a transformation step is represented by ovals. Thick arrows represent a flow of data, while dashed arrows depict the relation '*is used in*' whenever a process needs additional knowledge.

First a text is parsed in order to determine its syntax tree (process 1). This syntax tree is transformed by (2) into a Discourse Representation Structure (DRS) by using *construction rules* (CRs) as proposed in the Discourse Representation Theory (DRT) [10]. A CR consists of a *triggering structure* and a set of *actions*. A triggering structure represents a branch of a syntax tree. If the syntax tree for the given text contains the triggering structure of a CR, this rule's set of actions is applied. Such actions comprise modifications of the syntax tree and/or additions of elements to the DRS. A DRS is called 'reduced' if the corresponding syntax tree has been completely erased while executing the actions of matched CRs. In a next transformation step (3), a logic program is derived from such a DRS based on *Transformation Rules* (*TRs*). A TR consists of a *pattern–DRS* and a set of actions. If the pattern–DRS of a TR matches part of a DRS, the TR's actions are carried out. These actions can comprise modifications of the existing DRS and/or additions of rules to the logic program. While performing these transformations, two types of knowledge have to be distinguished:

- *discourse-specific* knowledge refers to behavior of agents in the scene domain.
- *scheme-specific* knowledge refers to knowledge about how the domain specific statements in the text have to be converted into a logic representation of the behavior of agents.

These two kinds of knowledge are distinguished by means of a so-called *semantic dictionary*. The entries of this dictionary consist of a pattern of one or more DRS-predicates and a set of actions which is processed if the pattern is found in a DRS. These patterns allow to act with the *names* of conditions in contrast to the patterns of a transformation rule which contain structural information only. A condition called `decompose` thus will be treated in a different manner than a condition called `concatenation`. The semantic dictionary and the transformation rules allow to derive a logic program which corresponds to the situation graph described in the processed text.

(1) and (2) are automatically generated by the so–called DRS–Creator–Generator. This tool takes the textual definition of a grammar as well as of CRs and automatically builds a DRS–Creator according to these definitions. This approach facilitates us to focus on linguistic problems instead of programming details while generating a new DRS–Creator (e.g., for German texts) and to quickly extend an existing grammar. The only additional knowledge needed then for the DRS–creation is the *syntactic dictionary*, which yields syntactic information about words, e.g., part of speech, gender, etc. .

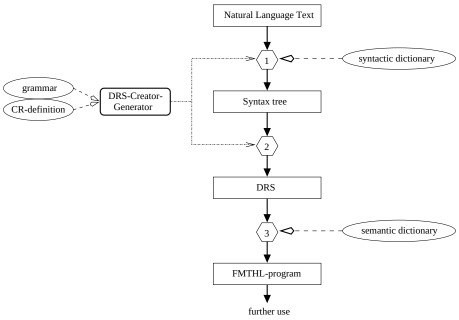


Figure 1. Transformation process from a natural language text input to an FMTHL logic representation.

4 EXPERIMENTAL RESULTS

4.1 Behavioral knowledge extraction

The first example concerns the processing of the text:

> *'crossing an intersection' is decomposed into a concatenation of 'driving to the intersection', 'driving across the intersection' and 'leaving the intersection'.*

This text describes the situation of an agent *crossing an intersection* to be temporally decomposed into a concatenation of three other situations (see, e.g., [8]). The analysis of this text – given a suitable grammar and also suitable CRs – results in the following DRS:

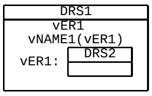
```
                              DRS1
 DR1    DR2    DR5    DR6    DR7    DR8    DR9
                     PDR1   PDR2
      ER44    ER46    ER48    ER50    ER52
          leaving_the_intersection(ER50)
          crossing_an_intersection(ER48)
       driving_across_the_intersection(ER46)
         driving_to_the_intersection(ER44)
                concatenation(DR2)
                 intersection(DR1)
                 CONJUNCTIVE(PDR2)
                 CONJUNCTIVE(PDR1)
                     agens(DR9)
                     agens(DR8)
                     agens(DR7)
                     agens(DR6)
                     agens(DR5)
                    of(DR2,PDR2)
               PDR1= ⊕ (ER46,ER50)
               PDR2= ⊕ (ER44,PDR1)

                  ER44:  | DRS2        |
                         | drive(DR5)  |
                         | to(DR1)     |
                  ER46:  | DRS3        |
                         | drive(DR6)  |
                         | across(DR1) |
                ER48:   | DRS4            |
                        | cross(DR7,DR1)  |
                ER50:   | DRS5            |
                        | leave(DR8,DR1)  |
             ER52:   | DRS6                   |
                     | into(DR2)              |
                     | decompose(DR9,ER48)    |
```

According to [10], a DRS always consists of a *universe* of referents and a set of conditions. The first three lines of the DRS above list the referents of the DRS, where `DR` stands for *discourse referents* (for individuals), `PDR` means *plural discourse referent* (sets of other referents) and `ER` always denotes referents for events (*event referent*). The rest of this DRS comprises the set of conditions. These conditions can express features of referents (`agens(DR9)`), relations between referents (`leave(DR8,DR1)`), but also more complex conditions like the definition of PDRs or ER–definitions comprising the name of the ER and a DRS defining the part of the discourse connected to that ER.

The transformation of this DRS into a logic program representing an SGT starts with the (repeated) application of a particular TR identifying explicitly mentioned situations. This TR has the form:

TR_SITMATCH1

PatternDRS:



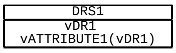
Actions:

```
collectDRKnowledge(DRS2);
addNewAttribute(SIT(vER1));
addNewAttribute(EVAL(vER1));
d = DRS2.copy();
e = new DRS();
e.addNewName(vNAME1(vER1));
addNewPredicate(SITSTATE(vER1,d));
addNewPredciate(SITNAME(vER1,e));
addNewPredciate(PREDICT(vER1,vER1));
removePredicate(vNAME(vER1));
```

This TR searches in the DRS for an ER and a name–condition for that ER. Once this pattern has been found, the actions listed in the actions–block of the TR are executed, adding several new conditions to the DRS (for example an attribute–condition `SIT(vER1)` which marks the ER as representing a situation). Then those conditions which led to the application of the TR (namely the name–condition and the ER itself) are deleted from the DRS. Other TRs analogously transform the DRS–conditions concerning the collection of three situations (expressed as PDRs) and shift remaining conditions from inside of ER–definitions up to the top–level DRS. This for example transforms the conditions `decompose(DR9,ER48)` and `into(DR2)` of the definition of `ER52` in the DRS into conditions of the form `decompose(ER52,DR9,ER48)` and `into(ER52,DR2)` of the DRS. The next transformation steps concern the distinction of discourse– and scheme–specific knowledge. With TRs like

TR_DICTPRED3

PatternDRS:



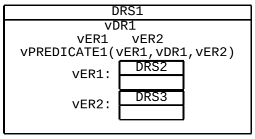
Actions:

```
callActions(vATTRIBUTE1(vDR1));
removePredicate(vATTRIBUTE1(vDR1));
```

and

TR_DICTPRED4

PatternDRS:



Actions:

```
callActions(vPREDICATE1(vER1,vDR1,vER2));
removePredicate(vPREDICATE1(vER1,vDR1,vER2));
```

(among others using slightly different patterns due to different numbers and types of arguments of conditions) this distinction can be achieved. As can be seen above, each of these TRs comprises a pattern with one condition. This is an attribute–condition with one DR as the argument in the first case and a predicate–condition with three arguments, one ER, one DR and another ER, in the second case. The actions–block of these TRs always consists of the method `callActions(<condition>)` and the deletion of the one pattern–condition. The method `callActions` causes the transformation process to query the semantic dictionary for the *meaning* of the DRS–condition which was matched by the TR. Within the semantic dictionary these two entries are important for the example described here:

```
...
ATTR:concatenation(vDR 1) {
   PRED:of(vDR 1, vPDR 1) {
     [ replaceOccurrencies(vDR 1, vPDR 1)]
   }}

PRED:decompose(vER 1, vDR 1, vER 2) {
   PRED:into(vER 1, vPDR 1) {
     ATTR:SITLIST(vPDR 1) {
       [ concatenateSituations(vPDR 1)
         specializeSitWithGraph(vER 1, vER 2, vPDR 1)]
    }}}
...
```

Each entry in this dictionary starts with one DRS–condition (the *key–entry*), where `ATTR` describes an attribute–condition and `PRED` a predicate–condition. This key–entry then can be followed by one or more sub–entries, each constraining the meaning of the key–entry by additional conditions, which themselves again can comprise sub–entries. The leaves of this *entry–tree* are given by a list of actions which at last define the meaning of all the conditions together collected on a path from the key–entry to that leaf. The reading of the first entry given above could thus be circumscribed as: *the meaning of* `concatenation(vDR 1)` *– given the additional condition* `of(vDR 1, vPDR 1)` *– is the execution of the action* `replaceOccurrencies(vDR 1, vPDR 1)`. The occurrence of the condition `concatenation(DR2)` in the DRS shown above thus will lead to the application of TR_DICTPRED3 (see above). This invokes a query to the semantic dictionary and – because a condition `of (DR2,PDR2)` can be found in the actual DRS, too – this will lead to the execution of the method mentioned above.

The whole transformation of the example–DRS – which could only be outlined here due to space limitations – comprises 43 TR–applications using 14 different TRs. The remaining TRs not treated here serve different other aspects of the transformation. The last TRs applied are those which actually write the desired result as a logic program. The text given at the beginning of this section thus results in:

```
-i : +i !  (state_ER44(DR7,DR1) :-
           (drive(DR7),to(DR1),agens(DR7),intersection(DR1))).
-i : +i !  (acti_ER44(DR7,DR1) :- (true)).
-i : +i !  (state_ER46(DR7,DR1) :-
           (drive(DR7),across(DR1),agens(DR7),intersection(DR1))).
-i : +i !  (acti_ER46(DR7,DR1) :- (true)).
-i : +i !  (state_ER48(DR7,DR1) :-
           (cross(DR7,DR1),agens(DR7),intersection(DR1))).
-i : +i !  (acti_ER48(DR7,DR1) :- (true)).
-i : +i !  (state_ER50(DR7,DR1) :-
           (leave(DR7,DR1),agens(DR7),intersection(DR1))).
-i : +i !  (acti_ER50(DR7,DR1) :- (true)).
-i : +i !  (actn_ER44(DR7,DR1) :- note(driving_to_the_intersection(DR7,DR1))).
-i : +i !  (actn_ER46(DR7,DR1) :- note(driving_across_the_intersection(DR7,DR1))).
-i : +i !  (actn_ER48(DR7,DR1) :- note(crossing_an_intersection(DR7,DR1))).
-i : +i !  (actn_ER50(DR7,DR1) :- note(leaving_the_intersection(DR7,DR1))).
-i : +i !  (pred_ER44(DR7,DR1) :-
           (+1 : +1 !  (fstate_ER46(DR7,DR1),!,eval_ER46(DR7,DR1)))).
-i : +i !  (pred_ER46(DR7,DR1) :-
           (+1 : +1 !  (fstate_ER50(DR7,DR1),!,eval_ER50(DR7,DR1)))).
-i : +i !  (pred_ER44(DR7,DR1) :-
           (+1 : +1 !  (fstate_ER44(DR7,DR1),!,eval_ER44(DR7,DR1)))).
-i : +i !  (pred_ER46(DR7,DR1) :-
           (+1 : +1 !  (fstate_ER46(DR7,DR1),!,eval_ER46(DR7,DR1)))).
-i : +i !  (pred_ER48(DR7,DR1) :-
           (+1 : +1 !  (fstate_ER48(DR7,DR1),!,eval_ER48(DR7,DR1)))).
-i : +i !  (pred_ER50(DR7,DR1) :-
           (+1 : +1 !  (fstate_ER50(DR7,DR1),!,eval_ER50(DR7,DR1)))).
-i : +i !  (eval_ER48(DR7,DR1) :- (state_ER44(DR7,DR1),!,eval_ER44(DR7,DR1))).
-i : +i !  (fstate_ER44(DR7,DR1) :- (fstate_ER48(DR7,DR1),!,state_ER44(DR7,DR1))).
-i : +i !  (facti_ER44(DR7,DR1) :- (facti_ER48(DR7,DR1),acti_ER44(DR7,DR1))).
-i : +i !  (fact_ER44(DR7,DR1) :-
           (facti_ER48(DR7,DR1),actn_ER44(DR7,DR1),acti_ER44(DR7,DR1))).
-i : +i !  (pred_ER44(DR7,DR1) :- (pred_ER48(DR7,DR1))).
-i : +i !  (eval_ER48(DR7,DR1) :- (state_ER46(DR7,DR1),!,eval_ER46(DR7,DR1))).
-i : +i !  (fstate_ER46(DR7,DR1) :- (fstate_ER48(DR7,DR1),!,state_ER46(DR7,DR1))).
-i : +i !  (facti_ER46(DR7,DR1) :- (facti_ER48(DR7,DR1),acti_ER46(DR7,DR1))).
-i : +i !  (fact_ER46(DR7,DR1) :-
           (facti_ER48(DR7,DR1),actn_ER46(DR7,DR1),acti_ER46(DR7,DR1))).
-i : +i !  (pred_ER46(DR7,DR1) :- (pred_ER48(DR7,DR1))).
-i : +i !  (eval_ER48(DR7,DR1) :- (state_ER50(DR7,DR1),!,eval_ER50(DR7,DR1))).
-i : +i !  (fstate_ER50(DR7,DR1) :- (fstate_ER48(DR7,DR1),!,state_ER50(DR7,DR1))).
-i : +i !  (facti_ER50(DR7,DR1) :- (facti_ER48(DR7,DR1),acti_ER50(DR7,DR1))).
-i : +i !  (fact_ER50(DR7,DR1) :-
           (facti_ER48(DR7,DR1),actn_ER50(DR7,DR1),acti_ER50(DR7,DR1))).
-i : +i !  (pred_ER50(DR7,DR1) :- (pred_ER48(DR7,DR1))).
-i : +i !  (eval_ER44(DR7,DR1) :- (fact_ER44(DR7,DR1),!,pred_ER44(DR7,DR1))).
-i : +i !  (eval_ER46(DR7,DR1) :- (fact_ER46(DR7,DR1),!,pred_ER46(DR7,DR1))).
-i : +i !  (eval_ER48(DR7,DR1) :- (fact_ER48(DR7,DR1),!,pred_ER48(DR7,DR1))).
-i : +i !  (eval_ER50(DR7,DR1) :- (fact_ER50(DR7,DR1),!,pred_ER50(DR7,DR1))).
-i : +i !  (pred_ER48(DR7,DR1) :- (fail)).
-i : +i !  (fstate_ER48(DR7,DR1) :- (state_ER48(DR7,DR1))).
-i : +i !  (facti_ER48(DR7,DR1) :- (acti_ER48(DR7,DR1))).
-i : +i !  (fact_ER48(DR7,DR1) :- (acti_ER48(DR7,DR1),actn_ER48(DR7,DR1))).
-i : +i !  (run_traversal_ER48(DR7,DR1) :-
           (state_ER48(DR7,DR1),!,eval_ER48(DR7,DR1))).
-i : +i !  (traversal :- (run_traversal_ER48(DR7,DR1))).
```

The FMTHL–programs representing SGTs are not easy to understand and – more important – not at all easy to inspect in order to find errors in the actual transformation–process (a detailed description of the way the above program represents an SGT is given in [8]). In order to simplify the inspection of SGTs built from natural language texts, we started to implement a graphical user-interface to SGTs using the DiaGen-tool (see [12]). This tool takes a definition of (almost) arbitrary diagram-types given as a hypergraph grammar and generates an initial editor in form of Java-classes, which then can be adapted to further requirements concerning the user interface.

One problem using this editor generator arose with the layout to be performed on diagrams representing SGTs: each situation graph of an SGT comprises directed, probably even cyclic, and not neccessarily connected graphs consisting of situation schemes and prediction edges. As no perfect layout algorithm for such a kind of graph was known to us, we decided to use the so-called *force layout* mechanism to perform a layout on the generated SGT-diagrams. A first result of the diagram showing the SGT represented by the FMTHL–program above can be seen in Figure 2. The diagram consists of two situation graphs: the upper one comprising the most general situation scheme (*crossing an intersection*). This situation is decomposed into the lower situation graph consisting of three consecutive situation schemes, one for each concept mentioned in the input text.

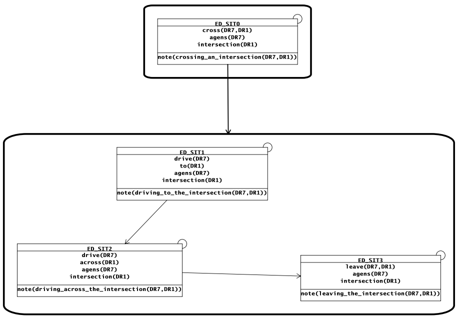


Figure 2. Diagram showing the situation graph tree (SGT) generated from the text example shown at the beginning of section 4.1.

4.2 Visualization

Up to now the system can visualize texts describing maneouvers of single vehicles at intersections. The text

A car came from Kriegsstrasse. It turned left at the intersection.

describes a scene at an intersection. A car approaches the intersection from Kriegsstrasse, turns left and leaves. The goal of processing this text is to derive a conceptual description of the scene, which then can be visualized.

In a first step the following DRS is computed from the given text:

```
                DRS1
      DR1    DR2    DR3
           ER1   ER2
        Kriegsstrasse(DR1)
        intersection(DR3)
             car(DR2)

                   DRS2
        ER1:    come(DR2)
                from(DR1)
                   DRS3
   ER2:         at(DR3)
             turn_left(DR2)
```

A description of a traffic scene contains no scheme-specific knowledge. The semantic dictionary thus needs not to be used and only those transformation rules concerning the discourse-specific knowledge are applied. The transformations of the DRS result in the FMTHL-program:

```
-i : +i !  Kriegsstrasse(dr1).
-i : +i !  car(dr2).
-i : +i !  intersection(dr3).
-i : +i !  from(er1,dr1).
-i : +i !  come(er1,dr2).
-i : +i !  turn_left(er2,dr2).
-i : +i !  at(er2,dr3).
```

This logic program depicts the activities of *coming* and *turning*. In [11] the authors describe a hierarchy of activities. Ordered by their temporal duration and aggregation from the smallest to the largest, these are *occurrence*, *action*, *episode* and *history*. Thus a history consists of one or more episodes which themselves comprise actions.

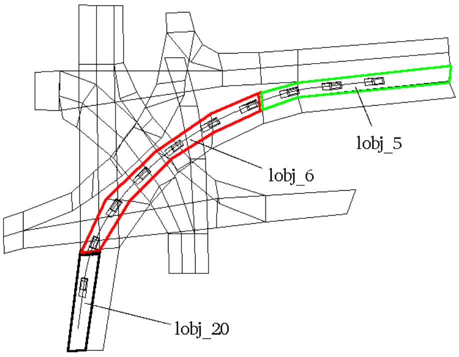


Figure 3. SIS generated from the given text.

Actions in turn are composed of occurrences. A history characterizes the most comprehensive form of describing an extended sequence of related activities. The story described in the text corresponds to a history. The aim is now to first identify the schematic history which is described in the text and to derive a sequence of occurrences from such a history in a next step, since only (elementary) occurrences can be visualized directly. The history is represented in FMTHL by means of the predicate: `history(HistoryName × Agent × Road)`. A set of rules is provided for derivation of the validity of the history predicate from FMTHL-programs as for example the one given above. For this example the validity of the history predicate

```
history(turningLeft,dr2,Kriegsstrasse)
```

is derived. In this case the history parameter `turningLeft` represents a left turning maneouver. Now the history predicate allows to derive a sequence of occurrence predicates using default values. It has been demonstrated already how a synthetic image sequence can be generated from such a sequence – see [15]. In order to achieve this, two primitive concepts `on` and `speed` are used which describe the position and the velocity of a vehicle. The predicate `on(Agent × Lane_object)` constitutes that an agent is positioned *on* a specific lane object. It thus has been assumed a-priori that a road consists of several lanes each of which may comprise several lane objects. A geometric model of an intersection and of its lane objects – which are of interest for this example – can be seen in Figure 3. The predicate `speed(Agent × Speedvalue)` describes the velocity of an agent whereas `Speedvalue` refers to a symbolic (not a numeric) description of a speed value. The history predicate derived above allows to derive the following program describing a set of occurrence predicates and their temporal relations:

```
1   : 500 ! agent(dr2).
1   : 500 ! model(dr2,car).
1   : 200 ! on(dr2,lobj_5).
201 : 300 ! on(dr2,lobj_6).
301 : 500 ! on(dr2,lobj_20).
1   : 200 ! speed(dr2,normal).
201 : 300 ! speed(dr2,normal).
301 : 500 ! speed(dr2,normal).
```

Here, an instant of time represents a 20 msec interval (half-period of a video frame). Thus the entire scene lasts 10 seconds. The set of occurrence predicates is interpreted in the following way. The agent of this scene is the individual called `dr2` whose geometric model corresponds to a car. As can be seen from the set of occurrences, the initial lane is `lobj_5`. After 200 time instants the agent changes to the lane called `lobj_6` and finally drives on lane `lobj_20` until the final time point 500. For every point of time in this interval the agent drives at regular speed - as can be noticed by inspecting the `speed(dr2,normal)` predicates.

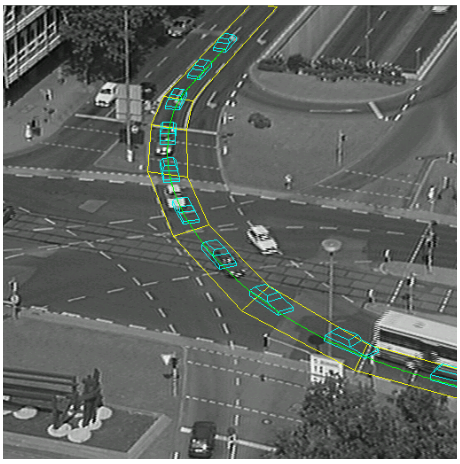


Figure 4. SIS projected into a picture of the corresponding intersection. The car model has been drawn every second.

The effect of the above set of rules is visualized in Figures 3 and 4. Figure 3 shows a top view and Figure 4 a projection of the lane model into an image frame from the intersection sequence which has been downloaded from `http://i21www.ira.uka.de/image_sequences/`. The car model has been drawn every 50 points of time in both figures. Only the left turning lane – the lane of interest – has been projected into an image of the intersection in Figure 4. The stationary background image of the real intersection in Figure 4 has been added for illustration purposes only. It should just demonstrate that the geometric model used to calculate the SIS is close to real-life data. This close relation might allow to compare the SIS with an OIS in the future.

5 CONCLUSION

In this contribution two possible applications of natural language processing in the context of a cognitive vision system have been proposed: the automatic acquisition of behavioral knowledge and the generation of a synthetic image sequence from a text. Non–trivial texts were processed in order to study these tasks. The same approach creates a logic representation of texts for both applications using the same processes and intermediate representations. This is achieved by enabling the system to distinguish domain- and scheme-specific knowledge. We thus use natural language texts in order to introduce schematic knowledge into our system. This schematic knowledge serves subsequently to instantiate the description of a specific development given either in the form of an input video sequence or of an input NL text.

6 FUTURE WORK

At the present point of implementation, our system uses different grammars, different CRs and different TRs for each of the two kinds of text exemplified in Section 4. Further investigations are required on how the grammars, CRs and TRs can be unified and extended such that more complex NL–descriptions of generic knowledge and traffic scenes can be processed. Another problem concerns the question of how SGTs can be used to analyse texts describing the actual behavior of an agent. In the past, these behavior schemes were used to *recognize* the behavior of an agent. Simple concepts derived from image sequence evaluation are being associated with complex concepts describing situations. Using SGTs to analyse texts describing what is happening in a recorded scene would need to *invert* that process and thus associate a sequence of situations derived from the text with simple concepts. At the present point of implementation, this inverted process is provided by additional a-priori hand-coded logic-rules. Once this is accomplished, the resulting simple concepts again have to be transformed into geometric trajectory data describing the movement of an agent explicitly. This then corresponds to the inversion of the derivation of simple concepts from geometric data as described in [11].

REFERENCES

[1] A. Chella, M. Frixione and S. Gaglio: *Understanding dynamic scenes.* Artificial Intelligence **123**:1–2 (2001) 89-132.

[2] B. Coyne and R. Sproat: *WordsEye: An Automatic Text-to-Scene Conversion System.* In: Proc. SIGGRAPH 2001, Los Angeles, CA, USA, pp. 487–496.

[3] A. Egges, A. Nijholt and P. Nugues: *Generating a 3D Simulation of a Car Accident from a Formal Description: the CarSim System.* In: V. Giagourta and M. G. Strintzis (Eds.), Proc. of the Int. Conf. on Augmented, Virtual Environments and Three-Dimensional Imaging (ICAV3D), Mykonos, Greece, May 2001, pp. 220–223.

[4] J. Fernyhough, A. G. Cohn and D. C. Hogg: *Constructing qualitative event models automatically from video input.* Image and Vision Computing **18**:2 (2000) 81-103.

[5] P. Gärdenfors: *Conceptual Spaces: The Geometry of Thought.* MIT Press, Cambridge, MA, 2000.

[6] R. Gerber and H.-H. Nagel: *(Mis?)-Using DRT for Generation of Natural Language Text from Image Sequences.* In: H. Burkhardt and B. Neumann (eds.): Proc. of the 5th European Conf. on Computer Vision (ECCV'98). 2-6 June 1998, Freiburg, Germany; (LNCS) 1407, Springer-Verlag, Berlin, Heidelberg, New York, 1998, pp. 255-270.

[7] C. Godéreaux, P.O. El Guedj, F. Revolta and P. Nugues: *Ulysse: An Interactive, Spoken Dialogue Interface to Navigate in Virtual Worlds.* In J. Vince and R. Earnshaw (eds), Virtual Worlds on the Internet, Chapter 4, IEEE Computer Society Press, Los Alamitos, 1999, pp. 53-70.

[8] M. Haag and H.-H. Nagel: *Incremental Recognition of Traffic Situations from Video Image Sequences.* Image and Vision Computing **18**:2 (2000) 137-153.

[9] R. J. Howarth and H. Buxton: *Conceptual descriptions from monitoring and watching image sequences.* Image and Vision Comupting **18**:2 (2000) 105-135.

[10] H. Kamp and U. Reyle: *From Discourse to Logic.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1993.

[11] H. Kollnig and H.–H. Nagel: *Ermittlung von begrifflichen Beschreibungen von Geschehen in Straßenverkehrsszenen mit Hilfe unscharfer Mengen.* Informatik Forschung und Entwicklung **8** (1993) 186–196 (in German).

[12] M. Minas: *Spezifikation und Generierung graphischer Diagrammeditoren.* Habilitationsschrift, Friedrich–Alexander–Universität Erlangen–Nürnberg, 2001. Berichte aus der Informatik; Shaker–Verlag Aachen 2001 (in German).

[13] A. Mukerjee, K. Gupta, S. Nautiyal, M.P. Singh and N. Mishra: *Conceptual description of visual scenes from linguistic models.* Image and Vision Computing **18**:2 (2000) 173-187.

[14] H.-H. Nagel: *Towards a Cognitive Vision System.* `http://kogs.iaks.uni-karlsruhe.de/CogViSys/` September 2001.

[15] H.-H. Nagel, M. Haag, V. Jeyakumar and A. Mukerjee: *Visualisation of Conceptual Descriptions Derived from Image Sequences.* Mustererkennung 1999, 21. DAGM-Symposium, Bonn, 15-17. September 1999, Springer-Verlag, Berlin, Heidelberg, u.a., pp. 364-371.

[16] R. Power, D. Scott and R. Evans: *What You See Is What You Meant: direct knowledge editing with natural language feedback.* In: H. Prade (ed): Proc. 13th ECAI, Brighton, UK, 1998, pp. 677–681.

[17] K. H. Schäfer: *Unscharfe zeitlogische Modellierung von Situationen und Handlungen in der Bildfolgenauswertung und Robotik.* Dissertation, Fakultät für Informatik der Universität Karlsruhe (TH), Karlsruhe, Juli 1996; erschienen in der Reihe Dissertationen zur Künstlichen Intelligenz (DISKI) **135**; infix–Verlag Sankt Augustin 1996 (in German).

[18] Y. Shinyama, T. Tokunaga and H. Tanaka : *Kairai - Software Robots Understanding Natural Language.* Third Int. Workshop on Human-Computer Conversation, Bellagio, Italy, 3-5 July 2000.