Representation of Behavioral Knowledge for Planning and Plan–Recognition in a Cognitive Vision System

Michael Arens and Hans–Hellmut Nagel

Institut für Algorithmen und Kognitive Systeme,
Fakultät für Informatik der Universität Karlsruhe (TH),
Postfach 6980, D–76128 Karlsruhe, Germany
{arens, nagel}@ira.uka.de

Abstract. The algorithmic generation of textual descriptions of image sequences requires conceptual knowledge. In our case, a stationary camera recorded image sequences of road traffic scenes. The necessary conceptual knowledge has been provided in the form of a so-called *Situation Graph Tree (SGT)*. Other endeavors such as the generation of a synthetic image sequence from a textual description or the transformation of machine vision results for use in a driver assistance system could profit from the exploitation of the same conceptual knowledge, but more in a planning (*pre*-scriptive) rather than a *de*-scriptive context.
A recently discussed planning formalism, *Hierarchical Task Networks (HTNs)*, exhibits a number of formal similarities with SGTs. These suggest to investigate whether and to which extent SGTs may be re-cast as HTNs in order to re-use the conceptual knowledge about the behavior of vehicles in road traffic scenes for planning purposes.

1 Introduction

Road traffic offers challenging examples for an algorithmic approach towards 'understanding' image sequences of such scenes. We are concerned here with systems which rely on an *explicated* knowledge base which comprises knowledge about the geometry of the depicted scene and about admissible trajectories of moving vehicles. In addition, conceptual knowledge is required in order to transform quantitative results related to geometric properties into textual descriptions of vehicle maneuvers and their context – see, e. g., [7, 9]. In these cases, the required knowledge has been provided in the form of a so-called *Situation Graph Tree (SGT)*.

An attempt to invert such a processs, i. e. to generate a synthetic image sequence from a textual description related to the same discourse domain of road traffic ([17]), access to the same or similar *conceptual* knowledge turns out to be desirable in order to infer details which have not been incorporated into the text because they were assumed to be available to a human in the form of commonsense knowledge. Driver assistance systems based on video cameras installed in vehicles and linked to machine vision systems constitute another example where conceptual knowledge about road traffic scenes will be needed, in particular if such assistance systems have to cope with the intricacies of innercity traffic. Preliminary results in this direction ([5, 6]) suggest the exploitation of knowledge originally provided for the interpretation of image sequences. In both cases, the conceptual knowledge may be needed for planning purposes. This consideration motivates our investigation to convert knowledge provided in the form of an SGT into another form more suitable for planning. Due to a number of formal similarities, *Hierarchical Task Networks (HTNs)* offer themselves as a good starting point for such an investigation.

In the sequel, we first sketch the formulation of planning tasks based on HTNs and then outline SGTs. Based on this exposition, we discuss our experience with a more detailed comparison between these two formalisms. As will be seen, a number of aspects which become apparent only upon closer inspection requires careful attention: the context in which these two formalisms have been developed influenced their properties.

2 Hierarchical Task Networks

Planning with Hierarchical Task–Networks (HTNs) [21] (see also [20]) is similar to STRIPS–like planning formalisms [4]. Each world–state (the state of the discourse in which a plan is searched) is represented as a set of atoms true in that state. An action modifying the world–state in STRIPS normally corresponds to a triple of sets of atoms: the preconditions, which have to be true whenever the action should be executed, the add–list of atoms, listing all atoms which will become true after the action has been executed, and the delete–list holding all atoms negated (or deleted) from the world–state by executing the action. A planning–problem in STRIPS can then be stated as a tuple $\mathbf{P} = \langle \mathcal{I}, \mathcal{G}, A \rangle$, where $\mathcal{I}$ denotes the initial world–state, $\mathcal{G}$ describes the desired world–state and A is the set of possible actions. A solution to such a planning problem is given by a sequence of actions, which – starting with the initial world–state – produce the desired world–state. Each atom contained in the desired world–state is called a *goal*.

In HTN–Planning, the goals of STRIPS are replaced by *goal–tasks*. In addition to these tasks, two other types of tasks can occur: *primitive tasks* and *compound tasks* [1]. While primitive tasks can be accomplished by simply executing an associated action, compound–tasks are mapped to one or more possible *task networks*. These networks are directed graphs comprising further tasks as nodes and successor relations between these tasks as edges, which naturally define the order in which tasks should be achieved. In addition to that, the edges can be attributed by further constraints concerning the preconditions of tasks and the assignment of variables comprised in task–descriptions. If a task network consists of only primitive tasks, it is called *primitive task network*.

Planning with HTNs can thus be formulated as follows [2]: The planning domain is described as a pair $\mathcal{D} = \langle A, M \rangle$, where A again denotes the set of possible actions and M is a set of mappings from compound tasks onto task networks. A planning problem can then be formulated as a tuple $\mathbf{P} = \langle d, \mathcal{I}, \mathcal{D} \rangle$ where d is an initial task network, $\mathcal{I}$ describes the initial world–state and $\mathcal{D}$ is a planning domain. Note that – in contrast to STRIPS – no desired world–state ($\mathcal{G}$) is given in the formulation of the planning problem. A solution to the problem $\mathbf{P}$ can thus not simply be a sequence of actions achieving such a world–state. Instead, each goal which has to be achieved by the plan is incorporated into the initial task network d as a goal task. The solution to $\mathbf{P}$ in HTN planning is a task network itself: given the initial task network d of $\mathbf{P}$, an expansion of that network is searched such that the resulting network is primitive and all constraints incorporated into the network are satisfied. If d is a primitive task network, then d itself is a solution to the problem $\mathbf{P}$ if all constraints in d are satisfied. Otherwise d contains at least one compound task c. For this task a mapping $m \in M$ is searched which maps (expands) the task c onto a further task network d_c. This means that one way to achieve the task c is to accomplish all the tasks in the network d_c. The result is a task network d' similar to d, but incorporating the new network d_c instead of the one task c. If the resulting network d' is primitive, a solution is found, otherwise another compound task can be expanded into a network and so on.

[3] have shown that HTN–Planning is strictly more expressive than STRIPS–style planning. [11] points out that – speaking about efficiency of planning formalisms – HTN–Planning profits from the fact that the user can *direct* the search for plans by defining the mapping from compound tasks to task networks and thus restrict the search space to those decompositions that lead to allowed or desired plans. While in STRIPS–style planners any sequence of actions leading from the initial state to the desired goal state is a valid plan, in HTN–planning only those sequences that can be derived by decomposing the initial task network into a primitive task network are supposed to be meaningful and feasible. By defining this mapping from compound tasks into task networks, the user not only directs the search for a problem solution, but also incorporates more knowledge about the planning domain into the planner.

2.1 Formal Syntax for HTNs

In the following formal definition of HTN–syntax we use the notation introduced by [13], where o_i denotes a primitive task and N_i stands for a non–primitive (or compound) task. The *mappings* from compound tasks onto task networks of [2] are called *reduction schemes* in [13]. Similar to [13] we will write $r_{i,j} = \langle \mathbf{C} \rangle$, which denotes that $r_{i,j}$ is the jth reduction scheme of the non–primitive task N_i, where $\mathbf{C}$ is a set of constraints describing a task network. These reduction schemes define the decomposition of non–primitive tasks into task networks: reduction schemes are defined as task networks using the 14 types of constraints shown in Fig. 1. After formulating a hierachical task network in terms of these 14 constraints, [13] define transformation schemes based on [12] which *encode* the defined network in form of propositional logic formulae. The solution to the

(1)	o_i:`<action_name>`	The primitive task symbol o_i is mapped onto the action name `<action_name>`.
(2)	N_j:`<task_name>`	The non–primitive task N_j is mapped onto the task name `<task_name>`.
(3)	$o_p \prec o_q$	The primitive task o_p temporally precedes the primitive task o_q.
(4)	$o_s \prec N_p$	The primitive task o_s temporally precedes the non–primitive task N_p.
(5)	$N_p \prec o_s$	The non–primitive task N_p temporally precedes the primitive task o_s.
(6)	$N_p \prec N_q$	The non–primitive task N_p temporally precedes the non–primitive task N_q.
(7)	$o_p \xrightarrow{f} o_q$	There exists a causal link between o_p and o_q such that f is an add effect of o_p and a precondition of o_q.
(8)	$o_p \xrightarrow{f} N_q$	There exists a causal link between o_p and N_q such that f is an add effect of o_p and a precondition of N_q.
(9)	$N_q \xrightarrow{f} o_p$	There exists a causal link between N_q and o_p such that f is an add effect of N_q and a precondition of o_p.
(10)	$N_p \xrightarrow{f} N_q$	There exists a causal link between N_p and N_q such that f is an add effect of N_p and a precondition of N_q.
(11)	$o_q \xrightarrow{f} ?$	The primitive task o_q has an add effect which can be used to supply a precondition of some other task not known a–priori.
(12)	$N_p \xrightarrow{f} ?$	The non–primitive task N_p has an add effect f which can be used to supply a precondition of some other task not known a–priori.
(13)	$? \xrightarrow{f} o_p$	The primitive task o_p has a precondition f which has to be supplied by some other task not known a–priori.
(14)	$? \xrightarrow{f} N_q$	The non–primitive task N_q has a precondition f which has to be supplied by some other task not known a–priori.

Fig. 1. The 14 constraints used by [13] to express task networks. (1) and (2) state a syntactical mapping. The remaining twelve constraint types result from the fact that two relations dominate the structure of a single task network: the two–digit temporal order relation (defined on two kinds of arguments, primitive and non–primitive tasks) leads to four constraint types ((3)–(6)). The also two–digit causal link relation is based on three different types of arguments: primitive and non–primitive tasks, and a *wild card* for not–specified tasks. As the relation $? \xrightarrow{f} ?$ would make no sense here, this leads to the remaining eight types of constraints ((7)–(14)).

initial planning problem is then found by the so–called *planning as satisfiability*–paradigm: Any model of the formulated formulae will correspond to a plan solving the planning problem.

Other approaches to HTN–planning use a similar formulation of the HTN–structure itself, but the planning on that structure is done by an algorithm which tentatively (and recursively) decomposes tasks and backtracks, if constraints are violated or no solution for the planning problem can be found with the actually decomposed task–network. Examples for this approach are SHOP (see [14]) and SHOP2 (see [15]), but also UMCP (see [1]).

3 Situation Graph Trees

As outlined in section 1, we use Situation Graph Trees (SGTs) for supplying behavioral knowledge to our vision system. In this formalism, the behavior of an agent is described in terms of situations an agent can be in. Transitions between these situations express a temporal change from one situation to another. The term *generically describable situation* [16] or *situation scheme* describes the situation of an agent schematically in two parts: a *state scheme* denotes the state of an agent and of his environment with logic predicates. If each of these *state atoms* is satisfied, the agent is *instantiating* this situation scheme. The other part of a situation scheme is called *action scheme.* This scheme denotes actions – again formulated in logic predicates – the agent is supposed or expected to execute if this situation scheme can be instantiated.

Due to the analysis of image sequences, a natural discretization of time is given by the temporal interval between two consecutive image frames. For each of these frames, a quantitative evaluation of the image is performed, which leads to a *quantitative description* of the depicted scene. The results of this evaluation are associated with concepts formulated in logic predicates, yielding a *conceptual description* for each point in time (each image frame) of the observed scene.

An observed agent in the scene should instantiate one situation described in an SGT for each point in time. The *expectation* of what situation the agent will instantiate at the next point in time can be expressed by so called *prediction edges.* A prediction edge always connects two situation schemes, meaning that if an agent has been recognized to instantiate the situation from which the edge starts, one probable[1] next situation for that agent could be the situation to which the edge points. Of course, an agent can persist in a single situation for more than one point of time. Thus a *self–prediction* – a prediction edge from a situation scheme to itself – has to be allowed in SGTs.

Situation schemes together with prediction edges build *situation graphs.* These graphs are directed – according to the definition of prediction edges – and can comprise cycles. Situation schemes inside such a graph can be marked as *start*

[1] Note that SGTs and the SGT–traversal described later in this section are *deterministic* formalisms, in contrast to *probabilistic* Bayesian–Networks (see [19]) utilized for example by [10] and [23]. Thus prediction edges deterministically define which (and in which order) situation schemes should be investigated.

situation and/or *end situation*. Each path from a start situation to an end situation defines a *sequence* of situations represented by the situation graph. To refine a single situation scheme, it has to be connected to one or more situation graphs by so called *specialization edges*. Imagine for example an image sequence of an innercity road intersection with multiple lanes for each direction. On the most abstract level of conceptual description it might be sufficient to describe an agent (some vehicle in the scene) to just cross the intersection (see Fig. 2). To instantiate such a situation scheme `cross_0`, an observed object (`Agent`) would only have to satisfy two predicates: `agent(Agent)`, stating that the actual object is indeed an agent of some action, and `traj_active(Agent)`, meaning that the actual object is the object of interest (read: the *trajectory* of that object is actually *active*). The only action atom of this situation scheme in Fig. 2 is to print a message whenever this scheme is instantiated (`note(cross(Agent))`). This description might then hold during the whole period of time this vehicle is depicted in the image sequence (and recognized by the vision system). For a more detailed description of what the observed agent is doing, one might divide the whole process of crossing an intersection into one situation in which the agent advances towards the intersection on a certain lane (`drive_to_intersection_1`), another one, in which the agent actually crosses the intersection area (`drive_on_intersection`), and a third one, which describes the agent as leaving the intersection on another lane (`leave_intersection_1`, see Fig. 2). These three situation schemes are temporally ordered in the way they are mentioned above, so they should be connected with prediction edges according to that order, marking the first one as a start– and the last one as an end situation. Each of these situation schemes might hold for an agent for more than one image frame, so they should also be connected to themselves by self–predictions. By connecting the more general situation scheme describing an agent as crossing an intersection with the specializing situation graph constructed above, one states that everytime an agent can be described as crossing an intersection, he might also be in one of the situations comprised in that graph. The situation scheme specialized in this way is called *parent situation scheme* of the specializing situation graph.

Of course, several more specializations of the situation scheme describing an agent as crossing an intersection are imaginable: the agent might turn left or right, he might follow or precede another vehicle, etc. . Multiple specializations of one situation scheme are possible in SGTs simply by connecting that parent scheme with several other situation graphs. Analogously, every situation scheme comprised in such a specializing graph might be specialized further. In the example above, while driving to the intersection, the agent might have to stop (`stop_in_front_of_intersection_2`), wait (`wait_in_front_of_intersection_2`), and start again (`start_in_front_of_intersection_2`), or he might proceed without stopping (`proceed_to_intersection_2`).

However – for reasons explained later in this section – it is not allowed to connect situation schemes and situation graphs in a way that cycles from a more general situation scheme via some more special situation schemes back to again more general situations exist in the SGT. For the same reasons one situation

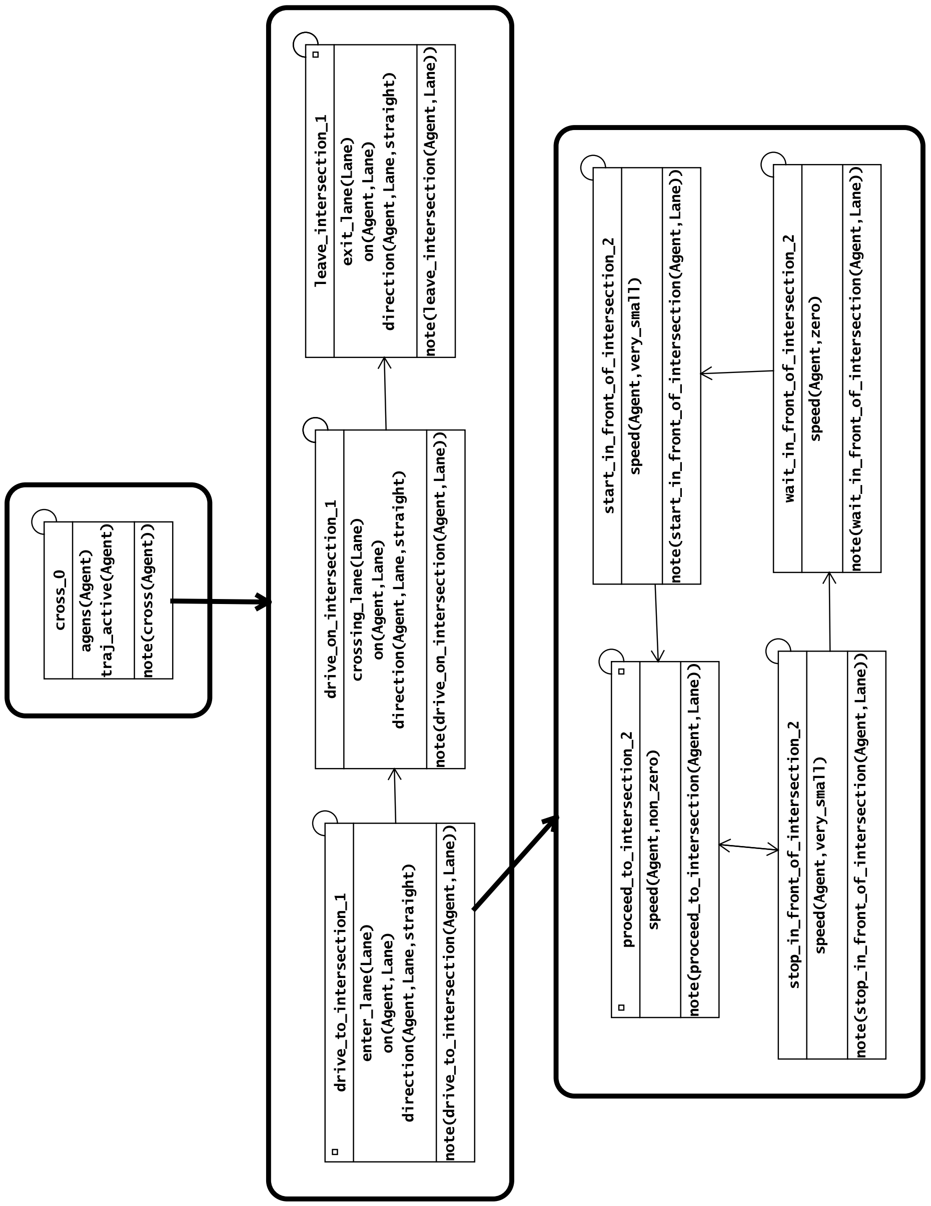
cross_0
agens(Agent)
traj_active(Agent)
note(cross(Agent))
drive_to_intersection_1
enter_lane(Lane)
on(Agent,Lane)
direction(Agent,Lane,straight)
note(drive_to_intersection(Agent,Lane))
drive_on_intersection_1
crossing_lane(Lane)
on(Agent,Lane)
direction(Agent,Lane,straight)
note(drive_on_intersection(Agent,Lane))
leave_intersection_1
exit_lane(Lane)
on(Agent,Lane)
direction(Agent,Lane,straight)
note(leave_intersection(Agent,Lane))
proceed_to_intersection_2
speed(Agent,non_zero)
note(proceed_to_intersection(Agent,Lane))
stop_in_front_of_intersection_2
speed(Agent,very_small)
note(stop_in_front_of_intersection(Agent,Lane))
wait_in_front_of_intersection_2
speed(Agent,zero)
note(wait_in_front_of_intersection(Agent,Lane))
start_in_front_of_intersection_2
speed(Agent,very_small)
note(start_in_front_of_intersection(Agent,Lane))

Fig. 2. Part of the situation graph tree describing the behavior of vehicles on an inner city road intersection taken from [18]. Rounded rectangles depict situation graphs. Situation schemes are shown as normal rectangles. Thin arrows stand for prediction edges, while bold arrows represent specialization edges. Circles in the right upper corner of situation schemes indicate a self–prediction of that scheme. Small rectangles to the left or to the right respectively of the name of situation schemes mark that scheme as a start– or end–situation. Detailed description in the text.

graph can only specialize exactly one or none situation scheme. Thus, situation schemes connected recursively to situation graphs build a tree–like structure, a situation graph tree.

The recognition of situations an agent is instantiating for each point in time could obviously be done by simply testing each and every situation scheme comprised in the SGT. Fortunately, this time–consuming approach can be significantly sped up by utilizing the knowledge encoded in form of prediction edges and specialization edges. The recognition of situations is thus performed by a so called *graph traversal*: The recognition of situations – and thus the traversal of the SGT – is started in the root situation graph[2]. Here, a start situation is searched for which each state atom is satisfied for the actual inspected point of time. If no such situation scheme can be found, the traversal fails. Otherwise the observed agent is instantiating that situation. If this situation scheme is specialized further by any situation graph, these graphs are again searched for start situation schemes with state atoms also satisfied in the actual point of time. If such a situation scheme can be found, it is instantiated by the agent. In this way the most special situation scheme that can be instantiated by the agent is searched. For the next point in time, only those situation schemes are investigated, which are connected by prediction edges to the situation scheme instantiated last. This also means that only situation schemes are investigated, which are members of the same graph. Thus, a prediction for a situation scheme on the same level of detail is searched. If none such scheme can be instantiated, two cases have to be distinguished: if the situation scheme last instantiated is an end situation, the failed prediction on this level of detail is accepted and the traversal is continued in the parent situation[3] of the actual graph. Otherwise (i.e., no end–situation was reached in the actual graph), the traversal fails completely. Thus, for each point of time, an agent is instantiating situation schemes on several levels of detail, each on a path from a scheme in the root graph to the most special scheme.

Concerning the action schemes comprised in these situation schemes, again two cases have to be distinguished. Each action atom in an action scheme can be marked as *incremental* or *non–incremental*. Incremental actions of a situation scheme are executed whenever this situation scheme lies on a path of schemes currently instantiated by the agent. Non–incremental actions of a situation scheme are only executed whenever this situation scheme is the most special scheme on a path of schemes an agent is instantiating.

[2] Here we suppose that an SGT comprises exactly one root situation graph. If a constructed SGT comprises more than one situation graph without a parent situation scheme, one single root graph can be introduced consisting of only one abstract situation scheme. This situation scheme can then be specialized by all situation graphs previously without a parent situation scheme.

[3] According to the definition of allowed specialization edges, the parent situation scheme of a situation graph is always unique for all graphs except the root graph of the SGT. In case of a failing prediction inside the root graph, the traversal fails completely, because no prediction is feasible inside the graph and no more general description of the behaviour of an agent exists.

3.1 Formal Syntax for SGTs

For a formal declaration of SGTs, [22] developed the declarative language `SIT++`. In this language, all aspects of SGTs described above can be expressed. The traversal of such an SGT is done by executing a logic–program obtained by an automatic translation of the `SIT++`–formulated SGT into rules of a *fuzzy metric temporal Horn logic* (FMTHL), also developed by [22]. By translating the SGT–traversal into a set of *fuzzy* logic rules, it is possible to operate with fuzzy truth values, which are essential to cope with the *uncertainty* arising in image evaluation due to sensor noise, but also with the inherent *vagueness* of natural language concepts. By translating the SGT–traversal into a set of *metric temporal* logic rules, it is possible to explicitly reason about time and thus for example demand minimum and maximum durations for the instantiation of particular situation schemes.

4 Comparison of HTNs and SGTs

Both – HTNs and SGTs – describe states and transitions between such states for a certain discourse in a hierarchical manner. In HTN–planning – as in every other planning task – a sequence of atomic actions is searched which solves a given problem. Thus, the structure of HTNs is action–oriented. In contrast, the graph traversal on SGTs used in our vision system *describes* situations and sequences of situations an observed agent is instantiating. Therefore the SGT–structure is state–oriented. It appears natural, though, to identify the state–atoms of a situation scheme in SGTs with preconditions of a task in HTNs (compare Fig. 1, (13) and (14)), as in both formalisms these atoms (state atoms and preconditions, respectively) have to be satisfied in order to execute the associated action(s). The action atoms of a situation scheme should then be identified with actions in HTNs. Because situation schemes can comprise more than one action atom, there cannot exist a one–to–one correspondence between situation schemes and (primitive) tasks. In case of more than one action atom inside a situation scheme, this scheme can only be identified with a non–primitive task (compare Fig. 1, (2)), which then should be decomposed into a sequence of primitive tasks mapped onto actions (compare Fig. 1, (1)) each corresponding to one action atom of the initial situation scheme.

The add– and delete–effects of an action are explicitly stated in HTNs as causal links between tasks (compare Fig. 1, (7)–(12)). In SGTs, only the association of states and corresponding actions are explicitly incorporated into a situation scheme. The effects of a certain action are not modelled inside the SGT, but are obtained from sensors, namely the underlying image sequence evaluation system[4]. Thus to facilitate planning with SGTs, either the add– and

[4] The need for a representation of *fuzzy* measures as well as for an explicite representation of time and a metric on time is a direct consequence of the connection of symbolic knowledge inside SGTs and the sensor evaluation outside the SGT.

delete–effects of actions have to be modelled inside the SGT or the sensor input has to be simulated[5].

The prediction edges of SGTs can be compared to the temporal relations included in the definition of task networks of HTNs (compare Fig. 1 (3)–(6)), though they cannot be totally identified: the temporal order–relation on tasks is (as an order–relation) *transitive*. The prediction edges of situation graphs define a relation on situation schemes which is not necessarily transitive (and normally is not transitive at all).

HTNs in the presented formalism do not naturally support loops inside a task network. Especially the analogy to self–predictions of situation schemes in SGTs can only be formulated in the HTN–syntax by introducing an abstract non–primitive task N_i, which then is decomposable into (at least) two other task networks: one (reduction scheme $r_{i,0}$) consisting of the final non–primitive task ($N_{i'}$) and another one ($r_{i,1}$), which consists of that final task ($N_{i'}$) and the initial task N_i again (additionally a temporal order between these tasks should be asserted, e.g., by the statement $N_{i'} \prec N_i$). In this way, an iterative execution of the task $N_{i'}$ is expressed by means of recursive reduction of non–primitive tasks, where $r_{i,0}$ represents the recursion end (compare, e.g., [11]).

The reduction schemes in the HTN–formalism correspond to the specialization edges leading from general situation schemes to situation graphs in SGTs, though there exists a difference in the utilization of these hierarchy–generating elements in both structures: in SGTs, all specialization edges and the situation graphs they point at are included in the SGT right from the start. This *entire* SGT is traversed then in order to find situation schemes which an observed agent instantiates. HTN–planning in contrast is started with one single initial task network (describing the planning problem). This task network is iteratively expanded to a *hierarchical* task network by decomposing non–primitive tasks. This means that one single SGT incorporates the complete knowledge about the (admissable) behaviour of agents in an observed discourse, whereas one single HTN only represents a single plan in the discourse for which a plan is searched. Thus, an SGT should not be compared to a single HTN, but rather to all HTNs which can be build with one set of primitive tasks, non–primitive tasks, and reduction schemes.

4.1 Expressing SGT–knowledge in the HTN–formalism

In the following, we assume that effects of actions of an SGT are obtained from a simulation as described in [8]. The simulation used there was controlled by

[5] This approach of sensor input simulation has already been applied by [8]: In case of an occlusion in the observed scene, sometimes not enough visual information can be obtained by the image evaluation system to properly actualize the estimated (quantitative) state of an observed agent. In such a case, [8] replaced the actualization of the state of the observed agent by a simulated motion model, controlled by action atoms obtained from SGT–traversal. This is done until enough information can be obtained from consecutive images again, e.g., due to an dissolving occlusion in the scene.

an incremental recognition of situations. The name of the actually instantiated situation scheme was printed to a stream. This stream was analysed in each (temporal) step, which led to new state atoms for the next point of time according to the simulation model. The only action needed here therefore is a print–command. To express the knowledge represented by the SGT depicted in Fig. 2, we start with an initial task–network d. This network contains only one non–primitive task N_0, which is bound to the task–name `pre_cross_0(Agent)`. As explained in the previous section, this pre–task is needed to express the self–prediction of the situation scheme `cross_0`. The preconditions of this task are given by the state atoms of the situation scheme `cross_0`. Thus, the complete initial task network can be formulated as

$$d = \left\langle \begin{array}{c} N_0 : \texttt{pre\_cross\_0(Agent)}, \\ ? \xrightarrow{\texttt{agent(Agent)}} N_0 \ , \ ? \xrightarrow{\texttt{traj\_active(Agent)}} N_0 \end{array} \right\rangle .$$

To express the self–prediction of the scheme `cross_0`, we use two reduction schemes for the non–primitive task N_0, namely

$$r_{0,0} = \left\langle \begin{array}{c} N_1 : \texttt{note(cross(Agent))}, \\ N_0 : \texttt{pre\_cross\_0(Agent)}, \\ N_1 \prec N_0 \end{array} \right\rangle$$

and

$$r_{0,1} = \left\langle N_1 : \texttt{note(cross(Agent))} \right\rangle .$$

The scheme `cross_0` was specialized by a graph consisting of three consecutive situation schemes. Each of these schemes again was connected to itself by a self–prediction. Thus, we get the following reduction scheme for the non–primitive task N_1:

$$r_{1,0} = \left\langle \begin{array}{c} N_2 : \texttt{pre\_drive\_to\_intersection\_1(Agent)}, \\ N_3 : \texttt{pre\_drive\_on\_intersection\_1(Agent)}, \\ N_4 : \texttt{pre\_leave\_intersection\_1(Agent)}, \\ ? \xrightarrow{\texttt{enter\_lane(Lane)}} N_2 \ , \ ? \xrightarrow{\texttt{on(Agent,Lane)}} N_2, \\ ? \xrightarrow{\texttt{direction(Agent,Lane,straight)}} N_2, \\ ? \xrightarrow{\texttt{crossing\_lane(Lane)}} N_3 \ , \ ? \xrightarrow{\texttt{on(Agent,Lane)}} N_3, \\ ? \xrightarrow{\texttt{direction(Agent,Lane,straight)}} N_3, \\ ? \xrightarrow{\texttt{exit\_lane(Lane)}} N_4 \ , \ ? \xrightarrow{\texttt{on(Agent,Lane)}} N_4, \\ ? \xrightarrow{\texttt{direction(Agent,Lane,straight)}} N_4, \\ N_2 \prec N_3, N_3 \prec N_4 \end{array} \right\rangle .$$

Again, each of the three pre–tasks (N_2, N_3, and N_4) has to be supplied with two reduction schemes expressing the self–prediction of the situation schemes to which they correspond. We thus obtain the two reduction schemes

$$r_{2,0} = \left\langle \begin{array}{c} N_5 : \texttt{note(drive\_to\_intersection(Agent, Lane))}, \\ N_2 : \texttt{pre\_drive\_to\_intersection\_1(Agent)}, \\ N_5 \prec N_2 \end{array} \right\rangle ,$$

$$r_{2,1} = \left\langle\, N_5 : \mathtt{note(drive\_to\_intersection(Agent, Lane))} \,\right\rangle ,$$

and two similar reduction schemes for each of the tasks N_3 and N_4. The transformation of the specializing situation graph of the scheme `drive_to_intersection_1` into the HTN–formalism is somewhat more complicated: This graph contains several loops in addition to the self–predictions, each starting and ending in the situation scheme `proceed_to_intersection`, because this is the only start– and end–situation of that graph. A transformation of this graph can only be done by translating each path from a start–situation to an end–situation into a separate reduction scheme. As the graph to be transformed here also contains self–prediction, the transformation of paths from start– to end–situation has to be done first. This leads to the following reduction schemes:

$$r_{5,0} = \left\langle \begin{array}{c} N_8 : \mathtt{pre\_pre\_proceed\_to\_intersection\_2(Agent)}, \\ ? \overset{\mathtt{speed(Agent,non\_zero)}}{\longrightarrow} N_8 \end{array} \right\rangle .$$

denotes the path starting in `proceed_to_intersection_2(Agent)` and ending there, visiting no other situation scheme. Another path visiting the situation scheme `stop_in_front_of_intersection_2(Agent)` is represented by

$$r_{5,1} = \left\langle \begin{array}{c} N_8 : \mathtt{pre\_pre\_proceed\_to\_intersection\_2(Agent)}, \\ N_9 : \mathtt{pre\_pre\_stop\_in\_front\_of\_intersection\_2(Agent)}, \\ N_{10} : \mathtt{pre\_pre\_proceed\_to\_intersection\_2(Agent)}, \\ ? \overset{\mathtt{speed(Agent,non\_zero)}}{\longrightarrow} N_8 \ , \ ? \overset{\mathtt{speed(Agent,very\_small)}}{\longrightarrow} N_9, \\ ? \overset{\mathtt{speed(Agent,non\_zero)}}{\longrightarrow} N_{10}, \\ N_8 \prec N_9, N_9 \prec N_{10} \end{array} \right\rangle .$$

The last path finally visits all four situation schemes of the graph and is represented by

$$r_{5,2} = \left\langle \begin{array}{c} N_8 : \mathtt{pre\_pre\_proceed\_to\_intersection\_2(Agent)}, \\ N_9 : \mathtt{pre\_pre\_stop\_in\_front\_of\_intersection\_2(Agent)}, \\ N_{11} : \mathtt{pre\_pre\_wait\_in\_front\_of\_intersection\_2(Agent)}, \\ N_{12} : \mathtt{pre\_pre\_start\_in\_front\_of\_intersection\_2(Agent)}, \\ N_{10} : \mathtt{pre\_pre\_proceed\_to\_intersection\_2(Agent)}, \\ ? \overset{\mathtt{speed(Agent,non\_zero)}}{\longrightarrow} N_8 \ , \ ? \overset{\mathtt{speed(Agent,very\_small)}}{\longrightarrow} N_9, \\ ? \overset{\mathtt{speed(Agent,zero)}}{\longrightarrow} N_{11} \ , \ ? \overset{\mathtt{speed(Agent,very\_small)}}{\longrightarrow} N_{12}, \\ ? \overset{\mathtt{speed(Agent,non\_zero)}}{\longrightarrow} N_{10}, \\ N_8 \prec N_9, N_9 \prec N_{11}, N_{11} \prec N_{12}, N_{12} \prec N_{10} \end{array} \right\rangle .$$

The pre–pre–tasks introduced in the reduction schemes above can then be further reduced with respect to self–prediction as exemplified above. Thus, the complete SGT described in Sect. 3 can be translated into the HTN–formalism of [13].

5 Conclusion

In the preceding section, SGTs and HTNs have been compared. A single SGT incorporates the complete knowledge about the behavior of agents in a discourse, while a single HTN always expresses a single plan in the world modelled by tasks and reduction schemes. Thus an SGT is rather comparable to all HTNs that can be build with these tasks and reduction schemes than to a single HTN. In other words: *One HTN is comparable to one instance of the schematic behavior description given by an SGT.* As a further result it can be stated that SGTs and HTNs correspond structurally in most aspects except for the following details:

– The (partial) temporal ordering of tasks in HTNs is an order–relation, while the relation defined on situation schemes by prediction edges is more general.
– Because of the strict temporal order between tasks, HTNs do not naturally support loops, though loops and self–predictions of SGTs can also be modelled with HTNs.
– HTNs explicitly denote add– and delete–effects of tasks. In SGTs, the effects of actions are obtained by sensor input from an observed (simulated) world.

Due to the considerable structural correspondences between SGTs and HTNs, the difference between the planning task conducted on the HTN–structure and the observation task utilizing SGTs predominantly lies in the algorithm performing the planning or the observation task respectively. One way to facilitate a planning task with the knowledge encoded in an SGT could therefore be the translation of the SGT into an HTN–syntax as outlined in Sect. 4. One of the HTN–planning algorithms mentioned in Sect. 2.1 could then be applied to the resulting set of tasks and reduction schemes. Because SGTs do not denote effects of action atoms, these effects would either have to be incorporated into the resulting HTN–formulation, or they would have to be obtained by simulated sensor input as described in [8]. This way of planning on SGTs would *adapt* the underlying knowledge–structure and leave the algorithm performed on that structure *as is*.

Another way to facilitate planning using the knowledge given in the form of an SGT is to modify one of the HTN–planning algorithms mentioned in Sect. 2.1 in a way that it can cope with the representation of SGTs and the differences with respect to the HTN–formulation arising due to that representation. Again, the results of action atoms comprised in the SGT would either have to be incorporated into the SGT or obtained from a simulation. The result of such an approach would be an *adapted* planning–algorithm, running on a knowledge–structure that could be left *as is*.

An analogous conclusion can be drawn concerning the feasibility of the HTN–formalism for plan–recognition or observation–tasks: if SGT– and HTN–formalisms are as far matchable as implied by the comparison drawn in Sect. 4, one could think of using the HTN–formulated knowledge to recognize (or observe) the behavior of an agent in the discourse domain for which the HTN was designed. As stated above regarding the utilization of SGT for planning, this could again be done by either a transformation of the HTN–formulated knowledge into an SGT, on which the graph–traversal outlined in Sect. 3 would then be applied, or the algorithm performing the graph–traversal could be adapted to the HTN–formalism.

6 Future Work

The correspondencies found between the SGT– and HTN–formalisms suggest a possible automatic transformation of one formalism into the other and vice versa, though further investigations have to show if such a transformation is algorithmically feasible. The possible use of the knowledge coded in HTNs for an *observation task* is interesting, but not our primary goal due to our experience with SGTs. We rather want to try to implement an HTN–planning algorithm on SGTs, *adapt* them to that *planning task* as much as neccessary without losing their capabilities for the observation task. The resulting structure could then (hopefully) be used for both, planning and observation tasks.

References

1. K. Erol, J. Hendler and D. S. Nau: *UMCP: A Sound and Complete Procedure for Hierarchical Task–Network Planning.* In: K. J. Hammond (Ed.): Proc. of the 2nd Int. Conf. on Artificial Intelligence Planning Systems (AIPS–94), June 13–15, 1994, University of Chicago, Chicago, Illinois, 1994, pp. 249–254.
2. K. Erol, J. Hendler and D. S. Nau: *HTN Planning: Complexity and Expressivity.* Proc. of the 12th National Conf. on Artificial Intelligence (AAAI–1994), Volume 2. Seattle, Washington, USA, July 31 – August 4, 1994. AAAI Press, 1994, pp. 1123–1128.
3. K. Erol, J. Hendler and D. S. Nau: *Complexity Results for HTN Planning.* Annals of Mathematics and Artificial Intelligence **18**:1 (1996) 69–93.
4. R. E. Fikes and N. J. Nilsson: *STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving.* Artificial Intelligence **2** (1971) 189–208.
5. K. Fleischer and H.–H. Nagel: *Machine–Vision–Based Detection and Tracking of Stationary Infrastructural Objects Beside Innercity Roads.* In: Proc. of the IEEE Intelligent Transportation Systems Conf. (ITSC), August 25–29, 2001, Oakland, CA, USA, pp. 525–530.
6. K. Fleischer, H.–H. Nagel, and T. M. Rath: *3D–Model–Based–Vision for Innercity Driving Scenes.* In: Proc. of the IEEE Intelligent Vehicle Symposium (IV–2002) June 18–20, 2002, Versailles, France.
7. R. Gerber: *Natürlichsprachliche Beschreibung von Straßenverkehrsszenen durch Bildfolgenauswertung.* Dissertation, Fakultät für Informatik der Universität Karlsruhe (TH), Karlsruhe, Januar 2000; erschienen im elektronischen Volltextarchiv der Universität Karlsruhe, `http://www.ubka.uni-karlsruhe.de/cgi-bin/psview?document=2000/informatik/8` (in German).
8. M. Haag: *Bildfolgenauswertung zur Erkennung der Absichten von Straßenverkehrsteilnehmern.* Dissertation, Fakultät für Informatik der Universität Karlsruhe (TH), Karlsruhe, Juli 1998; erschienen in der Reihe Dissertationen zur Künstlichen Intelligenz (DISKI) **193**; infix–Verlag Sankt Augustin 1998 (in German).
9. M. Haag and H.–H. Nagel: *Incremental Recognition of Traffic Situations from Video Image Sequences.* Image and Vision Computing **18**:2 (2000) 137–153.
10. R. J. Howarth and H. Buxton: *Conceptual Descriptions from Monitoring and Watching Image Sequences.* Image and Vision Computing **18**:2 (2000) 105–136.
11. S. Kambhampati: *A Comparative analysis of Partial Order Planning and Task Reduction Planning* SIGART Bulletin **6**:1 (1995) 16–25.

12. H. Kautz, D. McAllester, and B. Selman: *Encoding Plans in Propositional Logic.* In: L. C. Aiello, J. Doyle, and S. C. Shapiro (Eds.): Proc. of the 5th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts, USA, 1996; Morgan Kaufman, San Mateo, CA, USA 1996, pp. 374–384.
13. A. D. Mali and S. Kambhampati: *Encoding HTN Planning in Propositional Logic.* In: R. G. Simmons, M. M. Veloso, and S. Smith (Eds.): Proc. of the 4th Int. Conf. on Artifical Intelligence Planning Systems (AIPS–98), Pittburgh, Pennsylvania, USA, 1998; AAAI–Press, 1998, pp. 190–198.
14. D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz–Avila: *SHOP: Simple Hierarchical Order Planner.* In: Th. Dean (Ed.): Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI–99), Stockholm, Sweden, July 31 – August 6, 1999; Morgan Kaufmann, San Mateo, CA, USA 1999, pp. 968–973.
15. D. S. Nau, H. Muñoz–Avila, Y. Cao, A. Lotem, and S. Mitchell: *Total–Order Planning with Partially Ordered Subtasks.* In: B. Nebel (Ed.): Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI–2001), Seattle, Washington, USA, August 4–10, 2001; Morgan Kaufman, San Mateo, CA, USA 2001, pp. 425–430.
16. H.–H. Nagel: *From Image Sequences towards Conceptual Descriptions.* Image and Vision Computing **6**:2 (1988) 59–74.
17. H.–H. Nagel, M. Haag, V. Jeyakumar, and A. Mukerjee: *Visualization of Conceptual Descriptions Derived from Image Sequences.* Mustererkennung 1999, 21. DAGM–Symposium, Bonn, 15.–17. September 1999, Springer–Verlag, Berlin, Heidelberg, u.a. 1999, pp. 364–371.
18. H.–H. Nagel: *Natural Language Description of Image Sequences as a Form of Knowledge Representation.* In: W. Burgard, T. Christaller, and A. B Cremers (Eds.): Proc. of the 23rd Annual German Conf. on Artificial Intelligence (KI–99), Bonn, Germany, September 13–15, 1999, LNCS **1701**, Springer–Verlag, Berlin, Heidelberg, u.a. 1999, pp. 45–60.
19. J. Pearl: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufman, San Mateo, CA, USA 1988.
20. S. Russel and P. Norvig: *Artificial Intelligence: A Modern Approach.* Prentice Hall, Upper Saddle River, New Jersey, USA, 1995.
21. E. D. Sacerdoti: *Planning in a Hierarchy of Abstraction Spaces.* Artificial Intelligence **5** (1974) 115–135.
22. 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).
23. G. Socher, G. Sagerer, and P. Perona: *Bayesian reasoning on qualitative descriptions from images and speech.* Image and Vision Computing **18**:2 (2000) 155–172.