MOBAL's Predicate Structuring Tool

Volker Klingspor

24. Sept. 1991

Gesellschaft f"ur Mathematik und Datenverarbeitung
Schlo"s Birlinghoven
Sankt Augustin

Contents

1 Introduction

1.1 The problem

For acquisition of knowledge it is necessary to understand the whole knowledge base, not only single items of it. Littman writes 1987 [Kar88]

> *"...it would be very useful to study the organization of our knowledge engineer knowledge."*

Thus, we have to understand the structure of the knowledge. There are two types of data that can be structured. The first are the objects of the knowledge base. These are structured by a taxonomy of sorts like the STT of MOBAL [Kie88]. The second are the predicates of a knowledge base. The issue of this paper is to describe a tool called PST (predicate structuring tool), which structures these predicates. This is similar to the inference layer of KADS [BW89].

The dependencies between the predicates are given by the rules which can be represented by a rule graph. But we have a lot of troubles to present a rule graph of a large knowledge base on the screen, so normally the user has to focus on a little section of the graph. And it is impossible to understand the structure of a knowledge base by seeing only a small part of the dependencies. Thus, focussing alone is not sufficient for the inspection of the rules.

A second approach to reduce the graph is the abstraction of it. We can merge several predicates with the same intention to one node of the rule graph. By this merge, the number of nodes are reduced just as the number of edges between nodes. The abstraction destroys local dependencies, but the structure of the graph will be conserved.

The abstract representation of a knowledge base has several advantages:

- The user gets an overall view over the knowledge.
- The analysis and the integration of data is more simple.
- The user gets a guidance for the further knowledge acquisition.

In MOBAL, this abstract dependency graph is call *topology*. This paper describe the operations on topologies and a tool which builds a topology by abstracting the rule graph.

1.2 The tasks of the PST

In this section, we present some tasks which will be performed by the PST, because the PST is not only useful for inspection.

Inspectability: The user must be able to understand the knowledge base, supported by the PST.

Modularisation: The user can define modules of the knowledge base, the interfaces of two modules are described by edges between nodes of the modules.

Task-oriented representation: The user can give a task structure by the PST. This structure can be different from the given rule base.

Reduction of the hypothesis space: MOBAL's rule discovery tool (RDT) uses the PST to restrict the learnable rules to those rules which are compatible with the topology.

Focussing the learning: The difference between the user-given task oriented topology and a system-generated topology can be used for focussing the learning on some very interesting rules.

Automatic generation of a topology: The predicate structuring tool generates the inference structure of the knowledge base by abstracting the rule graph.

2 The PST: Representation and Operations

In MOBAL it is possible to create as many topologies as you like[1]. Every topology is a directed acyclic graph, a dag. All predicates of the knowledge base are attached to the nodes of the graph. The edges of a topology define possible or desired inferences, depending on the purpose of this topology. Every topology contains an isolated node, that means this node is adjacent to none of the others. It is called basic node. All not classified predicates are attached to this node.

The nodes of the topologies are described by the six-ary Prolog- predicate

`tnode (ID, Nodename, Comment, Predicates, Links, Topologyname)`[2]

ID: The ID is an unique internal number for access by the system. The exception is the basic node of a topology, it has the ID `Topologyname:basic_node`.

Nodename: The name serves for the access by the user and must be given by him.

Comment: Additional information given by the user.

Predicates: A list of predicates of the knowledge base attached to this node.

Links: A list of the predecessors of this node. The predecessors are represented by their ID.

Topologyname: This argument determines the topology the node belongs to.

2.1 The textual interface of the PST

This interface contains the following operations for the user:

- the presentation of a topology
- commands for adding and deleting nodes
- commands for adding and deleting links
- commands for attaching and reattaching predicates to or from nodes

[1] At least one topology must exist, that is normally called `system`

[2] Like usual in PROLOG, variables are written with an upper first character, constants are written with a lower one.

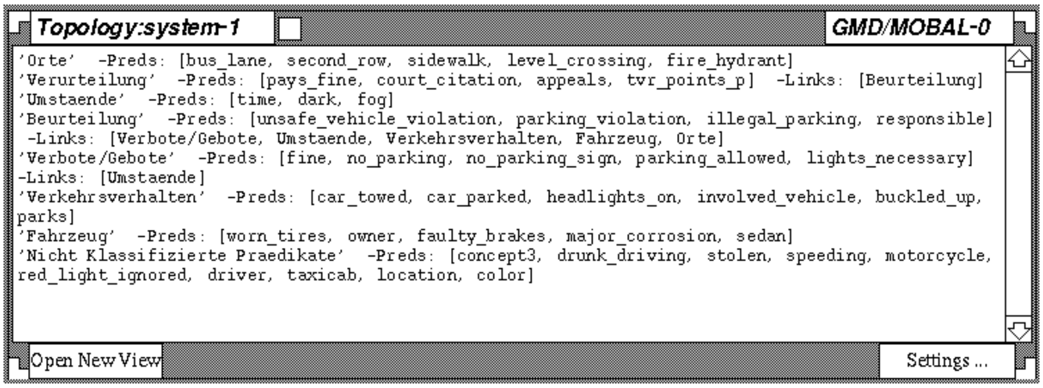

Figure 1: textuell presentation of the topology

- a command for creating new topologies.
- a command for generating a topology as an abstract rule graph

2.1.1 The presentation of a topology

The MOBAL-menu contains an item New Topology Window to open such a window. MOBAL asks the user for the topology he wants to open if more than one exists. Then MOBAL opens a HyperNeWS-window like the one shown in figure 1.

The nodes in this window have the following form:

$$\mathit{TopologieName} : \mathit{NodeName} \ -\texttt{Preds} : [\mathit{Pred_1}, \ldots, \mathit{Pred_n}] \ -\texttt{Links} : [\mathit{Link_1}, \ldots, \mathit{Link_m}]$$

The links will not be printed if the set is empty. They will be printed by their node-names instead of their IDs.

The user can do the normal operations like scrolling the window, resizing the window, moving or closing it. He can focus on a subset of the nodes by clicking on the button settings and entering a string that must be a substring of the focused nodes. He can sort the nodes by an arbitrary order, by giving a relation in the field sort (see figure 2).

If the user double-clicks on a presented node, a stack called node command stack opens to enter more commands, described in the following sections.

2.1.2 Entering of topology nodes

There are two possibilities to enter nodes. The first, Prompted Input in the MOBAL-menu, is interactive, the single items are requested by the system. The second is the input via the MOBAL- scratchpad. The scratchpad format is the same as the format of the presentation of nodes, only the keyword tnode must precede the entry [3]. A comment can precede the input.

; *Comment*

[3] Between the dash and the word Preds resp. Links must not be a space.

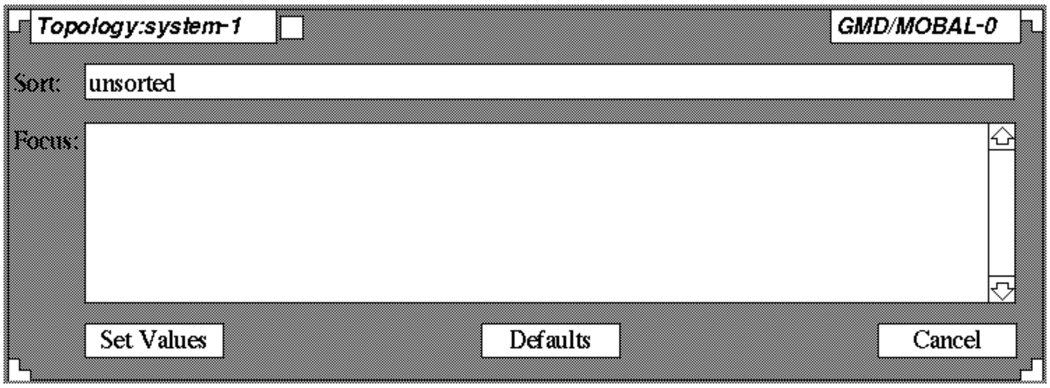

Figure 2: Focusing and sorting of nodes

$\texttt{tnode} : \mathit{TopologieName} : \mathit{NodeName}\ -\texttt{Preds} : [\mathit{Pred_1}, \ldots, \mathit{Pred_n}]\ -\texttt{Links} : [\mathit{Link_1}, \ldots, \mathit{Link_m}]$

The user must conserve the following restrictions:

- The names of nodes must be unique relative to all topologies. If a node with the same name exists, the system adds the first number n to the name of the new node, so that the nodename is unique.
- The input of comment is optional, if it isn't given, the comment of the node stays uninstantiated.
- The input of Preds and Links are also optional.
- If the set of predicates isn't empty, the system attaches these predicates to the new node and removes them from all other nodes of this topology. Unknown predicates are ignored.
- If the set of links isn't empty, all links that imply no cycles are added. If a link implies a cycle, the system asks the user whether it should shrink this cycle to one node or ignore this link. Unknown predecessors produce entries in the agenda of MOBAL (described later).

2.1.3 Deleting of nodes

Deleting of nodes can be forced by clicking the button `Delete` of the command-stack of this node. The predicates of this node will be classified to the basic node of this topology by the system.

2.1.4 Entering of links

The user can add new inks to a node by clicking `Add Link`. He can enter a single predecessor or a list of predecessors that will be add to the topology. The predecessors must be given by the nodename. If the new link creates cycles the same procedure as for entering new nodes will proceed.

2.1.5 Deleting links

By clicking `Remove Link` the user can delete a single link or a set of links.

2.1.6 Attaching predicates to a node

This is analogous to adding links, if the user clicks `Add Preds`. The specified predicates are removed from the other nodes and will be attached to the given node.

2.1.7 Removing predicates from a node

Clicking `Remove Pred` and entering a single predicate or a list of predicates will force an attaching of this predicates to the basic node of the topology.

2.1.8 Creating new topologies

The user can create an additionally topology by choosing the item `Create New Topology` of the MOBAL-menu. He must enter a new name for this topology.The created topology consists of a single node, the basic node. All predicates of the knowledge base are attached to this node.

2.1.9 Generating an abstraction of the inference structure

If the user clicks the menu item `Generate Topology`, the system applies several algorithms to the rule of the knowledge base. These algorithms can be chosen by the user. A detailed description of the existing algorithms follows in section 3.

2.2 The graphic interface

For the graphical presentation of the several types of data, Christian Haider has programmed a graphic interface [Hai90]. The user is not able to change the shown topology, but he can change the presentation of it. It is possible to create as many graphics as the user like, each in its own window.

The graphic will be displayed when the user clicks the button `Graphic` on the command stack of a node. He has to distinguish two cases:

- The chosen node is the basic node. Then the whole topology will be displayed. Isolated nodes like the basic node will not be displayed. Figure 3 shows such an output.

- The chosen node is not the basic node. This node and all adjacent nodes with the connecting edges will be displayed. This is a focused presentation of the topology, which can be expanded. The way to this is explained later. Figure 4 shows this operation on the node `Obligations/Prohibitions`.

Since the operations have no influence to the knowledge base, there exist no operations for deleting or adding nodes or links and so on. The operations change only the way of presenting the graph. There are four types of operations:

1. general operations on the window

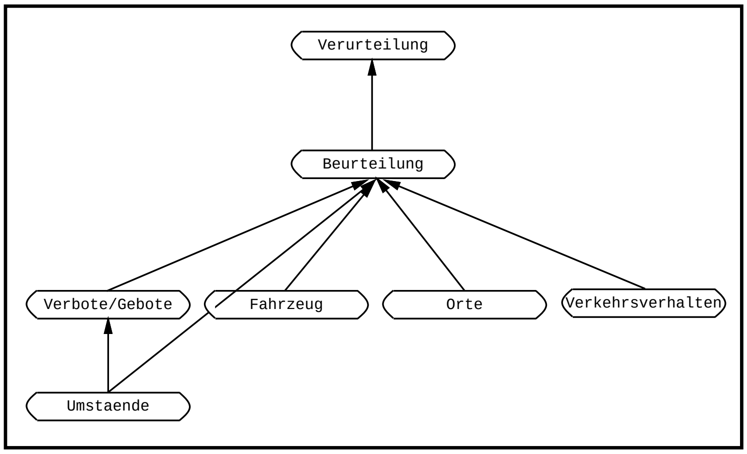

Figure 3: The user given topology of the domain Traffic Law

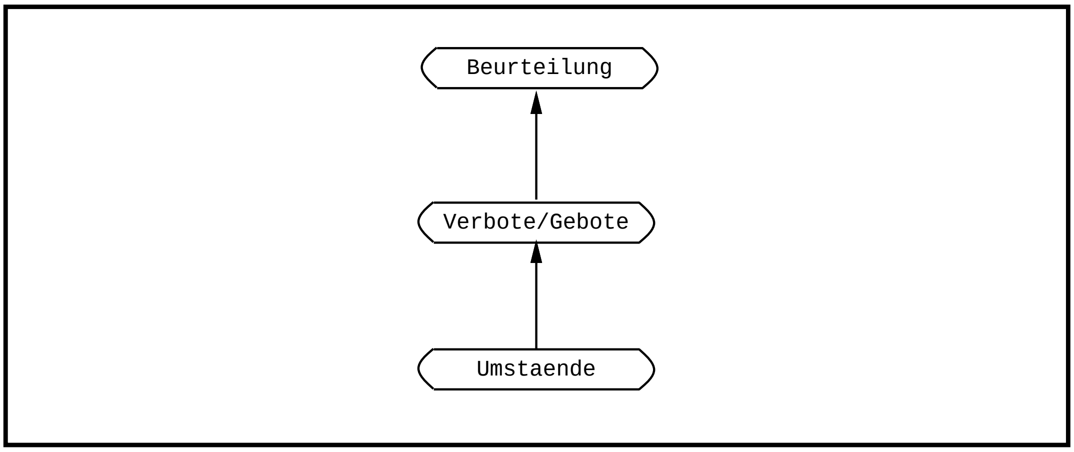

Figure 4: Focused presentation of the topology of Traffic Law

2. operations on the canvas
3. operations on the nodes
4. operations on the links

2.2.1 General operations on the window

The window can be closed finally by clicking on the square in the top of the window and iconized by clicking the other button. It can be resized by moving the lower right corner of it and moved by moving the top bar during pressing the middle mouse button. The width of the graph can be scaled by choosing a factor from one of $\{1/5, 1/4, 1/3, 1/2, 2, 3, 4, 5\}$ which is multiplied with the current scale factor.

2.2.2 Operations on the canvas

Left mousebotton: The user can select a set of nodes with a rubberbox.

Middle mousebotton: The user can move the graph on the canvas.

Ctrl + left mousebotton: The graph will be redrawn for displaying changes of the topology.

Shift + left mousebotton: The graph will be displayed centered.

Meta + left mousebotton: The size of the window will be toggled between the original size and the maximal size.

Meta + right mousebotton: The layout algorithms will be called.

Right mousebotton: The system shows a help stack.

2.2.3 Operations on the nodes

Left mousebotton: The node will be marked as selected or deselected deselected, depending on its previous state.

Ctrl + left mousebotton: The clicked node will be expanded or reduced. Expanding forces a drawing of all adjacent nodes of the chosen node. If the node is already expanded, the expansion will be made back, so that all adjacent nodes are removed.

Middle mousebotton: The chosen node can be moved. Is the node selected, all selected nodes will be moved.

Shift + middle mousebotton: The graph will be moved so, that the chosen node is in the center of the window.

Right mousebotton: The system shows the help stack for the node operations.

Ctrl + right mousebotton: The system displays the predicates, which are attached to this node.

2.2.4 Operations on the edges

Right mousebotton: The system shows the help stack.

Ctrl + right mousebotton: Source and target of the edge will be displayed.

2.3 The program interface

The program interface of MOBAL is developed by Jörg-Uwe Kietz [Kie90]. The interface makes five Prolog predicates available for external systems.

`mobal_get_tnode (!NodeStruc)`

The call can be made by an arbitratily instantiated atom of the form `tnode(_, _,_,_,_,_)`. MOBAL tries to unify the given NodeStruc with a node of a topology. So long as this is possible, backtracking returns alternatives.

`mobal_delete_tnode (!NodeStruc)`

With this call, the external system can force the removing of a topology node. If more than one unification of NodeStruc with a node of a topology is possible, the result of this call is unpredictable. NodeStruc will not be instantiated by the call.

`mobal_new_tnode (!NodeStruc)`

The node will be asserted into the topology. If the topology name of the node is uninstantiated, MOBAL adds the node to the topology `system`. If the ID is not given, MOBAL generates a new ID and instantiates the first parameter of NodeStruc with this ID. The other parameters of NodeStruc are handled like in section 2.1.2.

`mobal_tnode_stored (!NodeStruc)`

MOBAL calls this predicate every time a topology node has been added. If the external system defines `mobal_tnode_stored/1`, it has control over the entering of nodes.

`mobal_tnode_deleted (!NodeStruc)`

When a node has been deleted, MOBAL calls this predicate. It has the same sense like the previous predicate.

2.4 The Agenda

Tasks of the user which are still to be handled are managed by MOBAL's agenda. If the user adds nodes with unknown predecessors, the system can't insert such an edge. Instead of this, it adds the item `tnode_undefined` into the agenda. This item shows the user that he must define the specified node. Is this done, the system automatically inserts the missing edge and removes the agenda entry.

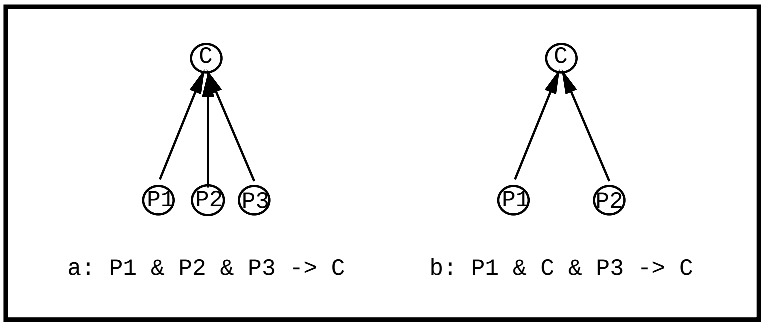

Figure 5: Rules and their resulting graphs

3 The automatic generation of a topology

This section is about the automatic generation of a topology by abstracting the inference structure of the knowledge base. We will introduce several techniques to build such an abstraction. All of the techniques use the rule graph of the knowledge base. Thus, first we have to define and build this rule graph.

3.1 Building the rule graph

In our context, the rule graph is a directed, possible cyclic graph with exactly one node for every predicate that occurs in a rule. Every rule, here in HORN clause logic, defines multiple edges between the several premisses and the conclusion. A recursive occurrence of a predicate, i.e. both as a premise and as a conclusion of a rule, will not make an edge. Figure 5 shows the graphs of two rules.

In MOBAL, there is the possibility to use special predicates. These are autoepistemic operators[4], build-in-operators[5] and the operator `not`. For all these operators, not the operator itself is used in the rule graph. Instead of this, the arguments of the operator are checked, whether they are predicates of the knowledge base and if this is the case, these arguments are used to build the rule graph.

Thus, the algorithm for each rule is:

1. Collect all premises of this rule.
2. Generate for every premise, if not yet existant, a topology node.
3. Generate also for the conclusion, if not existant, a node.
4. Enter for every premise of the rule an edge from this premise node to the conclusion node.

[4] `max_of, count, sum_of, unknown`

[5] Basic computations (`add, sub, prod, div`), comparisons (`eq, neq, gt, lt, ge, le`)

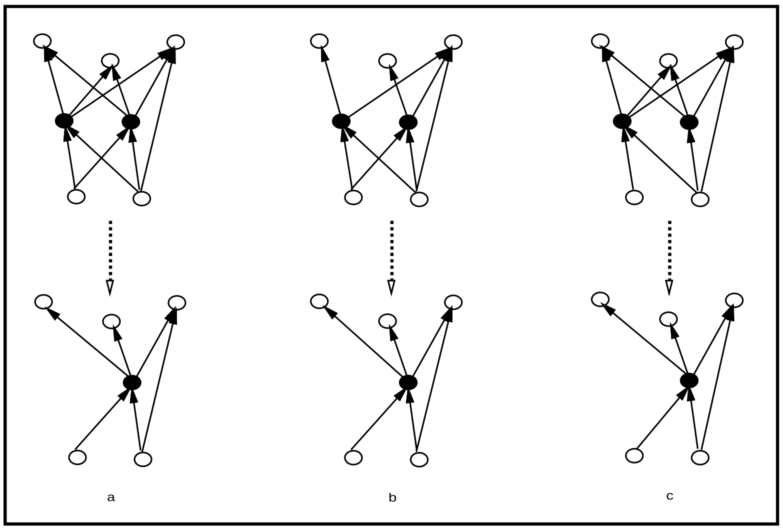

Figure 6: Merging of nodes

3.2 Shrinking of strong components

Here, we will describe the first algorithm for abstracting the rule graph. Subgraphs of a directed graph are said to be strong components, if, for all pairs of nodes i and j, there exists a directed path from i to j and back. Strong components consists of at least one cycle, shrinking of all strong components leads to an acyclic graph. This is a handy abstraction, since normally the computations in such cycles are local, and local computations are not sufficient for understanding the structure of a knowledge base. Thus the first algorithm is the O(n) algorithm of R. E. Tarjan [Sed88] to find the strong components and shrink each to one node. The algorithm bases on depth first search (like Nilsson [Nil82]). If a cycle is found, it will be shrinked during backtracing to the first visited node of the cycle. Thus every node is only visited once.

3.3 Merging nodes depending on neighborhoods

Shrinking of strong components alone is not a sufficient abstraction. The reduction of a graph is too small. Thus we need additional operations. We present in this section operations based on similarities of the predecessors and successors. The PST contains two algorithms with these operations.

- Merging nodes with the same predecessors.
- Merging nodes with the same successors.

The first algorithm leads to more tree-like graphs, the second leads to a graph that is smaller in the base. Both results make the graph easier to survey.

Figure 6 shows examples for these two cases and also (part a: of the figure) the cased that both, predecessors and successors must be equal.

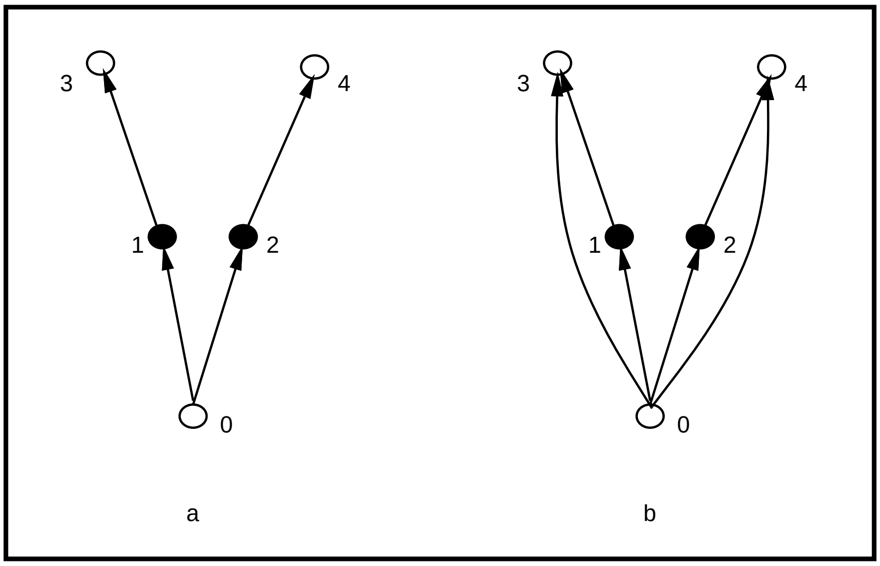

Figure 7: Problems with the control structure

First we have some troubles with the sequence of the merging, because the merge of two nodes can imply new possibilities for merging other nodes. So we need an order, which guarantees all possible merges. Normal breath first search order can't do this, because we can have multiple paths to a node. Figure 7:b shows a situation where the nodes 1 and 3 both have paths from the node 0 of length one. So it is arbitrary whether breath first search visit first the node 1 or first node 3. But if node 1 is visited first, and nodes with same predecessors are merged, the nodes 1 and 2 are first merged and than the nodes 3 and 4, because now, they have the same predecessor. Thus, we need an order, which guarantees to visit first nodes 1 and 2 and than 3 and 4. This is done by the topological order, that means, direct and indirect successors of a node in the graph are also direct or indirect successors relative to the order.

A topological order can be found by a modified depth first search. Normally, in depth first search, the visited node is handled first and than the successors of this node are visited. If we handle the node *after* visiting succeeding nodes, we have a reverse topological order. Thus we can control the merge like following:

same predecessors: We use the modified depth first search to visit the graph contrary to the directions of the edges. So the order is topological, the search handles the predecessor-free nodes first.

same successors: The modified depth first search visit the graph normally, the resulting order is the reverse topological order. The search handles first the successor-free nodes.

Both algorithms lead to a merging of all mergeable pairs of nodes. They run in time $O(n^2)$ for n nodes, $O(n)$ to visit every node and O(n) to search all mergeable partners for each node.

3.4 Merging of outsiders

Sometimes, there exists some nodes attached by only one predicate and linked to only one other node. These nodes we called *outsiders*. There is no lost of structure if we drag these nodes into the graph by merging them with the linked node. The necessary time to find and merge all outsiders is $O(n)$.

3.5 Merging of successor free nodes

Sometimes it is useful to merge all top level nodes of the graph. That leads to a topology with only one target node. In our domains, the result of this operation was a loss of structure, so we don't use this operation for the examples.

3.6 Other methods

Although the results of the algorithms above are good, we searched for other methods based on graph theory, to find incremental algorithms. We tried to use cut sets (like in the flow theory) but there are too many distinct cut sets and it is not possible to choose one of them. We also tried to use partitions but we saw no sense in this. We think the user is able to build good topologies with the given algorithms, because he can understand the actions.

3.7 The application of the algorithms by the user

The PST of MOBAL contains the parameter pst_generating_algorithms. This parameter is a list of algorithms, which will be applied to the rule base, if the user choose the MOBAL-menu entry Generate Topology. The possible items of the list are:

generate: The PST will build the rule graph of the knowledge base and shrink the strong components to nodes with the name `cycle_`n.

top_down: The graph will be visited by reverse topological order to merge all nodes with the same successors.

bottom_up: The graph will be visited in topological order to merge all nodes with the same predicates.

merge_top_level_nodes: The successor free nodes will be merged.

merge_outsiders: The outsiders of the graph will be dragged into him.

The algorithms are applied to the topology that is determined by the parameter `pst_topology_for_generating`. This topology must exist. The user can create it by the MOBAL-menu item `Create Topology` as described in section 2.1.8.

4 Topology-influenced learning

4.1 Learning in MOBAL

MOBAL's learning tool, the RDT, is a model based algorithm [KW92]. The hypothesis space is restricted by

- rule models and an order on this models.
- the number of arguments of the predicates in the rule models.
- the argument sorts, given by the user or built by the STT [Kie88].

There are two main reasons to restrict the hypothesis space also by the PST. First, if the user enters a topology, he gives the system a task structure. Thus, it is wise to use this structure to prevent the generation of senseless hypotheses. Learning by induction contains always the possibility of learning senseless rules, if there is an evidence for this rule in the knowledge base. Thus, the learned rule must be compatible with the task structure.

The second reason is that the restriction of the hypothesis space reduces the necessary time to test this space, so that learning becomes faster.

4.2 The restriction of learning

The PST enables a definition for rules that are admissible:

> *A rule is admissible relative to a topology, if the premises are only attached to nodes which are either predecessors of the conclusion node or this node itself.*

Carried over to pairs of predicates, the compatibility of predicates is defined as following:

> *A pair of predicates, a premise and a conclusion, is topology compatible, if the premise is attached to a predecessor of the conclusion node, or to this node itself.*

The relation of predecessors is not used transitively. The basic node of the used topology is an extraordinary node. The predicates of this node are compatible to all other predicates of the knowledge base.

4.3 The focussing of learning

The PST defines a special technique to focus on interesting hypothesis. If we have two topologies in a knowledge base, maybe the first entered by the user, the second build by the PST, then there are principal three types of predicate pairs.

1. The pair is not compatible relative to the user given topology. That means, the user don't want a rule with this combination.
2. The pair is compatible to both of the topologies. Then the combination of predicates is desired by the user, and the generated topology shows that such a combination really exists in the knowledge base.
3. The pair of predicates is compatible to the user given topology, but it is not compatible to that one, generated by the PST. The user want to have a rule with this combination, but no one exists. These pairs are very interesting to learn rules about.

Corresponding to these three possible combinations, there are three possible usages of the PST, to restrict the learning. The way, the PST restricts, the user can choose by a parameter of RDT, called `rdt_topology_restriction`. This parameter can become one of three values:

no: The PST is not used to restrict the hypothesis space.

yes: The hypothesis space consists of rules, which are compatible to the topology given by the PST-parameter `pst_topology_for_learning`.

focus: The hypothesis space of the previous case is additionally restricted to hypotheses, which are not compatible with the topology defined by the parameter `pst_topology_for_focussing`.

By using the parameters `pst_topology_for_learning` and `pst_topology_for_focussing` the user has to pay attention to the following aspects:

- If the user applies focussed learning, the two topologies have to be different. Otherwise, the system only restricts the hypothesis space normally, not focussed.
- The selection of the topologies is unrestricted. The topologies have not to be chosen like in the argumentation above, the first user given, the second system generated.
- The topologies must exist. The user can't choose other topologies by the human computer interface.

5 Conclusion

5.1 Results

We have used the PST to generate the topologies of two domains. The first is a small domain to test the several learning algorithms and making understandable demonstrations of these algorithms. The domain handles the German traffic law. It consists of 22 rule and 42 predicates. The rule graph of the domain is shown in figure 8. We apply the algorithms generate, top_down and bottom_up to this knowledge base, the result is stated in figure 9[6].

The topology contains five larger groups of predicates:

sidewalk: the places of the domain.

no_parking: places and vehicles are related to the events.

parking_violation: the several violations of the knowledge base.

lights_necessary: some states of vehicles.

time: external conditions, influencing the classification of events.

The we have tested the generation of hypothesis with this topology. We have made five test, all test are made without pruning the hypothesis space and restricting the space by the taxonomy of sorts, since these restrictions can influence the results:

- no topology is used for learning.
- the topology given by the user is applied.
- the topology generated by the PST is used.

[6]The nodes are named by an arbitrary representative of the attached set of predicates

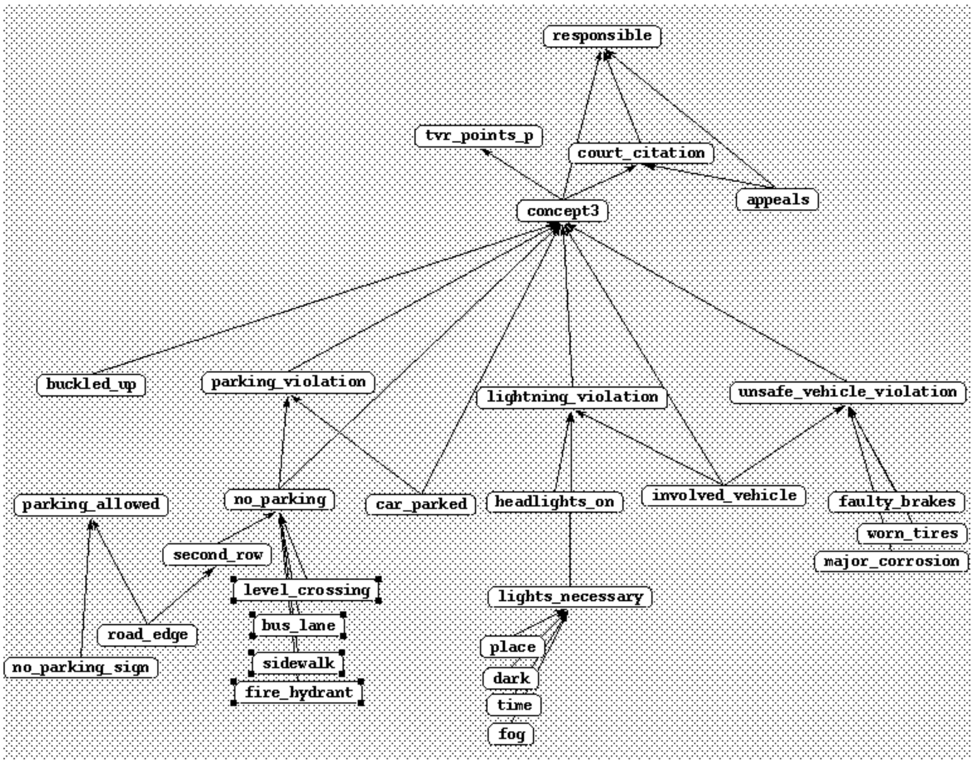

Figure 8: The rule graph of Traffic Law

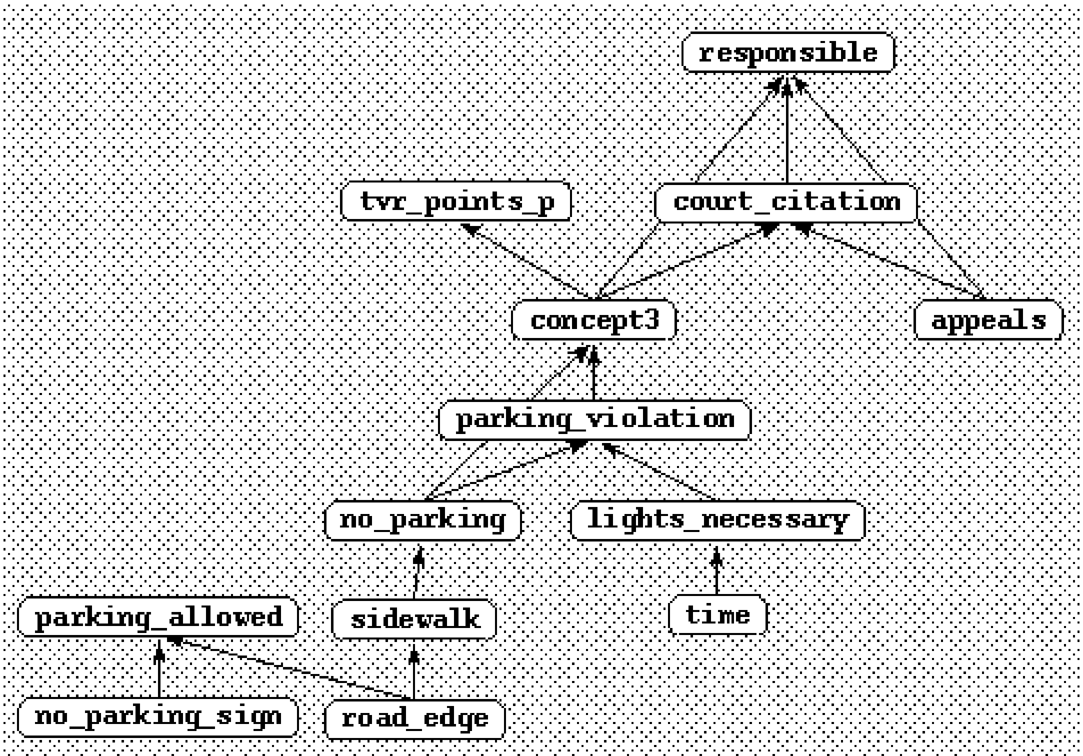

Figure 9: The generated topology of Traffic Law

	number of hypothesis	time to	test hypothesis
no topology	ca. 700 000	ca. 7-8	days
User topology	2 331	37	minutes
system topology	ca. 400 000	ca. 4	days
reduced system topology	2 042	31	minutes
focused learning	263	4	minutes

Figure 10: Results of the restrictions of learning

	#nodes	#edges	degree of reduction	#merged nodes
rule graph	125			
strong components	117	234	6 %	8
top-down	74	172	37 %	43
bottom-up	56	129	24 %	18
merge-outsiders	49	122	14 %	7
total	49	122	61 %	76

Figure 11: Results of the generation of a topology

- same as the previous, but without the unclassified predicates attached to the basic node.

- both topologies are used for focused learning.

We conducted the fourth experiment, because the existence of unclassified predicates blow up the hypothesis space enormously. Figure 10 shows the result of our experiment.

The second domain we have used to test the algorithms of PST is a large medical domain, called ICTERUS, containing 267 rules for 127 predicates. We have applied all of our algorithms to build a topology. The number of nodes which could be merged and the degree of reduction of nodes is presented in figure 11. Although the graph is not very small, the user can identify the several tasks of the inference process.

5.2 Discussion

First we give several advantages of building the topology as described:

- The algorithms are not based on heuristics, so the user can keep track of the merging of nodes very easyly.

- The algorithms are very quick, they all run in time $O(n^2)$.

- The set of compatible predicate pairs grows strictly monotonous during the process of building the topology.

- A single merge doesn't create new paths between the direct predecessors and successors of the merged nodes.

- The choice of the merged nodes guaranties that no new cycles spring up.
- The flexibility in choosing the algorithms that will be applied enables good abstractions for a lot of knowledge bases.

But the PST also contains some problems:

- The only algorithm which can be applied incrementally is the shrinking of strong components. Thus we can't use the PST incrementally.
- The PST doesn't influence the inference engine, so a modularisation as described by Clancey [Cla86] is not possible.

These are directions of future work. e

5.3 Acknowledgements

This work was partially supported by the European Community ESPRIT program under contract number P2554 "Machine Learning Toolbox". MOBAL is developed as a team effort of Christian Haider, Jörg-Uwe Kietz, Volker Klingspor, Katharina Morik (head), Edgar Sommer and Stefan Wrobel. The author especially wishes to thank Katharina Morik for important contributions during the design phase of the PST and for reading and commenting earlier drafts of this paper. An extended paper was published in German language as a master thesis at the University of Bonn, Germany [Kli91].

References

[BW89] Joost Breuker and Bob Wielinga. Models of expertise in knowledge acquisition. In G. Gueda and C. Tasso, editors, *Topics in Experts System Desing, Methodologies and Tools*, pages 265 – 295. North-Holland, Holland, 1989.

[Cla86] William J. Clancey. From GUIDON to NEOMYCIN and HERACLES in twenty short lessons: ONR final report 1979-1985. *The AI Magazine*, 7(3):40 – 60, August 1986.

[Hai90] Christian Haider. The MOBAL knowledge displayer (MLT internal memo). GMD (German Natl. Research Center for Computer Science), P.O.Box 1240, W-5205 St. Augustin 1, Germany, 1990.

[Kar88] Werner Karbach. Methoden und Techniken des Knowledge Engineering. Arbeitspapiere der GMD Nr. 338, German Natl. Research Center for Computer Science, P.O.Box 1240, W-5205 St. Augustin, 1988.

[Kie88] Jörg-Uwe Kietz. Incremental and reversible acquisition of taxonomies. *Proceedings of EKAW-88*, pages 24.1 – 24.11, 1988. Also as KIT-Report 66, Technical University Berlin.

[Kie90] Joerg-Uwe Kietz. MOBAL's Program Interface. Technical report, Machine Learning Toolbox ESPRIT Project P2154, October 1990. German Natl. Research Center for Computer Science, P.O.Box 1240, W-5205 St. Augustin.

[Kli91] Volker Klingspor. Abstraktion von Inferenzstrukturen in MOBAL. Master's thesis, Univ. Bonn, 1991.

[KW92] Jörg-Uwe Kietz and Stefan Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In Stephen Muggleton, editor, *Inductive Logic Programming*, chapter 16, pages 335 – 360. Academic Press, London, 1992. Also available as Arbeitspapiere der GMD No. 503, 1991.

[Law76] Eugene L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehard and Winston, 1976.

[LMT87] A. Ludwig, W. Mellis, and L. Thomas. IKEE - an integrated knowledge engineering environment for rule-base development. In *Proceedings of the seventh International Conference on Expert Systems and their Applications*, Avignon, 1987.

[Mor89] Katharina Morik. Sloppy modeling. In Katharina Morik, editor, *Knowledge Representation and Organization in Machine Learning*, pages 107–134. Springer Verlag, Berlin, New York, Jan. 1989.

[MWKE93] K. Morik, S. Wrobel, J.-U. Kietz, and W. Emde. *Knowledge Acquisition and Machine Learning - Theory, Methods, and Applications.* Academic Press, London, 1993.

[Nil82] Nils J. Nilsson. *Principles of artificial intelligence.* Springer, Berlin, 1982.

[Sed88] Sedgewick. *Algorithms.* Addison-Wesley, 2 edition, 1988.

[WB86] Bob Wielinga and Joost Breuker. Models of expertise. In *Proc. ECAI-86*, pages 306 – 318, 1986.

[Wro88] Stefan Wrobel. Design goals for sloppy modeling systems. *Intern. Journal of Man-Machine Studies*, 29:461 – 477, 1988. Also appeared in The Foundations of Knowledge Acquisition, vol. 4, J. Boose and B. Gaines, eds., Academic Press, 1990.