Formal Concept Analysis Methods for Dynamic Conceptual Graphs

Bernhard Ganter and Sebastian Rudolph*

Institute of Algebra
Department of Mathematics and Natural Sciences
Dresden University of Technology
Germany

Abstract Conceptual Graphs (CG), originally developed for static data representation have been extended to cope with dynamical aspects.
This paper adresses two questions connected with the topic: How can implicational knowledge about a system's states and behaviour be derived from a dynamic CG description and how can the CG specification process be supported by automatic or semiautomatic algorithms?
Based on Formal Concept Analysis (FCA) we propose methods for both problems. Guided by an example we introduce two kinds of formal contexts containig the dynamic system's information: state contexts and action contexts. From these the complete implicational knowledge can be derived. Combining the techniques of attribute exploration and determination of a formal context's concepts, we demonstrate a procedure which interactively asks for the validity of implications and from this information designs a dynamic CG system with the desired properties.

1 Introduction

Reasoning about actions and planning is a central topic in AI. In the field of robotics information about environment changes as the result of executeable actions has to be aquired and processed. Often storing and handling such information in a conceptual way appears to be useful [3]. Reasoning about actions also plays an important role for specification of dynamic systems and the generation of operational models.
In recent years efforts have been made to extend the theory of conceptual graphs to dynamic aspects. The intention was not only to describe change of facts with CGs, but in a certain way to simulate such change in a CG-based system. The present paper establishes a connection between the theory of processes in CG on the one hand and methods of analyzing and exploring data from FCA on the other. It proposes a contextual representation of dynamic knowledge which supports modeling of dynamic systems and enables reasoning about situations and transitions.
In Section 2 we give a short review of CG literature relevant to this work. Section 3 comes with a rather general notion of a dynamic system on which our

* Supported by DFG/Graduiertenkolleg.

methods are based and introduces an example. How such kinds of dynamic systems can be translated into formal contexts is demonstrated in Section 4. Finally, using another example, Section 5 shows an algorithm which generates a dynamic CG description in dialogue with the user.

2 Literature

In [7], Sowa proposed *actors*, a new kind of conceptual graph nodes describing functional relations.
The CG formalism including actors covers the whole range of dealing with static knowledge. However, the description, modeling, and execution of dynamic processes is beyond its intended purpose. Therefore Delugach proposed an extension of the theory in [4] by introducing *demons*. These represent processes triggered by the existence of certain CGs (which take the role of preconditions). They act by assertion and retraction of CG concepts. Graphically they are represented by a double-lined diamond box.
Mineau further extended this approach. He allowed demons to have whole CGs as input and output. Furthermore he described processes by pre- and postcondition pairs in [6]. He showed the possibility of translating processes into sets of elementary transitions and demonstrated how an extended CG formalism could be used to represent executable processes.

3 Labeled Transition Systems with Attributes

First we give a rather general notion of a dynamic system. At any time the system is assumed to be in a certain state. Processes can be described as transitions from one of its states to another. Thereby certain more or less observable state properties may change. In order to describe such a system formally, one can record all its possible states including their properties and additionally all possible actions as well as the transitions they cause.

Definition 1. *A* Labeled Transition System with Attributes *(short: LTSA) $\mathbb{T}$ is a 5-tuple (S, M, I, A, T) with*

- *S being a set of system states,*
- *M being a set of state attributes,*
- *$I \subseteq S \times M$ being a relation, where sIm means 'state s has attribute m',*
- *A being a set of actions, and*
- *$T \subseteq (S \times A \times S)$ being a set of transitions, where $(s_1, a, s_2) \in T$ means 'action a can cause the transition from state s_1 to state s_2'.*

Note that the definition of LTSAs is an extension of the classical notion of abstract automata. In both cases we have sets of states, actions and transitions specifying the system's dynamic behaviour.
Consider a simple example: a traffic light system at a crossroad. Altogether let

$$\begin{tikzcd}
\boxed{s_7 : \text{A.red, B.yellow}} \arrow[r] & \boxed{s_0 : \text{A.red, B.red}} \arrow[r] & \boxed{s_1 : \text{A.yellow,B.red}} \arrow[d] \\
\boxed{s_6 : \text{A.red, B.green}} \arrow[u] & & \boxed{s_2 : \text{A.green, B.red}} \arrow[d] \\
\boxed{s_5 : \text{A.red, B.yellow}} \arrow[u] & \boxed{s_4 : \text{A.red, B.red}} \arrow[l] & \boxed{s_3 : \text{A.yellow,B.red}} \arrow[l]
\end{tikzcd}$$

Figure 1. A state-transition graph for the traffic light scenario

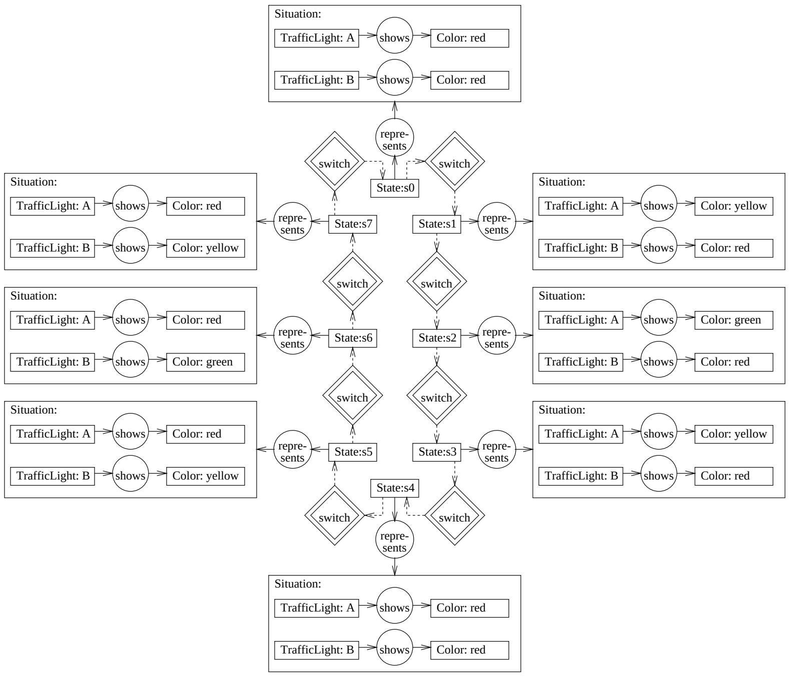

Figure 2. The CG version of the traffic light scenario's state description.

there be four traffic lights. Let A denote one pair of opposite traffic lights and B the other. In Figure 1 a graphical representation of the corresponding LTSA is shown. The possible states are drawn as boxes including their attributes. The arrows between the boxes show the transitions. Since in this case we have only one action, say *switch*, we omit the arc labelling. The system behaviour expressed by this graph can equivalently be described by a conceptual graph using the demons introduced by Delugach. The corresponding CG can be seen in Figure 2. In essence, this is a CG version of the state graph since the demons act by assertion and retraction of whole states. However, this kind of dynamic knowledge representation may be incovenient in some cases for several reasons. At first, the real structure of the system (i.e. its components) is not or only insufficiently apparent. Another (maybe even more evident) flaw appears if more complicated structures are considered. Then the state space may increase exponentially to a size which is difficult to handle computationally and impossible to represent to

people in an understandable way even in form of CGs.
To avoid this difficulty, another approach of representing dynamic systems can be used: Petri nets. (A Petri net is a directed graph with two kinds of nodes: *places* drawn as circles and *transitions* represented by quadratic boxes. Each place can be empty or occupied by a *token*. The dynamic behaviour is defined as follows: a transition may happen if all places with edges towards this transition are occupied by tokens. As a result, all these tokens are removed and new tokens are put at that places, to which an edge leads from the transition. For details see e.g. [2] or [8]. Petri nets have proved useful in the theory of distributed systems.)
In Figure 3 a Petri net representation of our example is shown. Eventually, also

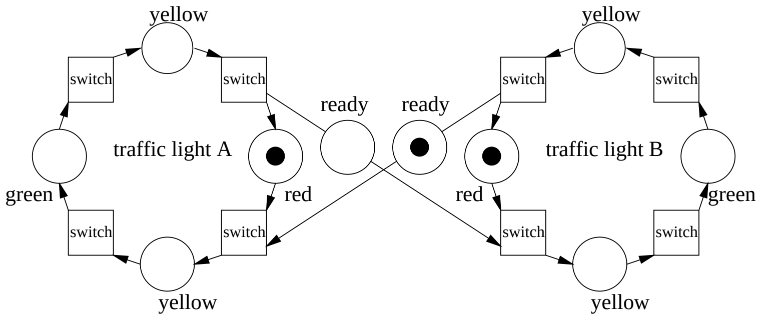

Figure 3. Petri net for the traffic light scenario.

this kind of representation can be transformed into a CG with demons. However, there is no more the possibility to identify one certain state except by the state attributes. So one has to make sure that all distinct token places in the Petri net are distinguished in the CG, otherwise the system behaviour is not completely specified. In our case, the both 'yellow'-places have to be distinguishably specified for both traffic lights, otherwise for example the successing state of s_7 in Figure 1 is not determined (it might be either s_8 as intended or s_6), which contradicts our purpose to specify the system behaviour exactly.
In general, a notion dealing with the possibility of behaviour specification by attributes is the following.

Definition 2. *A LTSA is called* ATTRIBUTE SPECIFIED, *iff for every four states* $s_1, s_2, s_3, s_4 \in S$ *the following holds:*

$$\begin{aligned} &\forall m \in M : (s_1 I m \Leftrightarrow s_2 I m \wedge s_3 I m \Leftrightarrow s_4 I m) \\ &\quad \Longrightarrow \forall a \in A : ((s_1, a, s_3) \in T \Leftrightarrow (s_2, a, s_4) \in T) \end{aligned}$$

This definition says that in an attribute specified LTSA two states 'behave' in the same way (according to the actions), if they have the same properties (according to the attributes). Conversely, this means, if two states 'behave' differently, they must be distinguishable by some attribute.
In order to avoid the unwanted indeterminism mentioned above, we have to

transform the described system (with the token places - or attributes - *red, yellow, green* and *ready* for A and B respectively) into an attribute specified one. This can be done by introducing two additional token places (*beforegreen* and *aftergreen*) for each traffic light. Now we are able to 'translate' the Petri net into a CG with demons. The result is shown in Figure 4.

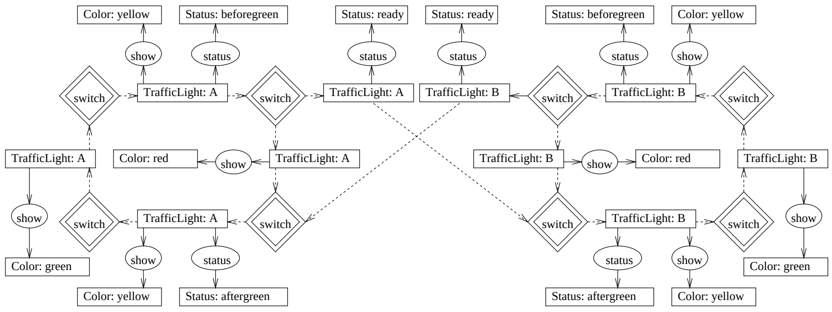

Figure 4. New CG for the traffic light scenario.

4 Introducing Formal Contexts and Deriving Information

Our purpose was to find an alternative description of the knowledge contained in the CG and Petri net representations, which enables implicational reasoning. We find it natural to apply FCA methods. We assume the reader to be familiar with the basics of FCA, for an introduction and details see e.g. [5].
FCA has mostly been applied to static data. There is some work by K.E. Wolff [9] about dynamic systems, but he considers only observation and description of state sequences without having actions triggered from outside the system. So we looked for another approach.
We intended to transform the dynamic knowledge specified in the preceding section into a contextual view, in order to enable the use of FCA methods of reasoning and data exploring. For this, we introduce two kinds of formal contexts.

4.1 The State Context

The first formal context contains only static information. It describes the system's states and their properties (attributes).

Definition 3. *The formal context* $\mathbb{K}_S := (S, M, I)$ *with the state set* S*, the attribute set* M*, and relation* $I \subseteq S \times M$ *from a LTSA* (S, M, I, A, T) *is called the* STATE CONTEXT *of this LTSA.*

	A.r	*A.y*	*A.g*	*A.bg*	*A.ag*	*A.rdy*	*B.r*	*B.y*	*B.g*	*B.bg*	*B.ag*	*B.rdy*
s_0	×						×					×
s_1		×		×			×					
s_2			×				×					
s_3		×			×		×					
s_4	×					×	×					
s_5	×							×		×		
s_6	×								×			
s_7	×							×			×	

Figure 5. State context of the traffic light example

The state context of our traffic light example is shown in Figure 5. We observe that certain rules hold for all states. E.g. whenever traffic light A shows yellow, traffic light B shows red. Such rules can be expressed as implications like *A.yellow* $\rightarrow$ *B.red*. FCA provides an algorithm to find an implicational base from the state context, containing the complete implicational knowledge of the system states according to the attributes under consideration. That is, all implications holding between these attributes (with regard to the context) can be deduced from this base. In our example it consists of the implications listed in Figure 6. Note that the symmetry between the both traffic lights is also apparent in the implicational base.

B.yellow	$\rightarrow$ *A.red*
A.yellow	$\rightarrow$ *B.red*
B.green	$\rightarrow$ *A.red*
A.green	$\rightarrow$ *B.red*
B.ready	$\rightarrow$ *A.red, B.red*
A.ready	$\rightarrow$ *B.red, A.red*
B.red,B.yellow	$\rightarrow \perp$
A.red,A.yellow	$\rightarrow \perp$
B.red,B.green	$\rightarrow \perp$
A.red,A.green	$\rightarrow \perp$
B.yellow,B.green	$\rightarrow \perp$
A.yellow,A.green	$\rightarrow \perp$
B.beforegreen,B.aftergreen	$\rightarrow \perp$
A.beforegreen,A.aftergreen	$\rightarrow \perp$
A.ready,B.ready	$\rightarrow \perp$

Figure 6. Implicational base of the traffic light state context

4.2 The Action Context Family

Until now we only dealt with static system properties. To involve the dynamic aspect of an LTSA, another kind of context is introduced.

Definition 4. *Given a LTSA (S, M, I, A, T), the* ACTION CONTEXT $\mathbb{K}_a$ *of an action $a \in A$ is the triple $(T_a, M_{A\Omega}, J_a)$ with*

- $T_a := \{(s_1, s_2) \mid (s_1, a, s_2) \in T\}$
- $M_{A\Omega} := M \times \{0, 1\}$

  We ocassionally write m_A for $(m, 0)$ and m_Ω for $(m, 1)$ for all $m \in M$.
- $J_a \subseteq T_a \times M_{A\Omega}$ *being a relation with*

  $$(s_1, s_2) J_a (m, i) :\Leftrightarrow \begin{cases} s_1 I m, & \text{if } i = 0 \\ s_2 I m, & \text{if } i = 1 \end{cases}$$

The set $(\mathbb{K}_a)_{a \in A}$ of all action contexts of a LTSA we call its ACTION CONTEXT FAMILY.

So the action context contains all transitions which can be caused by the action as objects. The attributes of an action context are twice the state attributes: once for the initial and once for the final state of the transition.
In our example, the action context family contains only one element: the *switch*-context. It is displayed in Figure 7. Again it is possible to derive an implicational

	$A.r_A$	$A.y_A$	$A.g_A$	$A.bg_A$	$A.ag_A$	$A.rdy_A$	$B.r_A$	$B.y_A$	$B.g_A$	$B.bg_A$	$B.ag_A$	$B.rdy_A$	$A.r_\Omega$	$A.y_\Omega$	$A.g_\Omega$	$A.bg_\Omega$	$A.ag_\Omega$	$A.rdy_\Omega$	$B.r_\Omega$	$B.y_\Omega$	$B.g_\Omega$	$B.bg_\Omega$	$B.ag_\Omega$	$B.rdy_\Omega$
(s_0, s_1)	×						×					×		×		×			×					
(s_1, s_2)		×		×			×								×				×					
(s_2, s_3)			×				×							×			×		×					
(s_3, s_4)		×			×		×						×					×	×					
(s_4, s_5)	×					×	×						×							×		×		
(s_5, s_6)	×							×		×			×								×			
(s_6, s_7)	×								×				×							×			×	
(s_7, s_0)	×							×			×		×						×					

Figure 7. The *switch*-context of the traffic light example

base from this context. As one can easily see, all implications from the state context hold twice in every action context: since every action leads from one state to another, every implication holding for each state has to hold on the '*A*-side' as well as on the '*Ω*-side' of each transition. The implicational base is shown in Figure 8 omitting the implications already contained in the state context as explained above.
Now we have determined the implicational knowledge of the considered dynamic system. Every implication of this kind holding for the dynamic system can be inferred from the implicational base. Thus it could be used in logical programs for reasoning about the system, maybe for verifying certain system properties. Of course, the implicational base can also be coded into a non-dynamic CG-based system as production rules as described in [1]. Figure 9 shows an example.

$A.ready_A$	$\rightarrow$	$B.beforegreen_\Omega$	$B.ready_A$	$\rightarrow$	$A.beforegreen_\Omega$
$A.ready_\Omega$	$\rightarrow$	$A.aftergreen_A$	$B.ready_\Omega$	$\rightarrow$	$B.aftergreen_A$
$A.red_A, B.red_A, A.red_\Omega$	$\rightarrow$	$A.ready_A$	$B.red_A, A.red_A, B.red_\Omega$	$\rightarrow$	$B.ready_A$
$A.red_A, A.red_\Omega, B.red_\Omega$	$\rightarrow$	$B.ready_\Omega$	$B.red_A, B.red_\Omega, A.red_\Omega$	$\rightarrow$	$A.ready_\Omega$
$A.beforegreen_A$	$\rightarrow$	$A.green_\Omega$	$B.beforegreen_A$	$\rightarrow$	$B.green_\Omega$
$A.beforegreen_\Omega$	$\rightarrow$	$B.ready_A$	$B.beforegreen_\Omega$	$\rightarrow$	$A.ready_A$
$A.aftergreen_A$	$\rightarrow$	$A.ready_\Omega$	$B.aftergreen_A$	$\rightarrow$	$B.ready_\Omega$
$A.aftergreen_\Omega$	$\rightarrow$	$A.green_A$	$B.aftergreen_\Omega$	$\rightarrow$	$B.green_A$
$A.yellow_A$	$\rightarrow$	$B.red_\Omega$	$B.yellow_A$	$\rightarrow$	$A.red_\Omega$
$A.yellow_\Omega$	$\rightarrow$	$B.red_A$	$B.yellow_\Omega$	$\rightarrow$	$A.red_A$
$A.green_A$	$\rightarrow$	$A.aftergreen_\Omega$	$B.green_A$	$\rightarrow$	$B.aftergreen_\Omega$
$A.green_\Omega$	$\rightarrow$	$A.beforegreen_A$	$B.green_\Omega$	$\rightarrow$	$B.beforegreen_A$
$A.yellow_A, A.yellow_\Omega$	$\rightarrow$	$\perp$	$B.yellow_A, B.yellow_\Omega$	$\rightarrow$	$\perp$

Figure 8. Implicational base of the traffic light's *switch*-context.

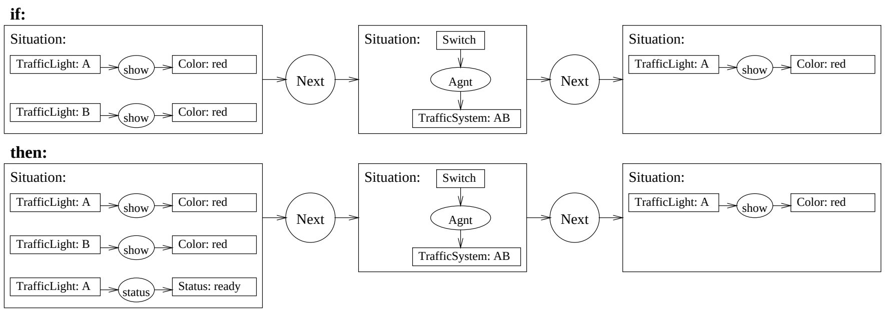

Figure 9. Production rule for one implication from the traffic light implicational base

5 From Implicit Knowledge to an Executable CG-Description

In the preceding section we showed a possibility of transferring the representation of a dynamical system (the state space and the transitions of which are already completely known) into a contextual representation in order to enable reasoning about that system.

In practice, the complete state space and the transitions usually are not explicitly known in advance. Or the system does not yet exist and an engineer wants to design it by specifying the system's static and dynamic properties.

For this case, FCA provides an appropriate tool: attribute exploration. The user just inputs the used state attributes and optionally some states and implications he already knows and then the algorithm asks 'questions' about the system, which the user has to answer.[1] In this way the user successively either makes his implicit knowledge of an existing system explicit or specifies the behaviour of

[1] The questions asked by the algorithm have the following form: 'Does the rule $p \rightarrow q$ hold in the system?' The user either confirms this or enters a counterexample and its attributes.

the system he wants to design. The output of the algorithm is an implicational base of the considered system.
Here the technique will be demonstrated by another example, which is already a bit more complicated. In particular, its state and action contexts are too large to be recorded and displayed completely in a convenient way. Yet the algorithm of attribute exploration is still feasible.

5.1 The Situation

Consider a dynamic system used every day: the telephone. Clearly, everyone has an implicit knowledge of its behaviour and knows how to use it correctly. Now this knowledge has to be expressed explicitly and formally correct by a CG or a Petri net, respectively. This is a task still manageable 'by hand' but already relatively intricate. Let our scope be two telephones, say A and B, which are connected to the outer world. Each of it has several states. First, it may be hung up - in this case it may be ringing or silent. If the phone is picked up, one may hear the dialling tone, the engaged tone, the call signal or one is connected to another telephone. The possible actions which can be performed are: pick up the phone, hang up, and dial a number.

5.2 Stepwise Exploration

To gain the needed information it is useful to start at an elementary level and elevate in the hierarchy of complexity. So the first step is to explore all interesting 'local' static properties of one telephone. This means an exploration involving the attributes *hungup, ringing, silent, pickedup, diallingtone, engagedtone, callsignal* and *connected*. The rule exploration yields the implicational base in Figure 10 as result. This rule base can be seen as set of constraints which ensure the consistency of one telephone's states.

hungup, pickedup	$\to \bot$
hungup, ringing, silent	$\to \bot$
pickedup, diallingtone, engagedtone	$\to \bot$
pickedup, diallingtone, callsignal	$\to \bot$
pickedup, diallingtone, connected	$\to \bot$
pickedup, engagedtone, callsignal	$\to \bot$
pickedup, engagedtone, connected	$\to \bot$
pickedup, callsignal, connected	$\to \bot$
ringing	$\to$ *hungup*
silent	$\to$ *hungup*
diallingtone	$\to$ *pickedup*
engagedtone	$\to$ *pickedup*
callsignal	$\to$ *pickedup*
connected	$\to$ *pickedup*

Figure 10. Implicational base for the exploration of the local static properties

attributes:			
$A.hungup$	$A.callsignal$	$B.hungup$	$B.callsignal$
$A.pickedup$	$A.callingB$	$B.pickedup$	$B.callingA$
$A.ringing$	$A.callingX$	$B.ringing$	$B.callingX$
$A.silent$	$A.connected$	$B.silent$	$B.connected$
$A.diallingtone$	$A.connectedB$	$B.diallingtone$	$B.connectedA$
$A.engagedtone$	$A.connectedX$	$B.engagedtone$	$B.connectedX$

implications: all rules from Figure 10 and:	
$A.callingB$	$\rightarrow A.calling$
$A.callingX$	$\rightarrow A.calling$
$A.callingX, A.callingB$	$\rightarrow \perp$
$A.connectedB$	$\rightarrow A.connected$
$A.connectedX$	$\rightarrow A.connected$
$A.connectedX, A.connectedB$	$\rightarrow \perp$

Figure 11. Starting information for the exploration of the system's static properties

$A.silent, B.callsignal$	$\rightarrow B.callingX$
$A.hungup, B.connected$	$\rightarrow B.connectedX$
$A.diallingtone, B.connected$	$\rightarrow B.connectedX$
$A.engagedtone, B.connected$	$\rightarrow B.connectedX$
$A.callingB$	$\rightarrow B.ringing$
$A.callingX, B.connected$	$\rightarrow B.connectedX$
$A.callsignal, B.pickedup$	$\rightarrow A.callingX$
$A.connectedB$	$\rightarrow B.connectedA$
$A.connectedX, B.connected$	$\rightarrow B.connectedX$

Figure 12. Implicational Base of the system's static properties

The next step is an exploration of the entire system's static properties (including two telephones in our case). For this, we introduce for each of the 'local attributes' mentioned above one attribute for telephone A and one for telephone B. Furthermore, we introduce four more attributes for each telephone, namely *callingB, callingX, connectedB, connectedX* for telephone A and *callingA, callingX, connectedA, connectedX* for telephone B, expressing whether the partner included in our scope or some other member is involved.
The background knowledge we can start with consists of two copies of the implicational base computed above: one for each telephone. We may then add some more implications to the background knowledge, linking the newly introduced attributes to the other ones. Figure 11 shows the set of attributes and the background information used as input for the second step of the exploration process. Furthermore, we can assume the behaviour of the two telephones to be equal and thus define a permutation on the set of attributes which - if applied to an implication - does not change its truth value. This permutation just 'exchanges' the two telephones. Entering this permutation in advance shortens the exploration process considerably.
The second exploration step yields implications concerning the whole system's possible states (again they can be seen as constraints) which are recorded in Figure 12. At the third exploration step the dynamical aspect is introduced. For

	$\rightarrow A.pickedup_A,\ A.hungup_\Omega\ ,\ A.silent_\Omega$
$B.hungup_\Omega$	$\rightarrow B.hungup_A$
$B.hungup_A$	$\rightarrow B.hungup_\Omega$
$B.pickedup_\Omega$	$\rightarrow B.pickedup_A$
$B.pickedup_A$	$\rightarrow B.pickedup_\Omega$
$B.connectedX_\Omega$	$\rightarrow B.connectedX_A$
$B.connectedX_A$	$\rightarrow B.connectedX_\Omega$
$B.callingX_\Omega$	$\rightarrow B.callingX_A$
$B.callingX_A$	$\rightarrow B.callingX_\Omega$
$B.diallingtone_\Omega$	$\rightarrow B.diallingtone_A$
$B.diallingtone_A$	$\rightarrow B.diallingtone_\Omega$
$B.silent_A$	$\rightarrow B.silent_\Omega$
$A.diallingtone_A,\ B.silent_\Omega$	$\rightarrow B.silent_A$
$A.engagedtone_A,\ B.silent_\Omega$	$\rightarrow B.silent_A$
$A.callingX_A,\ B.silent_\Omega$	$\rightarrow B.silent_A$
$A.connectedX_A,\ B.silent_\Omega$	$\rightarrow B.silent_A$
$B.ringing_\Omega$	$\rightarrow B.ringing_A$
$A.diallingtone_A,\ B.ringing_A$	$\rightarrow B.ringing_\Omega$
$A.engagedtone_A,\ B.ringing_A$	$\rightarrow B.ringing_\Omega$
$A.callingX_A,\ B.ringing_A$	$\rightarrow B.ringing_\Omega$
$A.connectedX_A,\ B.ringing_A$	$\rightarrow B.ringing_\Omega$
$B.engagedtone_A$	$\rightarrow B.engagedtone_\Omega$
$A.diallingtone_A,\ B.engagedtone_\Omega$	$\rightarrow B.engagedtone_A$
$A.engagedtone_A,\ B.engagedtone_\Omega$	$\rightarrow B.engagedtone_A$
$A.callingX_A,\ B.engagedtone_\Omega$	$\rightarrow B.engagedtone_A$
$A.connectedX_A,\ B.engagedtone_\Omega$	$\rightarrow B.engagedtone_A$
$A.callingB_A$	$\rightarrow B.silent_\Omega$
$A.callsignal_A B.ringing_A,\ B.ringing_\Omega$	$\rightarrow A.callingX_A$
$A.connectedB_A$	$\rightarrow B.engagedtone_\Omega$
$B.ringing_A,\ B.silent_\Omega$	$\rightarrow A.callingB_A$
$B.connected_A,\ B.engagedtone_\Omega$	$\rightarrow A.connectedB_A$

Figure 13. Implicational Base of the systems dynamic properties concerning action *A.putdown*

each action we want an implicational base concerning the pre- and poststates' attributes. So, like shown before, for each state attribute m we introduce two new attributes m_A and m_Ω, indicating whether the attributes hold before respectively after the action taking place. Naturally, all static property implications can be used as background knowledge twice: for the pre- and for the post-state. The result of this exploration step concerning the action *A.putdown* can be seen in Figure 13. The implicational bases of all action contexts together with the implicational bases concerning the static properties mentioned before represent the implicational knowledge of our dynamic system.

5.3 Transferring the Results into CGs

At the end we have to find a dynamic CG model of our described system by defining appropriate demons. The translation of a state attribute into CG description is intuitively clear - we just create a (non-dynamic) CG, which expresses the corresponding state property. To avoid confusion when considering the assertion and retraction process, we assume that the attributes are translated into distinct CGs. In order to obtain the according demons we can use another FCA technique: determining all concepts of a formal context. In our case the concepts (more exactly: the concept intents) can be derived from the implication set found in the former process. Now for each transition intent (which is a set $N \subseteq M_{A\Omega}$ with $N = \{n \mid tJn\}$ for some $t \in T_a$) we can define a demon in the following way:

Definition 5. *Let $a \in A$ be an action, $M_1, M_2 \subseteq M$ be sets of state attributes and let $\{m_A \mid m \in M_1\} \cup \{m_\Omega \mid m \in M_2\}$ a transition intent of the a-context. The demon corresponding to this transition intent retracts the CGs representing the attributes M_1 and asserts the CGs representing the attributes M_2. It is labeled with a.*

Obviously, the system designed in this way shows the intended and specified behaviour since it contains all possible state property combinations and respects all implications of the underlying implicational base.
However, this solution is still inconvenient for two reasons. First: it is not trivial to determine whether a concept intent is a transition intent because as mentioned before the whole transition set needs not (and is sometimes impossible) to be known. Second: the amount of demons needed to represent one elementary action is quite large.[2] One possibility of coping with both problems is to investigate the invariants.

Definition 6. *Let $a \in A$ be an action, $M_1, M_2 \subseteq M$ be two sets of state attributes and $m \in M \setminus (M_1 \cup M_2)$ a state attribute. We call m* INVARIANT UNDER ACTION $a \in A$ WITH PRECONDITIONS M_1 AND POSTCONDITIONS M_2 *if the two implications $M_{1A}, M_{2\Omega}, m_A \rightarrow m_\Omega$ and $M_{1A}, M_{2\Omega}, m_\Omega \rightarrow m_A$ hold in the a-context.*

With this definition we can express in which case a certain state attribute is not changed by an action. Using this notion of invariants, the number of demons for one action can be reduced and the transition intents need not to be determined:

Proposition 1. *Let $a \in A$ be an action and $M_1, M_2 \subseteq M$ be two sets of state attributes. If all $m \in M \setminus (M_1 \cup M_2)$ are invariant under a with preconditions M_1 and postconditions M_2 then the demon retracting the information in M_1 and asserting the information in M_2 shows the specified behaviour and makes all further demons retracting (at least) M_1 and asserting (at least) M_2 obsolete.*

A demon created in that way does not 'take care' of all invariant attributes and thus may substitute several demons due to Definition 5 which do 'take care'. This proposition facilitates to develope an algorithm which creates demons

[2] In fact one would need as many demons for one action a as there are transitions caused by a.

Demon nr.	retracts	asserts
1	*A.pickedup,A.connected,A.connectedX*	*A.hungup,A.silent*
2	*A.pickedup,A.connected,A.connectedB* *B.pickedup,B.connected,B.connectedA*	*A.hungup,A.silent* *B.pickedup,B.engagedtone*
3	*A.pickedup,A.callsignal,A.callingX*	*A.hungup,A.silent*
4	*A.pickedup,A.callsignal,A.callingB* *B.hungup,B.ringing*	*A.hungup,A.silent* *B.hungup,B.silent*
5	*A.pickedup,A.engagedtone*	*A.hungup,A.silent*
6	*A.pickedup,A.diallingtone*	*A.hungup,A.silent*

Figure 14. List of demons needed to represent the action *A.putdown*

from the transition context. We pass through a lectical ordered list of concepts and check, whether the condition mentioned in the proposition above holds for the concept intent. If this is the case, then a corresponding demon is created and all subconcepts of this concept are deleted from the list (since the list is ordered lectically, all subconcepts of a concept are listed after it and thus no superfluous intent has been passed before). We proceed until the end of the list is reached. Figure 14 shows the list of demons created in this way for the *A.putdown* action. In Figure 15 the first item of the table in Figure 14 is explicitly shown as dynamic CG.

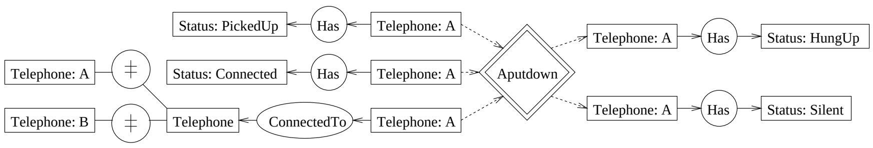

Figure 15. One dynamic CG from the list above

The complete dynamic CG for the telephone scenario would be by far too large to be displayed here. Now we have found a CG description of the dynamic system by means of FCA methods. It can be translated one to one into a petri net as well: we introduce a token place for every attribute, a transition for every demon and draw the directed edges accordingly. When used in a CG-based system, the demons have to be triggered in some way. This could be done by putting an additional retraction edge to each demon, which is linked to a CG expressing, that the corresponding action is initiated.
Note that also the case of indeterministic dynamic systems is covered by our approach.

6 Conclusion

FCA methods prove useful in both extracting processable implicational knowledge from dynamic systems with known state space and designing CG descriptions by stepwise exploration.
There are several questions arising from this work. What inferences can be done

about composite actions? How can the approach be extended to conditional actions and iterations? Given properties of an initial state and properties, which are to be achieved, how can an algorithm be found that efficiently generates a corresponding plan (i.e. a list of actions to be performed)? These questions are objects of ongoing research.
Additionally, the feasibility and efficiency of the proposed techniques have to be evaluated empirically by applying them to more complex scenarios. We expect that increasing complexity will require an elaborated theory as the basis for efficient algorithms.

References

1. J.F. Baget, D. Genest, M.L. Mugnier: A Pure Graph-Based Solution to the SCG-1 Initiative. In: W. Tepfenhart, W. Cyre (eds.): Conceptual Structures: Standards and Practices, LNAI 1640, Springer, Berlin-Heidelberg, pp. 335-376, 1999
2. B. Baumgarten: Petri-Netze. Grundlagen und Anwendungen. Spektrum, Heidelberg, 1996.
3. A. Chella, M. Frixione, S. Gaglio: Towards a Conceptual Representation of Actions. In: E. Lamma, P. Mello (eds.): AI*IA 99, LNAI 1792, Springer, Berlin-Heidelberg, pp. 333-344, 2000
4. H. S. Delugach: Dynamic Assertion and Retraction of Conceptual Graphs. In: Eileen C. Way (ed.): Proc. Sixth Annual Workshop on Conceptual Graphs. SUNY Binghamton, Binghamton, New York, pp. 15-26, 1991.
5. B. Ganter, R. Wille: Formal Concept Analysis: Mathematical Foundations. Springer, Berlin-Heidelberg 1999.
6. G. W. Mineau: From Actors to Processes: The representation of Dynamic Knowledge Using Conceptual Graphs. In: M.L. Mugnier, M. Chein (eds.): Conceptual Structures: Theory, Tools and Application, LNAI 1453, Springer, Berlin-Heidelberg, pp. 198-208, 1998.
7. J. F. Sowa: Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley, Reading, MA, 1984.
8. P. H. Starke: Analyse von Petri-Netz-Modellen. Teubner, Stuttgart, 1990.
9. K. E. Wolff: Towards a Conceptual System Theory. In: Proc. Third International Conference on Computing Anticipatory Systems. American Institute of Physics, 2000.