Coping with Unconsidered Context of Formalized Knowledge

Stefan Mandl and Bernd Ludwig

Lehrstuhl für Künstliche Intelligenz
Friedrich-Alexander-Universität Erlangen-Nürnberg
{`Stefan.Mandl, Bernd.Ludwig`}`@informatik.uni-erlangen.de`

Abstract. The paper focuses on a difficult problem when formalizing knowledge: What about the possible concepts that didn't make it into the formalization? We call such concepts the *unconsidered context* of the formalized knowledge and argue that erroneous and inadequate behavior of systems based on formalized knowledge can be attributed to different states of the unconsidered context; either while formalizing or during application of the formalization. We then propose an automatic strategy to identify different states of unconsidered context inside a given formalization and to classify which parts of the formalization to use in a given application situation. The goal of this work is to uncover unconsidered context by observing sucess and failure of a given system in use. The paper closes with the evaluation of the proposed procedures in an error diagnosis scenario featuring a plan based user interface.

1 Concerning the Unconsidered when Formalizing Knowledge

When formalizing informal knowledge about a problem domain, certain activities have to be performed. On the *Knowledge Management* (KM) side, the yet informal knowledge which is relevant for the problem at hand has to be identified and made accessible. *Knowledge Engineering* (KE) activities prepare the informal knowledge in such a way that it allows for a formal representation. The following activities are listed in [14]:

1. Learn the terminology of the problem domain
2. Cut down on the coverage of the knowledge to be formalized
3. Select representation formalism
4. Create conceptualization
5. Write down the contents using the terminology

Finally the knowledge has to be encoded as a formal system which is expressed in a *Knowledge Representation* (KR) language. In this paper, we restrict our discussions to formalized knowledge which is represented in a logical language – for the experiments, we use Answer-Set-Programming (ASP). As implementation of ASP, we use the Smodels System (see [13] and [18]).

At each level of activities – the KM-, KE-, and KR-levels – things are left behind:

- On the KM-Level, knowledge that is not considered important is unconsidered. Furthermore, if the relevant knowledge either cannot be found or is not accessible, it has to be left behind. One typical reason for knowledge not being accessible is the fact that it is often people (experts) who carry the knowledge and hence, when there are other urgent tasks, there is no time to acquire the knowledge from the expert.
- On the KE-Level, the use of a strict terminology and conceptualization allows to focus on the seemingly important concepts to be formalized, but in turn leaving everything behind that is not put into the terminology or modeled in the conceptualization.
- On the KR-Level, statements that cannot be expressed in a certain Knowledge-Representation language have to be approximated with other statements – leaving behind the original statement with the original meaning.

Thus, there is no doubt that no formalization is complete in the sense that everything that could have been said about the problem domain is being considered and included in the formalization. As Genesereth & Nilsson [3] put it:

> *Language (probably* any *language) cannot capture all that we want to say about the world.*

Every formlization takes place in a certain context, those parts of the context that are not put into the formalization make up the *unconsidered context* of the formalization.

Unconsidered context has the following properties:

- *Informal* – as the unconsidered parts are not covered by the conceptualization, there is no way to associate formal sentences with them; they have to remain informal.
- *Infinite* – every entity involved in the formalization introduced new backgrounds and perspectives. Even a single individual can easily come up with different opinions about the same subject.
- *Dynamic* – Being unconsidered does not mean that these things do not change; the real world is changing beyond the formalization.

Obviously the dynamics of the unconsidered context are a major problem. If the unconsidered context did not change, there would be little harm. But when it changes, it can influence the correctness of the considered parts of the formalization.

This circumstance has been recognized in the field of Planning. In [11], McCarthy explains the so called *qualification problem* as follows:

> *It seemed that in order to fully represent the conditions for the successful performance of an action, an impractical and implausible number of qualifications would have to be included in the sentences expressing them. For example, the successful use of a boat to cross a river requires, if the boat is a rowboat, that the oars and rowlocks be present and unbroken, and that they fit each other. Many other qualifications can be*

added, making the rules for using a rowboat almost impossible to apply, and yet anyone will still be able to think of additional requirements not yet stated.

All the qualifications that could have been added are unconsidered context. When the unconsidered context is changing, it can affect both, creating and using the formalized knowledge.

At *system creation time*, different setups of unconsidered context can lead to an inconsistent theory. Such a situation can easily occur when different people with different backgrounds and presuppositions work together on a project. They may agree on the surface-level but when it comes to writing down the knowledge, differences begin to show.

At *runtime*, changing unconsidered context has the potential to cause the formalized knowledge to be plainly wrong!

2 Previous Work

There is surprisingly little prior work on unconsidered context but as the problem is omnipresent when building intelligent systems, there is a lot of work motivated by problems that – we think – could be interpreted as problems actually caused by unconsidered context.

In Machine Learning, there is the problem of 'concept drift in online concept learning' (see for examples [7], or [19]), where, the concept to be learned appears to be changing over the course of time. Such behavior can actually be understood as *hidden context changes* (see [6] or [20]). Hidden context and unconsidered context are similar concepts. While *hidden context* is invisible from the formalization's point of view, *unconsidered context* was not put into the formalization beforehand – we use the term *unconsidered* in order to emphasize the responsibility of the knowledge engineer. Thus, some of the techniques developed for handling concept drift can be adapted to handle unconsidered context; in our work, we try to deal with a more general setting: 1) the proposed techniques are not restricted to online learning and 2) we try to formalize the previously unconsidered and therefore make it possible to reason about unconsidered contexts.

In the mid-1980s Rodney Brooks proposed to build *intelligent systems that contain no representation at all* ([1]) – this is maybe the most forceful reaction to the problems of formalizing knowledge. We do not know if unconsidered context was known to Brooks and explicitly influenced his ideas, but we do know that it causes a lot of problems when building knowledge based systems and henceforth assume that it influenced Brooks at least to some degree.

The field of *context aware computing* (see [17]) is born out of the idea that one has to add more and more sensors in order to improve system performance. Thus, advocates of context aware computing try to reduce the unconsidered context of systems. The major problem with this approach is: there is no real end to it. On the other hand, if failure is not critical and the system approaches close to 100% performance, just making some spurious mistakes in some obscur situations, much is being gained.

In order to increase modularity when building large knowledge bases (like the one described in [8]), *formal context logic* was created. Interestingly, no definition of 'context' is provided:

> *"Contexts are abstract objects. We don't offer a definition, but we will offer some examples."* [12]

Thus, in formal context logic, context is represented *inside* the system as a formal structure while the unconsidered context is located *outside* the system. As (see the next section) our intent is indeed to find a way to make the unconsidered context accessible inside the system (with hindsight, when the system is actually used), formal context logic may serve as a way to represent unconsidered context, but as the context structures used in our system are quite simplistic, formal context logic would have been overkill; we use a much more hands-on approach, at least for the time being.

If something is not unconsidered but seems to be unimportant, it would be nice if we wouldn't have to be concerned with it. There are some subfield that deal with default assumptions, namely Frames and Non-monotonic Reasoning (see [16]). Defaults are related to normality which in turn is related to context.

Belief-Change/Revision (see [2]) and Knowledge Base Refinement (see [4]) deal with different ways of modifying a given theory or knowledge base. The following chapter reveals our plans to follow their footsteps.

3 Coping with Unconsidered Context

3.1 Three Basic Strategies

As mentioned at the end of section 1, the concept of unconsidered context allows for a common understanding of different types of errors, thus techniques for dealing with unconsidered context provide the potential to improve a given erroneous theory. We have found three basic strategies in order to deal with theories that are influenced by the effects of unconsidered context:

1. The most obvious strategy simply is to *formalize again*. Insights that have been gained in the first iteration now help to avoid unfortunate constructions. Typically, in hindsight it is much easier to determine what should have been put into the formalization such that the required tasks can be accomplished.
2. In order to avoid the efforts of a re-formalization (both monetary and time), it is possible to *document the assumptions* the system makes with respect to the unconsidered context. Now the user has to decide whether she should trust answers given by the system.
3. The most ambitious alternative is the *automatic correction* of the given system. As shown in figure 1, from a given set of axioms a set of subsets is created, yielding a set of new systems. In the last step, a classifier function is created which selects the appropriate sub-system for a given query.

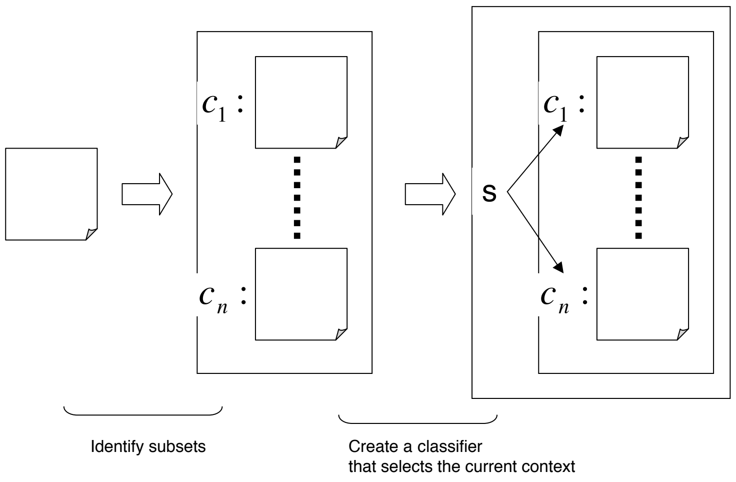


Fig. 1. Automatic correction overview

Besides the usefulness in itself, automatic correction is also interesting because the two other approaches – re-formalization and documentation – obviously also benefit from an auto-correcting system:

- Auto-correction – by delivering reasonable subsets – provides (indirect) hints on which parameters may be worth to consider for a re-formalization.
- By comparing the identified subsets, it should be possible to determine which queries are safe in the sense that every context answers them in the same way. Then one could provide documentation about the queries that are safe to ask.

Hence, automatic correction is a goal worthy to achieve, whether or not actually delivered systems are to be supplied with an auto-correcting component.

3.2 Identification and Classification

Identification and *classification* of unconsidered context are the two main activities that allow automatic correction. An identification function is mapping from a formalization W to a subset of the set of all subsets 2^{2^W} of W. Thus, having n Axioms in a declarative theory, there are 2^{2^n} possible selections, for a set of $n = 6$ elements, there are $18,446,744,073,709,551,616$ subsets of the set of all subsets. Hence, there is no simple generate-and-test approach for identification functions. Therefore, we rely heavily on heuristics and external assumptions about the structure and behavior of the unconsidered context.

In [10] we analyze an online context learning problem. The assumptions are that 1) the sequence of examples adheres to observations made in reality, thus the unconsidered context can only change in ways that are possible in reality.

2) We assumed that every significant different state of the unconsidered context reveals itself by causing different probabilities that certain features show up in observations. In the present paper, we deal with the case when the formalization is given as a set of axioms for a logical theory. Hence, there is no set or sequence of examples. In fact, we actually do require a sequence of test examples – queries with correct answers – for our identification procedure, but not for the given theory where the unconsidered contexts are to be identified.

3.3 An Evolutionary Approach to Identification and Classification

As picture 1 shows, finding the subsets and creating the classifier are conceptually distinct phases. Identification is actually an optimization problem.

When given an objective function the measures the quality of a set of subsets of the consituents of a formalization, we could use a standard genetic algorithm (see [5]) to find close to optimal solutions. The crucial part is the objective function. It could feature various heuristics like:

– The subsets should be distinct.
– The subsets should give different answers on as many queries as possible.
– The subsets should be correct on a given test sequence as long as possible before making an error.

All those heuristics are afflicted by the problem that they are ad-hoc; one does not know when to use them or not. A better founded heuristic function is presented in the next subsection.

3.4 Combining Identification and Classification

The general idea can be captured with the slogan: *There is no reason to identify contexts that cannot be classified later.* We have to understand that identification is one component of a larger system, no matter how good the results of the identification are with respect to some objective function, if the rest of the system cannot facilitate the results of identification, the system still performs weakly.

Hence we propose to take into account the performance of *the complete system*, comprising identification and classification as an objective function for identification. In other words: those identifications are set to have high (in our GA, higher means fitter) fitness, which perform well in conjunction with a given prediction function on a test set.

3.5 Do We Really Identify Unconsidered Context?

Even if the system performance increases, why should we belief that we found different states of unconsidered context? The answer is quite straightforward: If we have a set of subsystems and a way to select subsystems for queries such that the performance increases, then we have adapted the system to things that change outside the system, hence unconsidered context. We are not able to identify *specific* setups of the unconsidered context; we hope to be able to identify *equivalent* setups, the (informal) equivalence relation reads: 'makes the same subset of the given set of axioms a good description of the state of affairs.'

3.6 A Classification Function based on Actions

If the domain has the concept of *actions* then – because actions actively change the world – it is an obvious choice to base the fitness function on sequences of actions. Given a sequence of $n+1$ actions $a^0, \ldots, a^n$, the *current context* $c_i^n$is to be determined. This is a task suitable for the well-known Hidden-Markov-Models (HMMs; see [15]).

In order to use a HMM to determine the current context given a sequence of actions, we need to define what kind of *observations* are presented to the HMM. HMMs only deal with observations not actions. In order to overcome this limitation, from the set of all possible actions $\mathcal{A}$ and all possible contexts $\mathcal{C}$, we create the set of observations

$$\mathcal{O} = \mathcal{A} \times \mathcal{C}$$

In the implementation, we use integers to represent members of $\mathcal{O}$ which in turn represent combinations of actions and identified context; hence, we assume that the sets of actions and actions are finite.

The currently active context given an action and a sequence of previous observations is then chosen by comparing the likelyhood – as computed with the HMM – of all possible extensions of the given sequence with observations that contain the given action and choosing the most likely extension, thus obtaining the context.

4 The Coffee-Shop-Logistics (CSL) Diagnosis Domain

In order to evaluate the proposed techniques, we are in an uncomfortable position. On the one hand, we like to evaluate how our techniques perform when unconsidered, invisible parameters are changing; on the other hand, we like to perform a controlled experiment – we want to be sure that improvements do not happen by chance and changes in unconsidered context actually do happen during the evaluation. Therefore we decided to proceed as follows: 1) Create a formal domain (full), 2) decide which concepts should be unconsidered (small). 3) Create examples in the full domain and 4) map them to the small one by eliminating all unconsidered concepts.

The evaluation task is to do *error diagnosis*.

The domain used here is similar to the one described in [9] . There is a coffee-dispenser, a Lego$^{\text{TM}}$-Robot and a model train.

A typical history of events is the robot putting a cup under the spout of the coffee-dispenser, the coffee-dispenser filling the cup, the robot putting the cup on the wagon of the train, and the train delivering the cup to some user (see figure 4). The actual actions performed in the domain are not encoded as fixed procedures; instead, there is a model describing the state of the domain. When the user utters a goal like

"A small cup of coffee please"

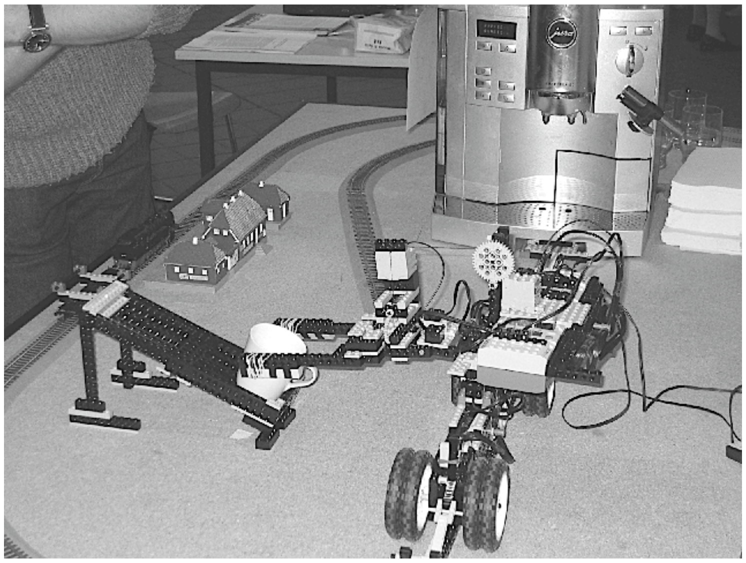


Fig. 2. The real domain which influenced the creation of the diagnosis domain. The robot (center) takes a cup from the ramp, then it will bring it to the coffee dispenser, when the cup is filled, the robot is to put the cup on the train.

the speech signal is (hopefully) and transformed into a formal goal description like:

```
(:goal (and (delivered cup)
            (mode-osc jura)))
```

which is a goal-state description in the planning language PDDL. Then, a planning system creates a plan – a sequence of actions – that when successfully executed will achieve the goal state. The involved speech recognition, natural language understanding and dialog managing technologies are not described in this paper; we assume that somehow a formal PDDL goal description is available. After each plan stepp, the world state is transformed into a list of logical facts. Diagnosis is performed by concatenating this list with another list of axioms suitable for diagnosis and using a model checker to find possible diagnosis. Thus, the system's performance can be evaluated by presenting situations and comparing the system's diagnoses with the true diagnoses. These are obtained from a correct diagnosis theory that operates on the 'full' domain (see page 7).

In order to introduce unconsidered context, we mark some concepts as invisible. Those are still used when planning and creating the diagnosis system to be optimized and when creating the evaluation examples, but they are removed before evaluating the identification and classifications procedure ('small'; see page 7). In order to evaluate the proposed the runtime data is created by a simulator which generates random goals, executes plans, performs diagnosis and records the results in example sequences.

4.1 Modeling the Domain

In this section, we show how the CSL domain was modeled. Table 1 lists the visible (considered) concepts. Please note that actions are also considered as con-

`cup`	a cup	`produce-coffee`	action (with obvious meaning)
`jura`	the coffee dispenser	`draw-off-osc`	action: produce one small cup
`robo`	the robot	`draw-off-tsc`	action: produce two small cups
`train`	the train	`draw-off-obc`	action: produce one big cup
`draw-off-tbc`	action: produce two big cups	`take-cup-off-spout`	action: (with obvious meaning)
`put-cup-on-spout`	action: (with obvious meaning)	`load-cup-on-waggon`	action: (with obvious meaning)
`deliver-cup`	action: (with obvious meaning)	`go-in-place`	action: train moves to special location

Table 1. Considered concepts in the CSL domain

cepts in the domain. Those are used because the *previous* action can be taken into account when defining diagnosis rules. Furthermore, we need some small numbers which we represent by using the symbolic constants `n0`, `n1`, ..., `n8`. Table 2 lists all the visible (considered) predicates used in the system. Table 3

under-spout `?c`	true when a cup `?c` is under the spout
ready `?x`	true when coffee dispenser `?x` is ready to produce coffee
service-request `?j`	true when the self diagnosis of coffee dispenser `?j` found an error
empty `?c`	is cup `?c` empty?
mode-osc `?j`	is coffee dispenser `?j` in mode 'one small cup'?
mode-tsc `?j`	is coffee dispenser `?j` in mode 'two small cups'?
mode-obc `?j`	is coffee dispenser `?j` in mode 'one big cup'?
mode-tbc `?j`	is coffee dispenser `?j` in mode 'two big cups'?
robo-loaded `?r`	true when robot `?r` is carrying cup `?c`
train-loaded `?t`	true when cup `?c` is loaded on train `?t`
parked `?c`	true when cup `?c` is in parking position
delivered `?c`	true when cup `?c` has been delivered
in-place `?t`	true when train `?t` stands ready
action `?a`	true if `?a` is the name of the last action performed (see table 1)

Table 2. Visible predicates in the CSL domain

shows the invisible (unconsidered) predicates. Even when we explicitly provide the diagnosis rules that contain unconsidered concepts, we hesitate from providing a set of diagnosis rules containing only visible concepts. Instead, we use a simulator to create examples that only contain visible concepts. We then use those examples as diagnosis rules. As a matter of fact, such rules are very bad, no generalization is applied. Still, as the number of concepts and predicates in the CLS-domain is very small, we hope to cover enough cases to get reasonable results. The next subsection provides more details on how the visible diagnosis rules are created via the simulator.

4.2 Using a Simulator to Generate 'Visible' Diagnosis Rules and Evaluation Examples

In order to create diagnosis rules and evaluation examples, we need a corpus of examples. We produce one by following the procedure outline below. As we want to generate data without user interaction, we have to provide additional healing actions that can be taken if a diagnosis indicates a problem. Thus, we are assuming that every diagnosed error can be corrected. The procedure to generate examples contains the following steps:

- randomly select one of a list of possible goals,
- given the current state of the system, create a plan that will achieve the goal,
- execute the plan (action by action),
- after each action, apply the full set of diagnosis rules ,
- record the pair

  $$\langle\langle \alpha_1, \ldots, \alpha_n, \text{not}\, \alpha_{n+1}, \ldots, \text{not}\, \alpha_{n+m}\rangle, \langle \mathit{diagnosis}_1(X_1), \ldots, \mathit{diagnosis}_k(X_k)\rangle\rangle$$

  but remove any literal that should be unconsidered!
- apply every possible healing actions (we assume that healing actions do not conflict!),
- continue until goal state is reached.

The procedure outlined above is iterated several times in order to create a large body of tuples. Those tuples are used as diagnosis rules and for evaluation. Figure 3 shows the diagnosis rules and therapy rules containing unconsidered parameters. A small fraction of the diagnosis rules obtained in this way (only those concerned with the diagnosis `no_more_coffee`) is shown in figure 4 (note that the diagnosis rules created in this way are not unique which also can be seen in the figure). The unconsidered parameters are used only inside the planning system, the plan executor and the diagnoser – all outputs are deprived of the unconsidered concepts. Thus, we achieve the desired effect: something relevant changes beyond the (small) representation.

water-left `?n`	`?n` is a measure for the amount of water left
coffee-left `?n`	`?n` is a measure for the amount of beans coffee beans left
cups-left `?n`	`?n` is the number of cups left

Table 3. Unconsidered predicates in the CSL-domain

```
action(deliver_cup) :- diagnosis(no_cup_available).
water_left(n0) :- diagnosis(no_more_water).
coffee_left(n0) :- diagnosis(no_more_coffee).
cups_left(n0) :- diagnosis(no_more_cups_left).

healing(refill_water) :- diagnosis(no_more_water).
healing(refill_coffee) :- diagnosis(no_more_coffee).
healing(bring_new_cup) :- diagnosis(no_cup_available).
healing(provide_new_cups) :- diagnosis(no_more_cups_left).
```

Fig. 3. Diagnosis rules with unconsidered concepts and selection of the appropriate healing actions for a given diagnosis

To perform the diagnosis, in addition to literals describing the current state and the diagnosis rules, we provide some background knowledge about the domain that restricts the created models to the actually possible cases. We have background knowledge for concepts that cannot be in a model at the same time and rules that forbid models with no diagnoses at all.

Figure 5 illustrates how visible diagnosis rules and evaluation examples are generated.

In order to evaluate the system, we create a sequence of evaluation examples in the same way as for the diagnosis rules; the major difference is that we keep the examples in sequence and do not remove double entries. Correctness per test t_i is defined to be the fraction of missing and wrong diagnoses, hence given system answers D_{t_i} diagnoses B_{t_i} which are known to be correct, the correctness is

$$\mathrm{Corr}(D_{t_i}|B_{t_i}) = \frac{|B_{t_i} \cap D_{t_i}|}{|B_{t_i} \cup D_{t_i}|}$$

```
diagnosis(no_more_coffee) :- mode_tsc(jura), in_place(train),
                             mode_osc(jura), under_spout(cup),
                             action(produce_coffee), ready(jura).
no_diagnosis(no_more_coffee) :- mode_tsc(jura), in_place(train),
                                mode_osc(jura), ready(jura),
                                action(load_cup_on_waggon),
                                train_loaded(train, cup).
diagnosis(no_more_coffee) :- mode_tsc(jura), in_place(train),
                             mode_osc(jura), under_spout(cup),
                             action(produce_coffee), ready(jura).
```

Fig. 4. Some of the simulator created diagnosis rules, reduced to visible concepts

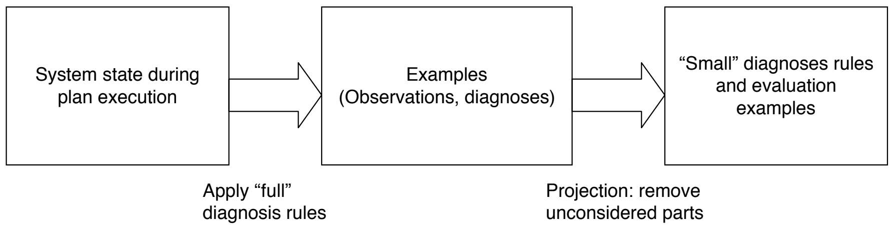

System state during plan execution
Examples (Observations, diagnoses)
"Small" diagnoses rules and evaluation examples
Apply "full" diagnosis rules
Projection: remove unconsidered parts

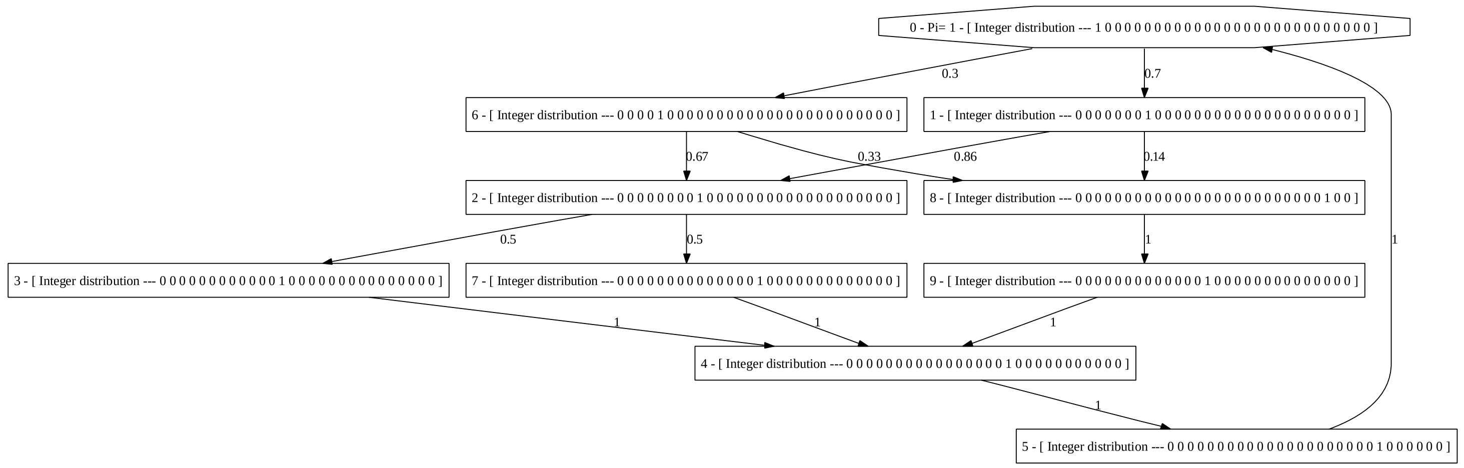
Fig. 5. High level view of visible rule and example generation

5 Results and Discussion

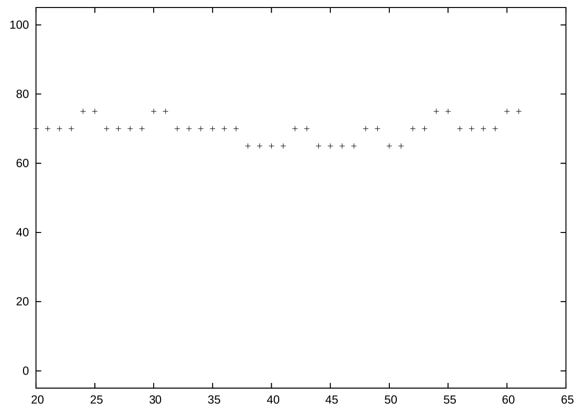
We discuss why systems based on formalized knowledge are inherently incomplete and can only be used under certain assumptions. We introduce the concept of 'unconsidered context' and argue that most of the failures of systems using formalized knowledge can be understood as being related to unconsidered context. We propose a high-level procedure to automatically correct a given formalization by using feedback obtained at the runtime of the system. On the technical level, we are using genetic algorithms and Hidden Markov Models for this task, but this is just one option!

The performance in the Coffee-Shop-Logistics domain, which is specifically created to allow for controlled experiments with unconsidered quantities, increases from 68% to 92% correctness. Figure 6 shows the created HMM and the smoothed correctness curve (using a smoothing window of size 10) for the sequence of evaluation examples. Taking a closer look at the HMM, one notices that in every state, exactly one observation is output and hence, exactly one of the identified contexts is selected. Thus, the hidden states of the HMM adapted themselves to the unconsidered contexts. In the future, we would like to focus on finding more complex interdependencies between unconsidered contexts. Additionally, the case in which no parts of the system are appropriate for the unconsidered context in an application situation is not covered yet.

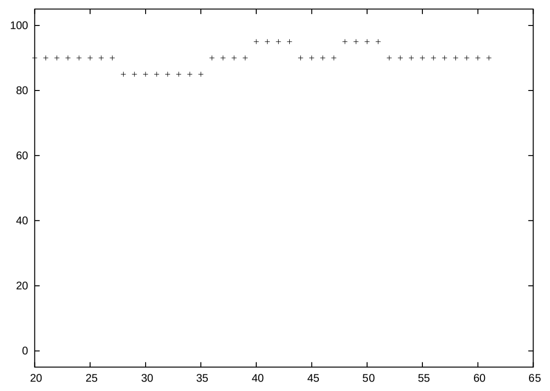
Acknowledgement. The authors wish to express their thanks to the anonymous reviewers of the first version of this paper. Your comments were very valuable for the creation of the final version!


0 - Pi= 1 - [Integer distribution --- 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
6 - [Integer distribution --- 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
1 - [Integer distribution --- 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
2 - [Integer distribution --- 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
8 - [Integer distribution --- 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
3 - [Integer distribution --- 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
7 - [Integer distribution --- 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
9 - [Integer distribution --- 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
4 - [Integer distribution --- 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
5 - [Integer distribution --- 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
0.3
0.7
0.67
0.33
0.86
0.14
0.5
0.5
1
1
1
1
1
1

(a) HMM created for prediction – octagonal node is initial node



(b) Baseline Performance



(c) Performance with Identification and HMM Predictor

Fig. 6. Results in the CSL-Domain

References

1. Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1), March 1986.
2. Peter Gärdenfors. Belief revision: An introduction. In Peter Gärdenfors, editor, *Belief Revision*, pages 1–28. Cambridge University Press, 1992.
3. Michael R. Genesereth and Nils J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers, Inc., 1987.
4. Allen Ginsberg, Sholom M. Weiss, and Peter Politakis. Automatic knowledge base refinement for classification systems. *Artificial Intelligence*, 2(35):192–226, 1988.
5. David E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Publishing Company, Inc, 1989.
6. Michael Bonnell Harries, Kim Horn, and Claude Sammut. Learning in time ordered domains with hidden changes in context. In *Papers from the AAAI 1998 Workshop on Predicting the Future: AI Approaches to Time-Series Problems*, pages 29–33, 1998.
7. David P. Helmbold and Philip M. Long. Tracking drifting concepts by minimizing disagreements. Technical Report UCSC-CRL-91-26, 1991.
8. Douglas B. Lenat. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.
9. Bernd Ludwig. Tracing actions helps in understanding interactions . In Jan Alexandersson and Alistair Knott, editors, *Proceedings of the 7th SIGdial Workshop on Discourse and Dialogue*, 2006.
10. Stefan Mandl, Bernd Ludwig, Sebastian Schmidt, and Herbert Stoyan. Recurring hidden contexts in online concept learning. In *Workshop on Planning, Learning and Monitoring with Uncertainty in Dynamic Worlds*, pages 25–29, 2006.
11. John McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.
12. John McCarthy. Notes on formalizing context. *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 1:555–560, 1993.
13. I. Niemelä, P. Simons, and T. Syrjänen. Smodels: a system for answer set programming. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning*, April 2000.
14. Frank Puppe, Herbert Stoyan, and Rudi Studer. Knowledge engineering. In Günther Görz, Claus-Rainer Rollinger, and Josef Schneeberger, editors, *Handbuch der Künstlichen Intelligenz*, chapter 15, pages 599–641. Oldenbourg Verlag, 4. auflage edition, 2003.
15. Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, 1989.
16. Raymond Reiter. Nonmonotonic reasoning. *Annual Review of Computer Science*, 2:147–186, 1987.
17. Albrecht Schmidt. *Ubiquitous Computing — Computing in Context*. PhD thesis, Lancaster University, 2002.
18. P. Simons. Extending and implementing the stable model semantics, 2000. Research Report 58, Helsinki University of Technology, Helsinki, Finland.
19. Gerhard Widmer and Miroslav Kubat. Learning flexible concepts from streams of examples: FLORA 2. In *European Conference on Artificial Intelligence*, pages 463–467, 1992.
20. Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.