Using Extended Logic Programming for Alarm-Correlation in Cellular Phone Networks

Peter Fröhlich[1], Wolfgang Nejdl[1], Michael Schroeder[2], ⋆
Carlos Damásio[2], and Luis Moniz Pereira[2]

[1] Universität Hannover, Germany (froehlic, nejdl)@kbs.uni-hannover.de
[2] City University, London, msch@cs.city.ac.uk
[3] Universidade Nova de Lisboa, Portugal (cd,lmp)@di.fct.unl.pt

Abstract. In this paper, we describe how to realise alarm-correlation in cellular phone networks using extended logic programming which provides integrity constraints, implicit and explicit negation.

1 Introduction

Though cellular phone networks already contain intelligent network elements diagnosing local faults, alarm bursts caused by network element failures cannot be handled properly by this technology [7]. To deal with alarm bursts alarm correlation systems are required to filter and condense the incoming alarms to meaningful high-level alarms and diagnoses. We review the application described in [7] and show how the problem is modelled and solved with extended logic programming which provides integrity constraints, implicit and explicit negation. Such a model-based approach allows to diagnose a system by constraining observed and predicted behaviour of the system. Consider Fig. 1. On the right we have observations of the actual system and on the left the model of the system with its predicted behaviour. If the constraints that observed and predicted behaviour should not differ is violated we can adapt the model's assumptions in order to satisfy the constraints and thus compute a diagnosis.

$$
\begin{array}{lcr}
\text{Model} & \text{Diagnoses} & \text{Actual System} \\
\downarrow & \uparrow & \downarrow \\
\text{Predictions} \longrightarrow & \text{Constraints} & \longleftarrow \text{Observations}
\end{array}
$$

Fig. 1. Model-based Diagnosis.

Mobile networks can be divided into three parts: the mobile station (MS), the access network with the base station transceiver (BTS) consisting of antennas, radio transceivers, cross connect systems (CC) and microwave (ML) or cable links (CL) and

⋆ Peter Fröhlich works now at ABB, Heidelberg, Carlos Damásio works also at Universidade Aberta de Lisboa.

the base station controller (BSC), and the switched network, which is connected to the access network by the BSC's. The BSC provides the radio resource management, which serves the control and selection of appropriate radio channels to interconnect the MS and the switched network. The switched network interconnects the MS to the communication partner, which might be another MS or an ISDN subscriber [7].

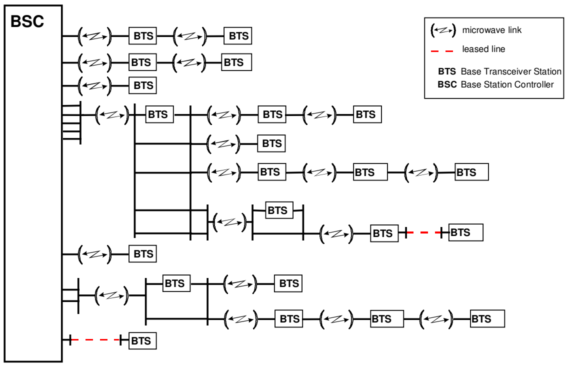


Fig. 2. Star-configuration of a base station subsystem [7].

The rest of the paper is organised as follows. First we introduce extended logic programs and diagnosis, then we show how to model cellular phone networks and alarm-correlation as extended logic programs, and finally we show how the alarm-correlation is realised in REVISE [5], a system for contradiction removal of extended logic programs.

2 Extended Logic Programming and Diagnosis

Since Prolog became a standard in logic programming much research has been devoted to the semantics of logic programs. In particular, Prolog's unsatisfactory treatment of negation as finite failure led to many innovations. Well-founded semantics [8] turned out to be a promising approach to cope with negation by default. Subsequent work extended well-founded semantics with a form of explicit negation and constraints [13] and showed that the richer language, called WFSX, is appropriate for a spate of knowledge representation and reasoning forms [11,12,1]. In particular, the technique of contradiction removal from extended logic programs [14] opens up many avenues in model-based diagnosis [12,3,4,10].

Definition 1. Extended Logic Program
An extended logic program is a (possibly infinite) set of rules of the form $L_0 \leftarrow L_1, \ldots, L_m, not L_{m+1}, \ldots, not L_n$ $(0 \leq m \leq n)$, where each L_i is an objective literal $(0 \leq$

$i \leq n$). An objective literal is either an atom A or its explicit negation $\neg A$.[1] Literals of the form $notL$ are called default literals. Literals are either objective or default ones.

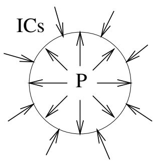
The behaviour of the system to be diagnosed is coded as an extended logic program. To express the assumption that the system works correctly by default we use negation by default.

Example 2. Consider the cellulular phone network in Fig. 2. Assumption a microwave link is ok, it properly propagates signals:

$$
\begin{aligned}
& signal(NE, up, Sender, Signal) \leftarrow \\
& \quad not\ ab(NE), \\
& \quad type(NE, ml), type(Sender, bts), \\
& \quad class(Signal, farend\_or\_status\_signal), \\
& \quad signal(NE, down, Sender, Signal).
\end{aligned}
$$

Rules as the one above allow one to predict the behaviour of the system to be diagnosed in case it is working fine. To express that normality assumptions may lead to contradictions between predictions and actual observations we introduce integrity constraints.

Definition 3. Integrity Constraint
An integrity constraint has the form $\bot \leftarrow L_1, \ldots, L_m, notL_{m+1}, \ldots, notL_n$ $(0 \leq m \leq n)$ where each L_i is an objective literal ($0 \leq i \leq n$), and $\bot$ stands for false.



Syntactically, the only difference between the program rules and the integrity constraints is the head. A rule's head is an objective literal, whereas the constraint's head is $\bot$, the symbol for false. Semantically the difference is that program rules open the solution space, whereas constraints limit it, as indicated on the left.

Example 4. Integrity Constraint
Now we can express that a contradiction arises if predictions and observations differ. In the setting of alarm-correlation we use for example the constraint

$$\bot \leftarrow \neg signal(bsc, down, Sender, alive), signal(bsc, down, Sender, alive).$$

to express that it is contradictory for the BSC to allegedly have received an alive signal of a BTS and to know at the same time that it has not.

A contradiction is always based on the assumption that the components work fine, i.e. the default literals are false. In general, we can remove a contradiction by partially dropping some closed world assumptions. Technically, we achieve this by adding a minimal set of revisable facts to the initially contradictory program:

[1] Note that the coherence principle relates explicit and default, or implicit, negation: $\neg L$ implies $notL$.

Definition 5. Revisable, Revision
The revisables R of a program P are a subset of the default negated literals which do not occur as rule heads in P. The set $R' \subseteq R$ is called a revision if it is a minimal set such that $P \cup R'$ is free of contradiction , i.e. $P \cup R' \not\models_{WFSX} \perp$

For details on the definition of the inference operator $\models_{WFSX}$ see e.g. [1,2].

Example 6. In the example above $not\ ab(NE)$ is a revisable.

The limitation of revisability to default literals which do not occur as rule heads is adopted for efficiency reasons, but without loss of generality. We want to guarantee that the truth value of revisables is independent of any rules. Thus we can change the truth value of a revisable whenever necessary without considering an expensive derivation of the default literal's truth value.

In the next section we will show how to model alarm-correlation as extended logic programs.

3 Modelling Cellular Phone Networks

The cellular phone networks are configured in a star topology (see Fig. 2) with exactly one path from a BTS to the BSC. Since such networks are highly dynamic an explicit model of the network is a necessary prerequisite for alarm correlation. Consider the path depicted in Fig. 3. Its topology is modelled by facts on the components' types and connections (see Fig. 4). The fact $conn(ml16, up, bsc, down)$ states, for example, that the up-stream port of microwave link $ml16$ is connected to the down-stream port of the BSC.

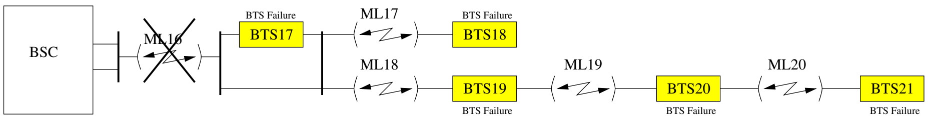


Fig. 3. Network's topology where microwave link $ml18$ is faulty [7].

$type(ml16, ml).$	$conn(ml16, up, bsc, down).$
$type(ml17, ml).$	$conn(ml18, up, ml16, down).$
$type(ml18, ml).$	$conn(bts18, up, ml17, down).$
$\ldots$	$\ldots$

Fig. 4. Facts for the network's topology.

The network elements are intelligent and perform local diagnosis resulting in alarm messages which are sent to the BSC. We distinguish three classes for the alarm messages:

- Farend alarms are generated by a BTS if the the components farther away from the BSC are not reachable anymore.
- BTS-failure alarms are generated directly in the BSC, when it detects that a BTS is not reachable anymore.
- The status signal *alive* is sent from each BTS on a periodical polling of the BSC. It is used to generate appropriate alarms in the BSC in case it gets lost on the way from the BTS to the BSC. The message is not physically present.

The alarms are classified by the facts *class* and *bts_failure_alarm* as shown in Fig. 5. The generation and suppression of alarms is captured by the rules shown in Fig. 6. The rules on the left-hand side state that each BTS sends an alive message and an alarm message in case of a farend alarm, respectively. The rules on the right-hand side express that there cannot be an alive message at the BSC if there is a BTS-failure or a farend alarm. Here explicit negation ($\neg signal$) proves to be very useful to get a compact model.

$$class(bts\_omu\_link\_fail, bts\_failure\_signal).$$
$$class(bcch\_missing, bts\_failure\_signal).$$
$$class(available\_traffic, bts\_failure\_signal).$$
$$class(lapd\_link\_failure, bts\_failure\_signal).$$
$$class(farend\_alarm\_1, farend\_signal).$$
$$class(alive, status\_signal).$$

$$\begin{array}{l} bts\_failure\_alarm(Sender) \leftarrow \\ \quad alarm(Sender, bts\_omu\_link\_fail), \\ \quad type(Sender, bts). \\ bts\_failure\_alarm(Sender) \leftarrow \\ \quad alarm(Sender, bcch\_missing), \\ \quad type(Sender, bts). \end{array}$$

$$\begin{array}{l} bts\_failure\_alarm(Sender) \leftarrow \\ \quad alarm(Sender, available\_traffic), \\ \quad type(Sender, bts). \\ bts\_failure\_alarm(Sender) \leftarrow \\ \quad alarm(Sender, lapd\_link\_failure), \\ \quad type(Sender, bts). \end{array}$$

Fig. 5. Alarm classes.

$$\begin{array}{l} signal(Sender, up, Sender, alive) \leftarrow \\ \quad type(Sender, bts). \\ \\ signal(Sender, up, Sender, Signal) \leftarrow \\ \quad alarm(Sender, Signal), \\ \quad class(Signal, farend\_signal), \\ \quad type(Sender, bts). \end{array}$$

$$\begin{array}{l} \neg signal(bsc, down, Sender, alive) \leftarrow \\ \quad bts\_failure\_alarm(Sender), \\ \quad type(Sender, bts). \\ \neg signal(bsc, down, Sender, alive) \leftarrow \\ \quad alarm(Sender, Signal), \\ \quad class(Signal, farend\_signal), \\ \quad type(Sender, bts). \end{array}$$

Fig. 6. Signal generation and suppression.

In Fig. 7 we formalize how signals are propagated over connections and through components (BTS or ML). While the BTS cannot fail, the microwave link may be faulty, though we assume by default that it is not abnormal (*not* $ab(NE)$). This default literal is a revisable whose truth value may be changed to satisfy the constraints.

$$\begin{array}{l} signal(NE_2, down, Sender, Signal) \leftarrow \\ \quad type(Sender, bts), class(Signal, farend\_or\_status\_signal), \\ \quad conn(NE_1, up, NE_2, down), signal(NE_1, up, Sender, Signal). \end{array}$$

$$\begin{array}{l|l} signal(NE, up, Sender, Signal) \leftarrow & signal(NE, up, Sender, Signal) \leftarrow \\ \quad class(Signal, farend\_or\_status\_signal), & \quad not\ ab(NE), \\ \quad type(NE, bts), type(Sender, bts), & \quad type(NE, ml), type(Sender, bts), \\ \quad NE \neq Sender, & \quad class(Signal, farend\_or\_status\_signal), \\ \quad signal(NE, down, Sender, Signal). & \quad signal(NE, down, Sender, Signal). \end{array}$$

Fig. 7. Signal propagation.

Finally, we have to specify the integrity constraints that it is contradictory to have and not to have an alive message at the BSC and that there has to be either an alive message, or a BTS failure alarm, or the message was lost by the BTS (see Fig. 8). The literal *message_lost* is a revisable which is assumed false, but may be changed to satisfy the constraints. Lost messages are more likely to occur that abnormal microwave links which is specified in Fig. 8.

$$\begin{array}{l|l} \bot \leftarrow type(Sender, bts), & \bot \leftarrow \neg signal(bsc, down, Sender, alive), \\ \qquad not\ signal(bsc, down, Sender, alive), & \qquad signal(bsc, down, Sender, alive). \\ \qquad not\ bts\_failure\_alarm(Sender), & probability(ab(\_), 0.001). \\ \qquad not\ message\_lost(Sender). & probability(message\_lost(\_), 0.1). \end{array}$$

Fig. 8. Constraints for signals and a-priori probabilities of revisables.

In order to compute diagnoses given the above model, consider Fig. 1 again. The model of the network is consistent as long as there are no alarms. If alarms are generated and received by the BSC the constraints are violated and satisfying them yields the correct diagnoses of the problem.

Example 7. Consider the alarms in Fig. 9. The burst of messages is difficult to survey for a human operator, though the most probable explanation by revision is fairly easy: microwave link $ml16$ is abnormal.

To correlate the alarms, or to speak more generally, to compute revisions of contradictory extended logic programs, we use the REVISE system [5] which implements an adaption of the hitting-set algorithm [15,9] suitable for extended logic programming.

$alarm(bts17, bcch\_missing).$	$alarm(bts19, pcm\_failure).$
$alarm(bts17, bcf\_bie\_alarm\_in).$	$alarm(bts19, bts\_omu\_link\_fail).$
$alarm(bts17, pcm\_fail).$	$alarm(bts20, bcch\_missing).$
$alarm(bts17, bts\_omu\_link\_fail).$	$alarm(bts20, pcm\_fail).$
$alarm(bts18, bcch\_missing).$	$alarm(bts20, bts\_omu\_link\_fail).$
$alarm(bts18, pcm\_failure).$	$alarm(bts21, bcch\_missing).$
$alarm(bts18, bts\_omu\_link\_fail).$	$alarm(bts21, pcm\_fail).$
$alarm(bts19, bcf\_bie\_alarm\_in).$	$alarm(bts21, bts\_omu\_link\_fail).$

Fig. 9. Alarms.

4 Computing the Revisions

Before we show how the revisions are computed , we need some defintions. Conflicts are sets of default assumptions that lead to a contradiction.

Definition 8. Conflict
Let P be an extended logic program with default literals D. Then $C \subset D$ is a *conflict* iff

$$P \cup \{\neg c \mid not\ c \in C\} \models \bot$$

To compute revisions, we have to change default assumptions so that all conflicts are covered. Such a cover is called hitting set, since all conflicts involved are hit.

Definition 9. Hitting Set
A *hitting set* for a collection of sets C is a set $H \subseteq \bigcup_{S \in C} S$ such that $H \cap S \neq \{\}$ for each $S \in C$. A hitting set is minimal iff no proper subset of it is a hitting set for C.

Conflicts and hitting sets allow to compute revisions:

Theorem 10. *Adopted from [15]*
Let P be a program. Then R is a revision of P iff R is a minimal hitting set for the collection of conflicts for P.

Thus, to compute revisions we use hitting sets which are obtained from hitting set trees [15]:

Definition 11. Hitting Set Tree
Let C be a collection of sets. An HS-tree for C, call it T, is a smallest edge-labeled and node-labeled tree with the following properties:

1. The root is labeled $\surd$ if C is empty. Otherwise the root is labeled by an arbitrary set of C.
2. For each node n of T, let $H(n)$ be the set of edge labels on the path in T from the root node to n. The label for n is any set $\Sigma \in C$ such that $\Sigma \cap H(n) = \{\}$, if such a set Σ exists. Otherwise, the label for n is $\surd$. If n is labeled by the set Σ, then for each $\sigma \in \Sigma$, n has a successor, n_σ, joined to n by an edge labeled by σ.

To compute hitting set trees, Reiter proposed an algorithm [15] which was corrected in [9]. For the sake of brevity and clarity we explain the algorithm with its adaption to extended logic programs informally and by means of an extensive example.

The derivation tree on the right of Figure 10 show only the most relevant literals, the actual proof tree is based on the SLX proof-procedure for WFSX described in [1]. Since programs may be contradictory the paraconsistent version of WFSX is used. The top-down characterization of WFSX relies on the construction of two types of AND-trees (T and TU-trees), whose nodes are either assigned the status successful or failed. T-trees compute whether a literal is true; TU-trees whether it is true or undefined. A successful (resp. failed) tree is one whose root is successful (resp. failed). If a literal L has a successful T-tree rooted in it then it belongs to the paraconsistent well-founded model of the program (WFM_p); otherwise, i.e. if all T-trees for L are failed, L does not belong to the WFM_p. Accordingly, failure does not mean falsity, but simply failure to prove verity.

A T-tree is constructed as an ordinary SLDNF-tree. However, when a $notL$ goal is found the subsidiary tree for L is constructed as a TU-tree: $notL$ is true if the attempt to prove L true or undefined fails. To enforce the coherence principle we add in TU-trees the literal $not\neg L$ to the resolvant, whenever the objective literal $\neg L$ is selected for expansion. When a $notL$ goal is found in a TU-tree the subsidiary tree for L is constructed as a T-tree: $notL$ is true or undefined if the attempt to prove L true fails. Besides these differences, the construction of TU-trees proceeds as for SLDNF-trees.

Apart from floundering, the main issues in defining top-down procedures for Well-Founded Semantics are infinite positive recursion, and infinite recursion through negation by default. The former gives rise to the truth value false (the query L fails and the query $not\ L$ succeeds when L is involved in the recursion), and the latter to the truth value undefined (both L and $not\ L$ fail). Cyclic infinite positive recursion is detected locally in T-trees and TU-trees by checking if a literal L depends on itself. A list of local ancestors is maintained to implement this pruning rule. For cyclic infinite negative recursion detection a set of global ancestors is kept. If one is expanding, in a T-tree, a literal L which already appears in an ancestor T-tree then the former T-tree is failed. If one is expanding, in a TU-tree, a literal L which already appears in an ancestor TU-tree then the literal L is successful in the former TU-tree. The SLX proof-procedure has been extended in the REVISE implementation, and named SLXA, to be able to return the conflicts supporting $\perp$.

The SLXA procedure makes sure in T-trees that a revisable is false; in TU-trees it assumes that every revisable found is true. The revisables discovered are collected and returned to the invoker. Mark the similarities with Eshghi and Kowalski's abductive procedure where there is a consistency phase and an abductive phase [6].

The calls to SLXA are driven by the REVISE engine. Its main data structure is the hitting-set tree. The construction of the hitting-set tree is started on candidate $\{\}$, meaning that the revisables initially have their default value. We say that the node $\{\}$ has been expanded when the SLXA procedure is called to determine one conflict. If there is none, then the program is non-contradictory and the revision process is finished. Otherwise, the REVISE engine computes all the minimal ways of satisfying the conflicted integrity constraint returned by SLXA, i.e. the sets of revisables which have to be added to program in order to remove that particular conflict. For each of these sets of revisables, a child node of $\{\}$ is created. If there is no way of satisfying the conflicted integrity the program is contradictory. Else the REVISE engine selects a node to expand according to some preference criterium and cycles: it determines a new conflict, it expands that node with the revisables which remove the conflict, etc.... When a solution is found, i.e. there is no conflict, it is kept in a table for pruning the revision tree, by removing any nodes which contain some solution, and have been selected according to the preference criterium.

The order in which the nodes of the revision tree are expanded is important to obtain minimal solutions first. To abstract the preference ordering we assign to every node a key and a code. The code is a representation of the node in the lattice induced by the preference ordering $\preceq$. The key is a natural number with the property that given two elements e_1 and e_2 in the preference ordering $\preceq$ such that the corresponding key_1 and key_2 obey $key_1 \leq key_2$ then $e_2 \not\prec e_1$. This guarantees that if we expand first the nodes with smallest key we find the minimal solutions first. Furthermore, two nodes with the same key are incomparable. For the case of minimality by set-inclusion, the key corresponds to the cardinality of the candidate set for revision.

Minimality criteria, as for example minimality by cardinality, minimality by set-inclusion or minimality by probability, are implemented as an abstract data type with three functions: `empty_code`, `add_new_rev`, and `smaller_code`. The first returns the code associated with the candidate set $\{\}$. By definition, the key of this candidate set is 0. Given a new literal which has to be added to a node, the second function computes the new code and new key of the new candidate set. The latter function implements the subsumption test of two codes with respect to the preference ordering.

Example 12. Consider the description of the phone network of the previous section. Given the two alarm

$$alarm(bts20, bcch_missing). \qquad alarm(bts20, lapd_link_failure).$$

the most probable solution is that microwave 19 is faulty and a message from $bts21$ was lost. To compute these two revisions, REVISE proceeds as shown in Figure 10. The left column shows the expanding hitting set tree and the right the inference of contradictions with the responsible conflict. For clarity's sake, the inference tree contains only the most relevant literals such as *signal*, etc. Assuming nothing (2), the base station controller explicitly derives from the given alarms that there cannot be an alive signal of $bts20$. However, since all components are assumed to be working correctly, the BSC also derives that there should be an alive signal of BTS20. The components assumed fault-free and involved in the propagation are microwave links $ml16, ml18, ml19$. They form the first conflict.

In (3) these three microwave links form the first conflict, the root of the hitting set tree. The tree is now step by step expanded. First, it is assumed that $ml16$ is abnormal (4). A new conflict is derived since the *bsc* does not have an alive signal from $bts17$, neither a failure alarm, nor was a message lost. The latter is revisable, i.e. we may assume that messages get lost, and thus the conflict $msg\_lost(bts17)$ is returned. The hitting set tree is updated accordingly (5) and in particular, the nodes are re-ordered according to their probability. While the branches for microwave links 18 and 19 have a probability of 0.001, the branch of $ab(ml16), msg\_lost(bts17)$ has only a probability of 0.0001.

Similar to (4), $ml18$ is now assumed abnormal (6), leading to a new conflict involving $msg\_lost(bts19)$. This process of changing assumptions, deriving a contradiction with associated conflict and updating the hitting-set tree accordingly until finally, assuming $ml19$ abnormal and the message from $bts21$ lost, there is no further conflict. Due to the constant re-ordering of the hitting-set tree to expand the most probable candidate next, we know that we found the most probable solution and can terminate the search.

Conclusions

In cellular phone networks faults of microwave links are likely to cause alarm showers, which put the system operator to a hard test. Alarm correlation tools are needed, which support the operators by identifying the cause of a large number of alarm messages.

The main contribution of this paper is twofold: We have shown how to model alarm-correlation in cellular-phone networks declaratively as extended logic programs and how to diagnose these networks by satisfying violated integrity constraints using the REVISE[2] system. The integrity constraints and implicit negation are used to constrain predictions and observations and express default assumptions about the components states, respectively.

Many practical problems like alarm-correlation require a logical explanation of observations and therefore abductive capabilities as used in our approach. When compared with other approaches such as neural networks [16], there are three major benefits:

1. A very small and maintainable system description, that separates structural from behavioural components and thus makes changes of the network topology easy.
2. A propagation model which allows to correctly diagnose unforeseen errors as well as multiple faults.
3. Failure probability estimates (see Fig. 8), which lead to correct diagnoses even on noisy data, where alarm messages have been lost.

References

1. J. J. Alferes, C. V. Damásio, and L. M. Pereira. A logic programming system for non-monotonic reasoning. *Journal of Automated Reasoning*, 14(1):93–147, 1995.
2. J. J. Alferes and L. M. Pereira. *Reasoning with Logic Programming*. (LNAI 1111), Springer-Verlag, 1996.
3. Carlos Viegas Damásio, Wolfgang Nejdl, and Luis Moniz Pereira. REVISE: An extended logic programming system for revising knowledge bases. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Knowledge Representation and Reasoning*, pages 607–618, Bonn, Germany, May 1994. Morgan Kaufmann.

[2] REVISE is online at www.soi.city.ac.uk/˜msch.

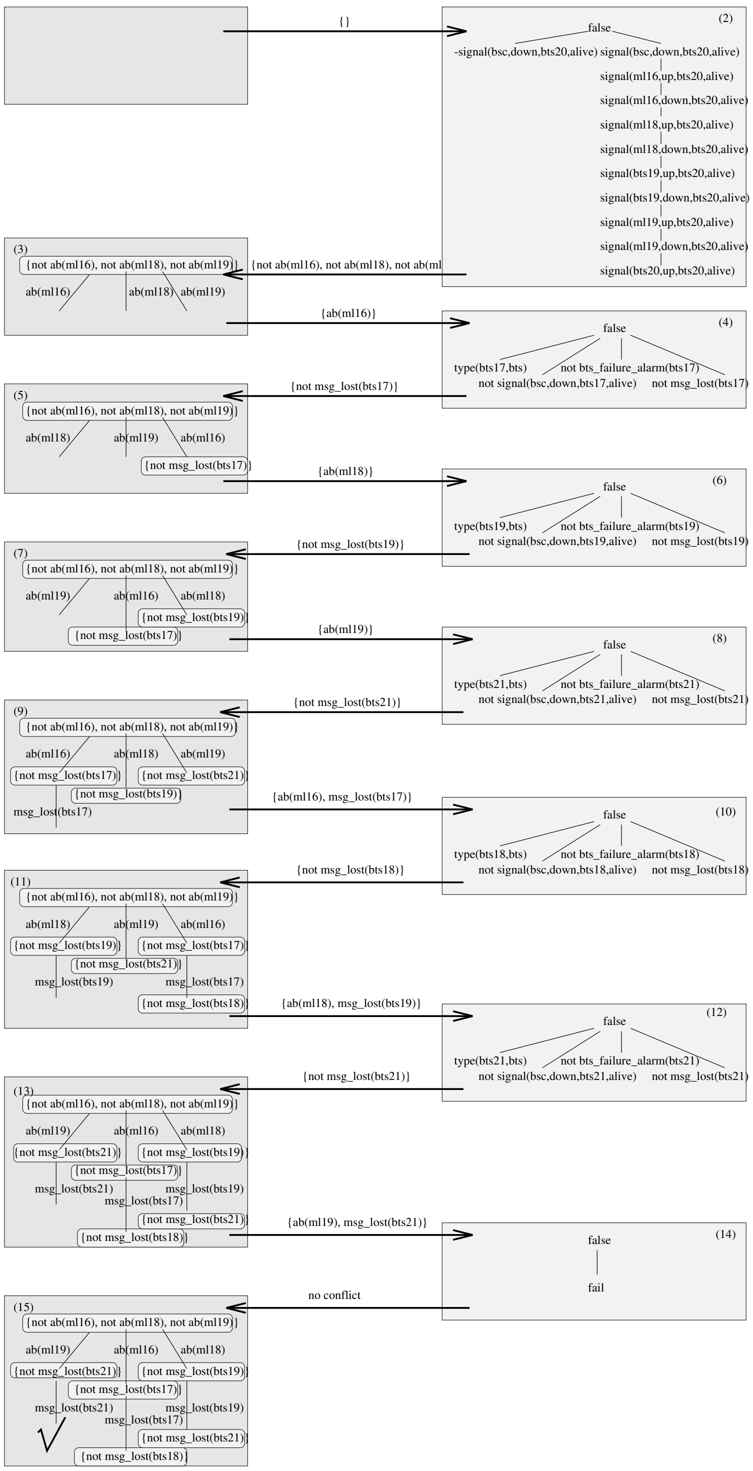


Fig. 10. Computation of revisions in REVISE.

4. Carlos Viegas Damásio, Wolfgang Nejdl, Luis Moniz Pereira, and Michael Schroeder. Model-based diagnosis preferences and strategies representation with meta logic programming. In Krzysztof R. Apt and Franco Turini, editors, *Meta-logics and Logic Programming*, chapter 11, pages 269–311. The MIT Press, 1995.
5. Carlos Viegas Damásio, Luis Moniz Pereira, and Michael Schroeder. REVISE: Logic programming and diagnosis. In *Proceedings of the Conference on Logic Programming and Non-monotonic Reasoning LPNMR97*. LNAI 1265, Springer–Verlag, 1997.
6. K. Eshghi and R. Kowalski. Abduction compared with negation by failure. In *6th Int. Conf. on LP*. MIT Press, 1989.
7. Peter Fröhlich, Wolfgang Nejdl, Klaus Jobmann, and Hermann Wietgrefe. Model-based alarm correlation in cellular phone networks. In *Fifth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, January 1997.
8. Allen Van Gelder, Kenneth Ross, and John S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proceeding of the 7th ACM Symposium on Principles of Databse Systems*, pages 221–230. Austin, Texas, 1988.
9. Russell Greiner, Barbara A. Smith, and Ralph W. Wilkerson. A correction of the algorithm in reiter's theory of diagnosis. *Artificial Intelligence*, 41(1):79–88, 1989.
10. I. A. Móra and J. J. Alferes. Diagnosis of distributed systems using logic programming. In C. Pinto-Ferreira and N.J. Mamede, editors, *Progress in Artificial Intelligence, 7th Portuguese Conference on Artificial Intelligence EPIA95*, volume LNAI990, pages 409–428. Springer–Verlag, Funchal, Portugal, 1995.
11. L. M. Pereira, J. N. Aparicio, and J. J. Alferes. Non–monotonic reasoning with logic programming. *Journal of Logic Programming. Special issue on Nonmonotonic reasoning*, 17(2, 3 & 4), 1993.
12. L. M. Pereira, C. V. Damásio, and J. J. Alferes. Diagnosis and debugging as contradiction removal. In L. M. Pereira and A. Nerode, editors, *2nd Int. Workshop on Logic Programming and Non-Monotonic Reasoning*, pages 334–348, Lisboa, Portugal, June 1993. MIT Press.
13. Luis Moniz Pereira and José Júlio Alferes. Well founded semantics for logic programs with explicit negation. In *B. Neumann (Ed.), European Conference on Artificial Intelligence*, pages 102–106. John Wiley & Sons, 1992.
14. Luis Moniz Pereira, José Júlio Alferes, and Joaquim Aparicio. Contradiction Removal within Well Founded Semantics. In A. Nerode, W. Marek, and V. S. Subrahmanian, editors, *Logic Programming and Nonmonotonic Reasoning*, pages 105–119, Washington, USA, June 1991. MIT Press.
15. Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–96, 1987.
16. Hermann Wietgrefe, Klaus-Diter Tuchs, Klaus Jobmann, Guido Carls, Peter Fröhlich, Wolfgang Nejdl, and Sebastian Steinfeld. Using neural networks for alarm correlation in cellular phone networks. In *Proceedings of International Workshop on Applications of Neural Networks in Telecommunications*, 1997.