

# A Compositional Reasoning System for Executing Nonmonotonic Theories of Reasoning

J. Engelfriet and J. Treur

Vrije Universiteit Amsterdam, Department of Artificial Intelligence  
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands  
email: treur@cs.vu.nl, URL: <http://www.cs.vu.nl/~treur>

## 1 Introduction

An agent that is reasoning about the world often needs to draw (defeasible) conclusions that are not logically entailed by its (incomplete) knowledge about the world. Nonmonotonic reasoning systems can be used to model these reasoning patterns. Implementations of agents that can reason in a defeasible manner need to include an implemented nonmonotonic reasoning system. In different applications, agents may need different types of nonmonotonic reasoning. Therefore, it is useful to develop a generic reasoning system that covers different types of nonmonotonic reasoning (as opposed to implementations for one specific nonmonotonic formalism as in e.g., [Ni96], [RS94] or [CMT96]). Moreover, the development of such a generic reasoning system can be made in a transparent manner if a central role is played by an implementation-independent design specification based on current software engineering principles, such as compositionality and information hiding.

Reasoning can be seen as an activity of an agent taking place in time. The agent starts with a set of initial beliefs to which it applies some (nonmonotonic) rules to arrive at a new state, in which it has more knowledge. In this new state, the agent may again apply rules to arrive at a next state. Viewing the agent from the outside, we can look at the knowledge of the agent at all points in time. Thus, more formally, we can describe the behavior of this agent by a reasoning trace, or a temporal model that describes what the agent knows at the different points in time. Since (nonmonotonic) reasoning may be nondeterministic (the agent may have a choice of which rules to apply), we should allow multiple traces that start with the same set of initial beliefs. If we want to describe the reasoning of the agent exhaustively, we may incorporate traces starting with any set of initial beliefs. To specify such sets of temporal models (or traces), temporal logic can be used. This general temporal view of reasoning was put forth in [ET96], where we introduced a formal notion of temporal model, and a temporal specification language suited to specify sets of temporal models. In the current paper a design and specification of an executable nonmonotonic reasoning system to implement this generic approach to nonmonotonic reasoning is presented. For any specification of a theory of nonmonotonic reasoning, given as an input to the system, the possible reasoning traces are generated.

The system was developed using the compositional modelling environment DESIRE (framework for DEsign and Specification of Interacting REasoning components; see [LPT92], [BDJT97], [BTWW95]). This environment for the

development of compositional reasoning systems and multi-agent systems has been successfully used to design various types of reasoning systems and multi-agent systems and applications in particular domains. The DESIRE software environment offers a graphical editor, an implementation generator and an execution environment to execute specifications automatically. In its use to build complex reasoning systems, DESIRE can be viewed as an advanced theorem proving environment in which both the knowledge and the control of the reasoning can be specified in an explicit, declarative and compositional manner.

In this paper, in Section 2 we briefly summarize the generic approach introduced in [ET96]. In Section 3 a very brief introduction to DESIRE is presented. In Section 4 the design of the nonmonotonic reasoning system is discussed. An example trace of the system is described in Section 5, and Section 6 gives conclusions and suggestions for further research.

## 2 Temporal Specification of Nonmonotonic Reasoning

In this section we will briefly review the main notions introduced in [ET96]. The basic language in which the agent expresses its beliefs will be propositional logic. Semantically, a state is a set of propositional models, representing the set of beliefs consisting of the formulas true in all models in the state.

### Definition 2.1 (Temporal Model)

- i) An *information state* is a non-empty set of propositional models. A propositional formula  $\alpha$  is true in an information state  $\mathbf{M}$ , denoted  $\mathbf{M} \models \alpha$ , if  $\mathbf{m} \models \alpha$  for all  $\mathbf{m} \in \mathbf{M}$ . For two information states  $\mathbf{M}, \mathbf{N}$ , we say  $\mathbf{M}$  contains more information than  $\mathbf{N}$ , denoted  $\mathbf{N} \leq \mathbf{M}$  if  $\mathbf{M} \subseteq \mathbf{N}$ .
- i) A *temporal model* is a sequence  $(\mathcal{M}_s)_{s \in \mathbf{N}}$  where each  $\mathcal{M}_s$  is an information state.
- ii) A temporal model  $\mathcal{M}$  is *conservative* if  $\mathcal{M}_s \leq \mathcal{M}_{s+1}$  for all  $s \in \mathbf{N}$ .
- iii) The *refinement ordering*  $\leq$  on temporal models is defined by:  
 $\mathcal{M} \leq \mathcal{N} \Leftrightarrow$  for all  $s$ :  $\mathcal{M}_s \leq \mathcal{N}_s$  and  $\mathcal{M}_0 = \mathcal{N}_0$ .

Note that when  $\mathbf{N} \leq \mathbf{M}$ , for all propositional formulas  $\alpha$  we have  $\mathbf{N} \models \alpha \Rightarrow \mathbf{M} \models \alpha$ , so that indeed  $\mathbf{M}$  contains more information than  $\mathbf{N}$ . The conservativity of a temporal model means that the agent never forgets anything it has previously deduced. The temporal language we will introduce contains the operators  $\mathbf{H}_0$ ,  $\mathbf{F}$ ,  $\mathbf{G}$  and  $\mathbf{C}$  which refer respectively to knowledge at point  $\mathbf{0}$ , knowledge at some point in the future, at all points in the future, and to knowledge in the current time point. The formal semantics of these operators are as follows.

### Definition 2.2 (Temporal Interpretation)

- a) For a propositional formulae  $\varphi$ :
  - $(\mathcal{M}, s) \models \mathbf{F}\varphi \Leftrightarrow$  there exists  $t \in \mathbf{N}$ ,  $t > s$  such that  $\mathcal{M}_t \models \varphi$
  - $(\mathcal{M}, s) \models \mathbf{G}\varphi \Leftrightarrow$  for all  $t \in \mathbf{N}$  with  $t > s$ :  $\mathcal{M}_t \models \varphi$
  - $(\mathcal{M}, s) \models \mathbf{C}\varphi \Leftrightarrow \mathcal{M}_s \models \varphi$

- $\mathcal{M}, s \models \mathbf{H}_0\varphi \quad \Leftrightarrow \quad \mathcal{M}_0 \models \varphi$   
 b) For a temporal formula  $\alpha$ :  
 $\mathcal{M}, s \models \neg \alpha \quad \Leftrightarrow \quad$  it is not the case that  $\mathcal{M}, s \models \alpha$   
 c) For a set  $\mathbf{A}$  of temporal formula:  
 $\mathcal{M}, s \models \bigwedge \mathbf{A} \quad \Leftrightarrow \quad$  for all  $\varphi \in \mathbf{A}$ :  $\mathcal{M}, s \models \varphi$   
 d) A formula  $\varphi$  is true in a model  $\mathcal{M}$ , denoted  $\mathcal{M} \models \varphi$ , if for all  $s \in \mathbf{N}$ :  $\mathcal{M}, s \models \varphi$   
 e) A set of formulae  $\mathbf{T}$  is true in a model  $\mathcal{M}$ , denoted  $\mathcal{M} \models \mathbf{T}$ , if for all  $\varphi \in \mathbf{T}$ ,  $\mathcal{M} \models \varphi$ . We call  $\mathcal{M}$  a model of  $\mathbf{T}$ .

In a specification, we allow only a certain format in the temporal formulas, which corresponds to the application of a rule by the agent:

**Definition 2.3 (Reasoning Theories)**

- a) A formula is called a (*nonmonotonic*) *reasoning formula* if it is of the form  
 $\alpha \wedge \beta \wedge \varphi \wedge \psi \rightarrow \mathbf{G}\gamma$ , where  
 $\alpha = \bigwedge \{ \mathbf{H}_0\varepsilon \mid \varepsilon \in \mathbf{A} \}$  for a set of propositional formulae  $\mathbf{A}$ .  
 $\beta = \bigwedge \{ \neg \mathbf{H}_0\delta \mid \delta \in \mathbf{B} \}$  for a set of propositional formulae  $\mathbf{B}$ .  
 $\varphi = \bigwedge \{ \neg \mathbf{F}\theta \mid \theta \in \mathbf{C} \}$  for a set of propositional formulae  $\mathbf{C}$ .  
 $\psi = \bigwedge \{ \mathbf{C}\zeta \mid \zeta \in \mathbf{D} \}$  for a set of propositional formulae  $\mathbf{D}$ .  
 $\gamma$  is a propositional formula.  
 b) A set  $\mathbf{Th}$  of reasoning formulae is called a *theory of reasoning*.

In a temporal model of a theory of reasoning, all (nonmonotonic) rules which are applicable at some point in time, have actually been applied by the agent. But we also want to make sure that the agent knows *nothing more* than what it can deduce. So we will look at models of a theory (the rules have been applied) in which the knowledge of the agent over time is minimal (the agent knows nothing more).

**Definition 2.4 (Minimal Temporal Model)**

- A temporal model  $\mathcal{M}$  is called a *minimal model* of a theory  $\mathbf{Th}$  if it is a model of  $\mathbf{Th}$  and for any model  $\mathcal{N}$  of  $\mathbf{Th}$ , if  $\mathcal{N} \leq \mathcal{M}$  then  $\mathcal{N} = \mathcal{M}$ .

The minimal models of a theory describe the reasoning process of an agent specified by this theory. The compositional reasoning system we will describe in Section 4 will find these minimal models by "executing" the theory. Our approach can therefore be seen as an executable modal (nonmonotonic) logic (see [FO95]).

We will give an example of such a theory. In [ET93] it was established that there exists a faithful translation of Reiter's default logic into the temporal language introduced above. The minimal models of the translation correspond to extensions of the theory. We will not go into the details of the translation, but rather give an example. Let the following default theory  $\langle \mathbf{W}, \mathbf{D} \rangle$  be given:  $\mathbf{W} = \{ \mathbf{a}, \mathbf{d}, \mathbf{b} \rightarrow \neg \mathbf{c} \}$  and  $\mathbf{D} = \{ (\mathbf{a}: \mathbf{b}) / \mathbf{b}, (\mathbf{d}: \mathbf{c}) / \mathbf{c}, (\mathbf{b}: \neg \mathbf{c}) / \mathbf{e} \}$ . The atoms  $\mathbf{a}$  and  $\mathbf{d}$  will be initial facts. Formally, the formula  $\mathbf{b} \rightarrow \neg \mathbf{c}$  would also be a fact, but since the generic reasoning system to be described below does not perform general propositional reasoning (it uses a subset of natural deduction called *chaining*), we will translate it into the following two rules which describe the application of the formula:

- (1)  $Cb \rightarrow G\neg c$ ,
- (2)  $Cc \rightarrow G\neg b$ ,

The default rules are translated as follows:

- (1)  $Cb \rightarrow G\neg c$ ,
- (2)  $Cc \rightarrow G\neg b$ ,
- (3)  $Ca \wedge \neg F\neg b \rightarrow Gb$ ,
- (4)  $Cd \wedge \neg F\neg c \rightarrow Gc$ ,
- (5)  $Cb \wedge \neg Fc \rightarrow Ge$

The idea behind this translation is that we should add the conclusion of a default rule like  $(b: \neg c) / e$  when  $b$  has been derived, and  $\neg c$  remains consistent throughout our further reasoning, which means we should never derive  $c$  in the future. The minimal models of the resulting theory are described by the following picture:

|          | M                   | N                   |
|----------|---------------------|---------------------|
| <b>a</b> | 1   1   1   1   ... | 1   1   1   1   ... |
| <b>b</b> | u   1   1   1   ... | u   u   0   0   ... |
| <b>c</b> | u   u   0   0   ... | u   1   1   1   ... |
| <b>d</b> | 1   1   1   1   ... | 1   1   1   1   ... |
| <b>e</b> | u   u   1   1   ... | u   u   u   u   ... |

Figure 1. Minimal models

In this picture, a **1** means the corresponding atoms has been derived, a **0** means its negation has been derived, and a **u** means that neither the atoms nor its negation have been derived. The model **M** corresponds to the extension  $Cn(\{a, b, \neg c, d, e, b \rightarrow \neg c\})$ , and the model **N** corresponds to the extension  $Cn(\{a, \neg b, c, d, b \rightarrow \neg c\})$ . The reader familiar with default logic may check that these are (the only) extensions of  $\langle W, D \rangle$ . We will use this example in Section 5, where an example trace of the generic reasoning system is given. In this same fashion one can translate a logic program to a temporal theory, yielding the stable semantics.

### 3 A Specification Framework for Compositional Systems

In the framework DESIRE knowledge of

- (1) a task hierarchy,
- (2) information exchange,
- (3) sequencing of tasks,
- (4) task delegation,
- (5) knowledge structures

are explicitly modelled and specified. Each of these types of knowledge is discussed below.

#### 3.1 Task Composition

To model and specify composition of tasks, knowledge of the following types is required:

- o a *task hierarchy*,
- o information a task requires as *input*,
- o information a task produces as a *result* of task performance
- o *meta-object* relations between tasks

Within a task hierarchy *composed* and *primitive* tasks are distinguished: in contrast to primitive tasks, composed tasks are tasks for which sub-tasks are identified. Sub-tasks, in turn, can be either composed or primitive. Tasks are directly related to components: composed tasks are specified as composed components and primitive tasks as primitive components. Primitive components are executed by a simple classical deduction system, based on the inference relation chaining (modus ponens and conjunction introduction).

Information required/produced by a task is defined by *input* and *output signatures* of a component. The signatures used to name the information are defined in a predicate logic with a hierarchically ordered sort structure (*order-sorted predicate logic*). Units of information are represented by the ground *atoms* defined in the signature.

The role information plays within reasoning is indicated by the level of an atom within a signature: different (meta)levels may be distinguished. In a two-level situation the lowest level is termed *object level information*, and the second level *meta-level information*. Meta-level information contains information about object level information and reasoning processes; for example, for which atoms the values are still unknown (*epistemic information*). Often more than two levels of information and reasoning occur, resulting in meta-meta-... information and reasoning.

### 3.2 Information Exchange Between Tasks

Information exchange between tasks is specified as *information links* between components. Each information link relates output of one component A to input of another component B, by specifying which truth value of a specific output atom of A is linked with which truth value of a specific input atom of B. Atoms can be renamed: each component can be specified in its own language, independent of other components. The conditions for activation of information links are explicitly specified as task control knowledge: a kind of knowledge of the sequencing of tasks.

### 3.3 Sequencing of Tasks

Task sequencing is explicitly specified within components as *task control knowledge*. Task control knowledge includes not only knowledge of which tasks should be activated when and how, but also knowledge of the goals associated with task activation and the extent to which goals should be derived. These aspects are specified as component and link activation together with task control foci and extent to define the component's goals. Components are, in principle, black boxes to the task control of an encompassing component: task control is based purely on information about the success and/or failure of component reasoning. Reasoning of a component is considered to have been successful with respect to its task control focus if it has reached the goals specified by this task control focus to the extent specified (e.g., any or every).

### 3.4 Delegation of Tasks

During knowledge acquisition a task as a whole is modelled. In the course of the modelling process decisions are made as to which tasks are (to be) performed by which *agent*. This process results in the delegation of tasks to the parties involved in task execution.

### 3.5 Knowledge Structures

During design an appropriate structure for domain knowledge must be devised. The meaning of the concepts used to describe a domain and the relations between concepts and groups of concepts, are determined. Concepts are required to identify objects distinguished in a domain (domain-oriented ontology), but also to express the methods and strategies employed to perform a task (task-oriented ontology). Within primitive components concepts and relations between concepts are defined in *hierarchies* and *rules* (based on *order-sorted predicate logic*). In a specification references to appropriate knowledge structures (specified elsewhere) suffice; compositional knowledge structures are composed by reference to other knowledge structures.

## 4 A Generic Compositional Nonmonotonic Reasoning System

The nonmonotonic reasoning task of finding minimal models of a reasoning theory is modelled as a composition of two subtasks. The first subtask generates all possible continuations of the reasoning trace, and the second subtask selects a continuation. Within the first task, first it is determined which rules are applicable. Applicable rules are the rules

$$\alpha \wedge \beta \wedge \varphi \wedge \psi \rightarrow \mathbf{G}\gamma,$$

where

$$\alpha = \wedge\{ \mathbf{H}_0\epsilon \mid \epsilon \in \mathbf{A} \} \text{ for a set of propositional formulae } \mathbf{A}.$$

$$\beta = \wedge\{ \neg \mathbf{H}_0\delta \mid \delta \in \mathbf{B} \} \text{ for a set of propositional formulae } \mathbf{B}.$$

$$\varphi = \wedge\{ \neg \mathbf{F}\theta \mid \theta \in \mathbf{C} \} \text{ for a set of propositional formulae } \mathbf{C}.$$

$$\psi = \wedge\{ \mathbf{C}\zeta \mid \zeta \in \mathbf{D} \} \text{ for a set of propositional formulae } \mathbf{D}.$$

$\gamma$  is a propositional formula.

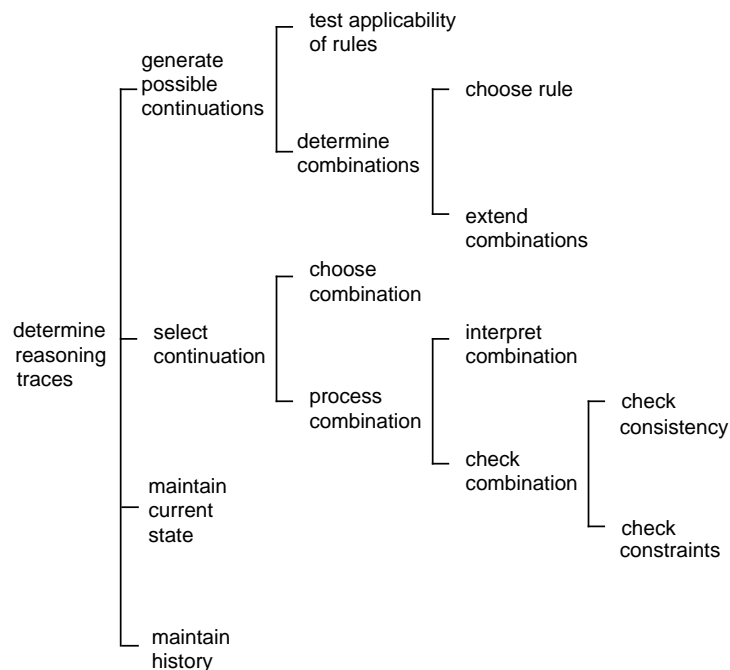
for which the conditions that refer to the past and present (i.e.,  $\alpha, \beta, \psi$ ) are fulfilled. Next, for each of the applicable rules, for the future-directed conditions  $\varphi$  two possibilities are generated:

- either the conditions that refer to the future will be fulfilled in the reasoning trace that is generated, or
- these future-directed conditions will not be fulfilled.

In the first case the rule will contribute its conclusion to the reasoning process and we have to make sure in the future that the future conditions were indeed fulfilled (we add *constraints* to ensure this). In the second case no explicit contribution will be made by the rule. However, in this second case, by the subsequent generation of the reasoning trace it will have to be guaranteed that the future-directed conditions indeed will be

violated (and we again add constraints to ensure this). In this sense an implicit effect on the reasoning trace occurs: all traces that do not contradict these conditions will be rejected (see [ET96b]). Note that we do not try to *execute* the future condition; we merely guess whether it will be fulfilled. In this respect these rules are not of the form "declarative past implies imperative future" of [Ga89].

The design of the compositional nonmonotonic reasoning system has been specified in DESIRE, according to the five types of knowledge discussed in Section 3. Five levels of abstraction are distinguished in the task hierarchy (see Figure 2).

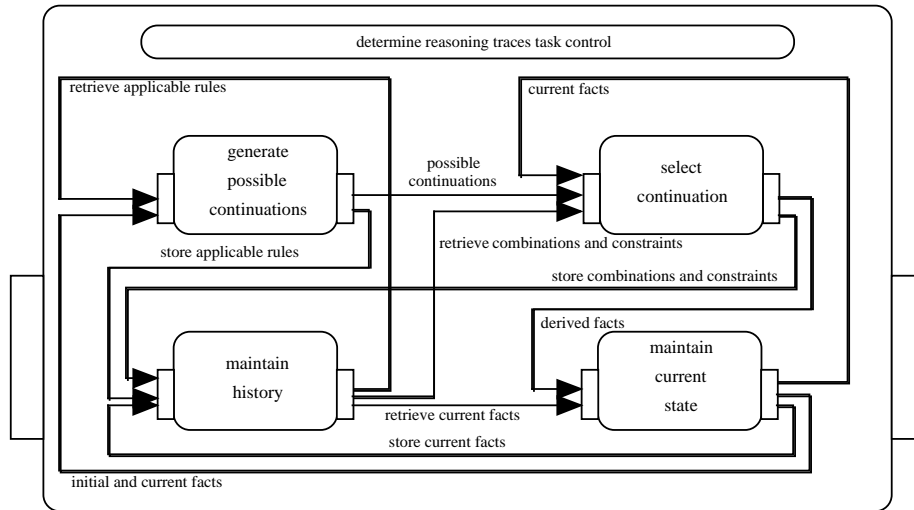


**Figure 2 . Complete task hierarchy of the system**

#### 4.1 Top Level of the System

At the highest level the system consists of four components. During the reasoning process, in the component `maintain_current_state` the facts are represented that have been derived. The component `maintain_history` stores relevant aspects of the reasoning process in order to perform belief revision if required. The reasoning is performed by the component `generate_possible_continuations`, which generates the possible next steps of the reasoning trace and `select_continuation` which chooses one of these possibilities. By this selection the actual next step in the reasoning trace is determined. In Figure 3 the information exchange at the top level of the system is depicted. (In this picture and the following ones, we have left out some links which go from a

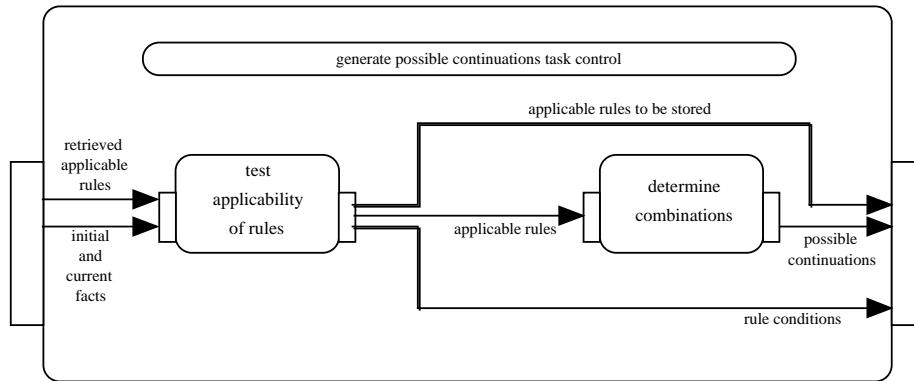
component to itself. For instance, such links are sometimes needed to model a closed-world assumption.)



**Figure 3 . Information exchange at the top level of the system**

#### 4.2 Generate Possible Continuations

Within the component `generate_possible_continuations` two sub-components are distinguished: `test_applicability_of_rules` and `determine_combinations`; see Figure 4 for the information flow at this level.

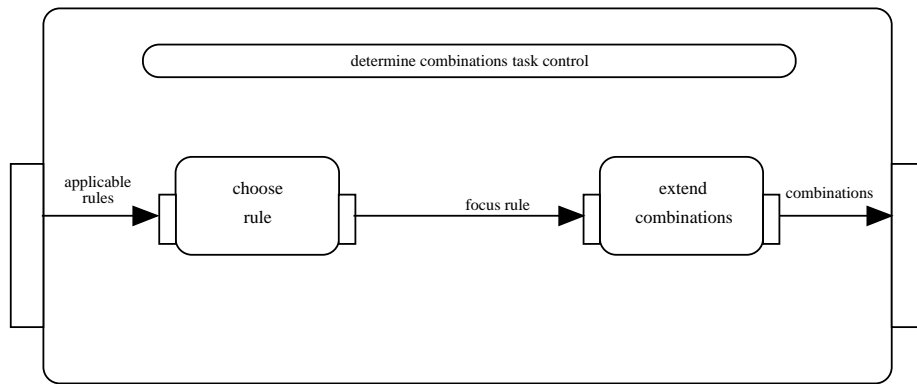


**Figure 4. Information exchange within generate possible continuations**

The former component is primitive and determines the rules that are applicable in the current state of the reasoning process. Its knowledge base consists of rules, for example, of the form:

|   |   |
|---|---|
| <b>if</b> posH0condition(L: Literals, R: Rules) | <b>if</b> currentcondition(L: Literals, R: Rules) |
| <b>and not</b> initial_fact(L: Literals)        | <b>and not</b> current_fact(L: Literals)          |
| <b>then not</b> applicable(R: Rules)            | <b>then not</b> applicable(R: Rules)              |

By application of a form of the Closed World Assumption, the applicable rules are derived. The second component determines for each of the applicable rules two possibilities: it is assumed that either (+) the conditions of the rule that refer to the future will be fulfilled in the reasoning trace that is generated, or (-) these future-directed conditions will not be fulfilled.



**Figure 5. Information exchange within determine combinations**

The component `determine_combinations` is rather simple (see Figure 5). The applicable rules are treated one by one and combinations are constructed. Both subcomponents are primitive. The knowledge base of the first subcomponent just consists of one rule:

```
if applicable(R: Rules)
and not covered(R: Rules)
then infocus(R: Rules)
```

The knowledge base of the second subcomponent consists of the following two rules:

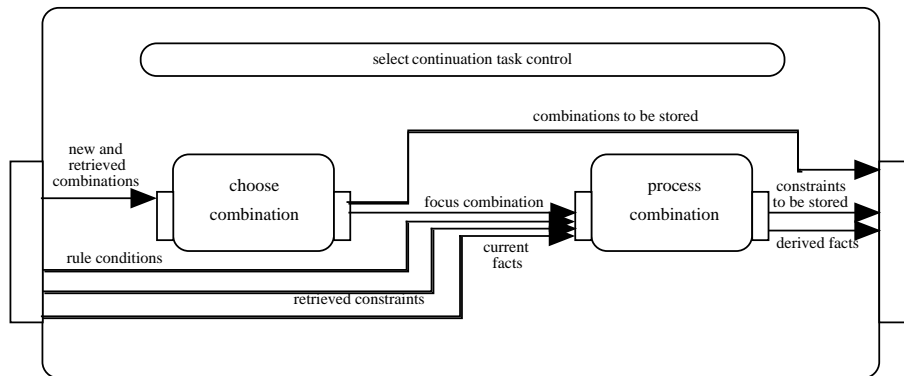
|  |  |
|--|--|
| <b>if</b> old_combination(C: combinations) | <b>if</b> old_combination(C: combinations) |
| <b>and</b> infocus(R: rules)               | <b>and</b> infocus(R: rules)               |
| <b>and</b> futuredependent(R: rules)       | <b>then</b> new_combination(app(C:         |
| <b>then</b> new_combination(app(C:         | combinations, tup(R: rules, pos)));        |
| combinations, tup(R: rules, neg)));        |  |

These rules build new combinations from old combinations and the rule in focus. The predicate `app` (for `append`) is used to build up a list, and the predicate `tup` (for `tuple`) is

used to make tuples. Only rules with a part that refers to the future (futuredependent) are allowed *not* to be applied (meaning that  $\text{tup}(R: \text{rules}, \text{neg})$  may occur in a combination).

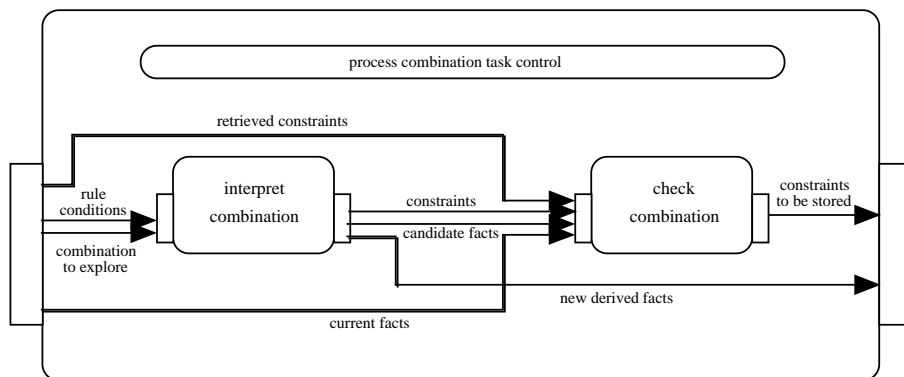
### 4.3 Select Continuation

Within the component `select_continuation`, focus combinations are chosen one by one and processed (see Figure 6).



**Figure 6. Information exchange within select continuation**

Processing a combination is performed by first making an interpretation of the information represented by a combination, and subsequently checking on consistency against the current facts and checking the constraints (see Figures 7 and 8).



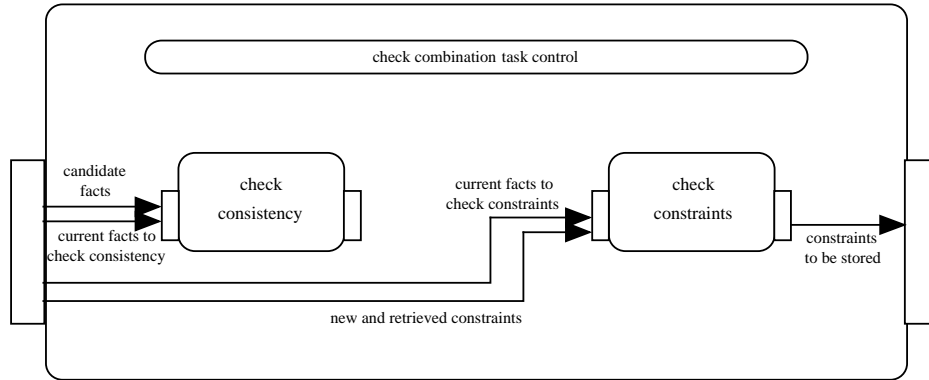
**Figure 7. Information exchange within process combinations**

The knowledge base of the primitive component `check_consistency` consists of three rules, an example of which is:

```

if next(A: Atoms)
and current(neg(A: Atoms))
then inconsistency

```



**Figure 8. Information exchange within check combinations**

## 5 Example Trace

In this section we will give an example of the execution of our generic reasoning system. The theory of reasoning is the translation of the example default theory of Section 2. When this theory is given to the generic reasoning system, the following will happen (in this list, on the left we will indicate the literals that are derived, on the right the component in which it is derived, in brackets; not everything derived in every component is listed and no information links are listed):

|   |                               |
|---|-------------------------------|
| - a, d  | (maintain_current_state)      |
| - applicable(r3), applicable(r4)  | (test_applicability_of_rules) |
| - new_combination(app(app(nil, tup(r3, pos)), tup(r4, pos))),<br>new_combination(app(app(nil, tup(r3, neg)), tup(r4, pos))),<br>new_combination(app(app(nil, tup(r3, pos)), tup(r4, neg))),<br>new_combination(app(app(nil, tup(r3, neg)), tup(r4, neg))) | (determine_combinations)      |
| - selectedcombination(app(app(nil, tup(r3, pos)), tup(r4, pos)))  | (choose_combination)          |
| - next(c), next(b)  | (interpret_combination)       |
| - c, b  | (maintain_current_state)      |
| - applicable(r1), applicable(r2)  | (test_applicability_of_rules) |
| - new_combination(app(app(nil, tup(r1, pos)), tup(r2, pos)))  | (determine_combinations)      |
| (since these rules do not refer to the future, only one combination will be generated, in which both rules are applied)   |                               |
| - selectedcombination(app(app(nil, tup(r1, pos)), tup(r2, pos)))  | (choose_combination)          |
| - next(neg(b)), next(neg(c))  | (interpret_combination)       |
| - inconsistency   | (check_consistency)           |

(this uses the rule **if next(neg(A: atoms)) and current(A: atoms) then inconsistency** with **next(neg(b))** and **current(b)**; now `maintain_history` will be invoked with target set **retrieve**. Then `choose_combination` will fail, since there was only one combination at this point - **app(app(nil, tup(r1, pos)), tup(r2, pos))** - so another **retrieve** is performed.)

|   |                         |
|---|-------------------------|
| - <b>selectedcombination(app(app(nil, tup(r3, neg)), tup(r4, neg)))</b> | (choose_combination)    |
| - <b>sometimes_true(neg(c), r4), (sometimes_true(neg(b)))</b>           | (interpret_combination) |

(after `maintain_current_state`, `test_applicability_of_rules` will find no more applicable rules).

|                        |                     |
|------------------------|---------------------|
| - <b>incorrect(r4)</b> | (check_constraints) |
|------------------------|---------------------|

(this is derived from the constraint **sometimes\_true(neg(c), r4)** and **not(current(neg(c)))**; a **retrieve** will occur)

|   |                               |
|---|-------------------------------|
| - <b>selectedcombination(app(app(nil, tup(r3, pos)), tup(r4, neg)))</b>   | (choose_combination)          |
| - <b>sometimes_true(neg(c), r4), next(b), never_true(neg(b))</b>  | (interpret_combination)       |
| - <b>b</b>  | (maintain_current_state)      |
| - <b>applicable(r1), applicable(r5)</b>   | (test_applicability_of_rules) |
| - <b>new_combination(app(app(nil, tup(r5, pos)), tup(r1, pos))),<br/>new_combination(app(app(nil, tup(r5, neg)), tup(r1, pos)))</b> | (determine_combinations)      |

(**r1** has to be applied (it does not refer to the future), but for **r5** there is a choice)

|   |                          |
|---|--------------------------|
| - <b>selectedcombination(app(app(nil, tup(r5, pos)), tup(r1, pos)))</b> | (choose_combination)     |
| - <b>next(neg(c)), next(e), never_true(c)</b>                           | (interpret_combination)  |
| - <b>neg(c), e</b>  | (maintain_current_state) |

(no more applicable rules are found by `test_applicability_of_rules`, so `check_constraints` is invoked, which finds no violated constraints)

- at this point `user_interaction` will display the window of figure 9.

The minimal model displayed corresponds to the extension based on the literals {**a, b, ¬ c, d, e**}. When the user clicks "no", the execution will stop. Otherwise a **retrieve** will be performed by `maintain_history`, after which `choose_combination` will choose the combination in which **r5** is not applied (but **r1** is). This will eventually lead to a violated constraint, so another **retrieve** will occur. Then `choose_combination` will take the last combination (for **r3** and **r4**), in which **r3** is not applied and **r4** is. Ultimately, the second minimal model will be found and displayed (corresponding to the literals {**a, ¬ b, c, d**}). If the user wants to search for another minimal model, none will be found, and a message indicating this will be displayed, after which execution ends.

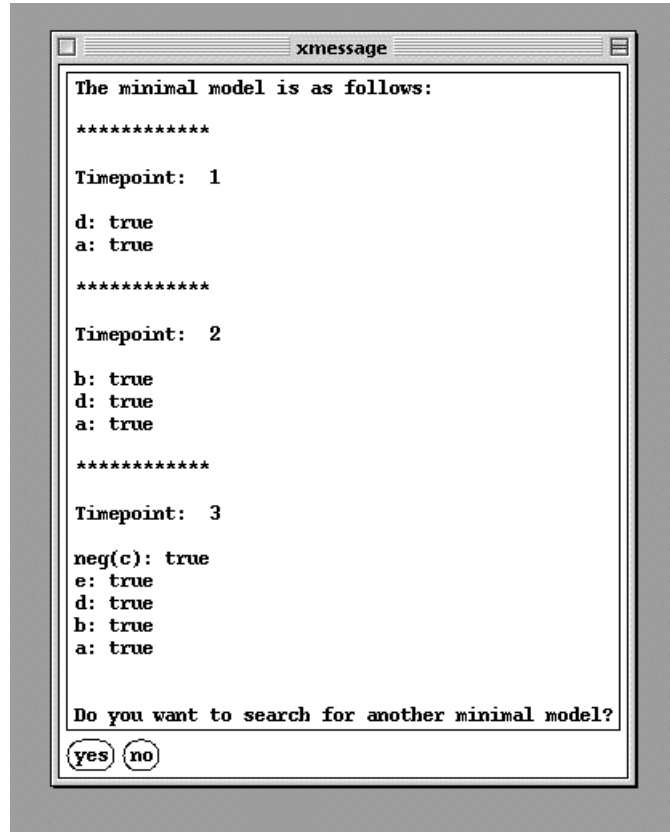


Figure 9. User interaction

## 6 Discussion

In this paper the framework DESIRE for the design of compositional reasoning systems and multi-agent systems was applied to build a generic nonmonotonic reasoning system. The outcome is a general reasoning system that can be used to model different nonmonotonic reasoning formalisms (see for instance [ET94]), and that can be executed by a generic execution mechanism. The main advantages of using DESIRE (compared to a direct implementation in a programming language such as PROLOG) are:

- o the design is generic and has a transparent compositional structure; it is easily readable, modifiable and reusable. The generic nonmonotonic reasoning system is easily usable as a component in agents that are specified in DESIRE. For example, if an agent is designed that has default knowledge, then the generic reasoning system can be included as one of the agent's components and the

representation of the agent's default knowledge can be translated to the temporal representation of the generic reasoning system.

- o explicit declarative specification of both the static and dynamic aspects of the nonmonotonic reasoning processes, including their control. The current system generates one or all reasoning traces that are possible without any specific guidance. However, a number of approaches to nonmonotonic reasoning have been developed that in addition use explicit knowledge about priorities between nonmonotonic rules (e.g., [Br94], [TT92]). This knowledge can easily be incorporated within the component `select_continuation`, in particular within its subcomponent `choose_combination`.

Even though the efficiency of the reasoning system can be improved (by adding knowledge to prevent generating possible continuations that can easily be seen to violate constraints, by adding heuristic knowledge in the selection of continuations, etc.), implementations for specific nonmonotonic formalisms (e.g. for default logic, see [Ni95], [CMT96], or [RS94]) can often be made more efficient. In general they lack, however, the ability to handle different kinds of nonmonotonic reasoning, and the extendibility of our approach. Also, they cannot handle dynamic queries (e.g., has a literal been derived before time point 3).

## References

- [BDJT97] Brazier, F.M.T., Dunin-Keplicz, B., Jennings, N.R., Treur, J. : "DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework", *International Journal of Cooperative Information Systems* **6** (1997) , Special Issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems (M. Huhns, M. Singh, eds.), in press. Shorter version in: V. Lesser (ed.), *Proc. of the First International Conference on Multi-Agent Systems*, ICMAS-95, MIT Press, 1995, pp. 25-32
- [BTWW95] Brazier, F.M.T., Treur, J., Wijngaards, N.J.E., Willems, M.: "Formal Specification of Hierarchically (De)composed Tasks", in: B.R. Gaines, M.A. Musen (eds.), *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, KAW'95, SRDG Publications, 1995, pp. 25/1 - 25/20
- [Br94] Brewka, G.: "Adding Priorities and Specificity to Default Logic", in: C. MacNish, D. Pearce, L.M. Pereira (eds.), *Logics in Artificial Intelligence, Proceedings of the 4th European Workshop on Logics in Artificial Intelligence*, JELIA '94, Lecture Notes in Artificial Intelligence **838**, Springer-Verlag, 1994, pp. 247-260
- [CMT96] Cholewinski, P., Marek, V.W., Truszczyński, M.: "Default Reasoning System DeReS", in: *Proceedings 5th International Conference on Principles of Knowledge Representation and Reasoning*, KR-96, Morgan Kaufmann, 1996.
- [ET93] Engelfriet, J., Treur, J.: "A Temporal Model Theory for Default Logic", in: M. Clarke, R. Kruse, S. Moral (eds.), *Proceedings 2nd European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, ECSQARU '93, Lecture Notes in Computer Science **747**, Springer-Verlag, 1993, pp. 91-96. Extended version in *Journal of Logic, Language and Information*, **7**, 1998, pp. 369-388.
- [ET94] Engelfriet, J., Treur, J. : "Temporal Theories of Reasoning", in: C. MacNish, D. Pearce, L.M. Pereira (eds.), *Logics in Artificial Intelligence, Proceedings of the 4th European Workshop on Logics in Artificial Intelligence*, JELIA '94, Lecture Notes in Artificial

Intelligence **838**, Springer-Verlag, 1994, pp. 279-299. Also in: *Journal of Applied Non-Classical Logics* **5**(2), 1995, pp. 239-261

- [ET96a] Engelfriet, J., Treur, J. : "Specification of Nonmonotonic Reasoning", in: D.M. Gabbay, H.J. Ohlbach (eds.), *Practical Reasoning, International Conference on Formal and Applied Practical Reasoning, FAPR'96*, Lecture Notes in Artificial Intelligence **1085**, Springer-Verlag, 1996, pp. 111-125. Extended version in *Journal of Applied Non-Classical Logics*, **10**, 2000, pp. 7-27
- [ET96b] Engelfriet, J., Treur, J. : "Executable Temporal Logic for Nonmonotonic Reasoning", to appear in *Journal of Symbolic Computation*, Special issue on Executable Temporal Logic, vol. 22, 1996, pp. 615-625.
- [FO95] Fisher, M., Owens, R. (eds.) : *Executable Modal and Temporal Logics, Proceedings of the IJCAI'93 workshop*, Lecture Notes in Artificial Intelligence **897**, Springer-Verlag, 1995
- [Ga89] Gabbay, D. : "The Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems", in: B. Banieqbal, H. Barringer, A. Pnueli (eds.), *Temporal Logic in Specification*, Lecture Notes in Computer Science **398**, Springer-Verlag, pp. 409-448
- [LPT92] Langevelde, I.A. van, Philipsen, A.W., Treur, J. : "Formal Specification of Compositional Architectures", in: B. Neumann (ed.), *Proceedings of the 10th European Conference on Artificial Intelligence*, ECAI'92, John Wiley & Sons, pp. 272-276
- [Ni95] Niemelä, I. : "Towards Efficient Default Reasoning", in: *Proceedings 14th IJCAI*, Morgan Kaufmann, 1995, pp. 312-318
- [Ni96] Niemelä, I. : "Implementing Circumscription Using a Tableau Method", in: W. Wahlster (ed.), *Proceedings 12th European Conference on Artificial Intelligence*, ECAI'96, John Wiley & Sons, pp. 80-84
- [RS94] Risch, V., Schwind, C.B.: "Tableau-Based Characterization and Theorem Proving for Default Logic", *Journal of Automated Reasoning* **13**: 223-242, 1994
- [SB96] Schaub, T., Brüning, S. : "Prolog technology for default reasoning (An abridged report)", in: W. Wahlster (ed.), *Proceedings 12th European Conference on Artificial Intelligence*, ECAI'96, John Wiley & Sons, pp. 105-109
- [TT92] Tan, Y.H., Treur, J. : "Constructive Default Logic and the Control of Defeasible Reasoning", in: B. Neumann (ed.), *Proceedings of the 10th European Conference on Artificial Intelligence*, ECAI'92, John Wiley & Sons, pp. 299-303