

# Compositional Verification of Knowledge-based Task Models and Problem Solving Methods

Frank Cornelissen, Catholijn M. Jonker, Jan Treur

Vrije Universiteit Amsterdam, Department of Artificial Intelligence

De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

URL: <http://www.cs.vu.nl/~{jonker,treur}>

Email: {jonker,treur}@cs.vu.nl

**Abstract** In this paper a compositional verification method for task models and problem solving methods for knowledge-based systems is introduced. Required properties of a system are formally verified by deriving them from assumptions that themselves are properties of sub-components, which in their turn may be derived from assumptions on sub-sub-components, and so on. The method is based on properties that are formalised in terms of temporal semantics; both static and dynamic properties are covered. The compositional verification method imposes structure on the verification process. By the possibility to focus at one level of abstraction (information and process hiding), compositional verification provides transparency and limits the complexity per level. Since verification proofs are structured in a compositional manner, they can be reused in case of reuse of models or modification of an existing system. The method is illustrated for a generic model for diagnostic reasoning.

**Keywords** Compositional verification, knowledge-based systems, diagnostic reasoning model, formal compositional modelling.

## 1 Introduction

When designing complex knowledge-based systems, it is often hard to guarantee that the specification of a system that has been designed actually fulfills the needs, i.e., whether it satisfies the design requirements. Especially for critical applications, for example in aerospace domains, there is a need to

prove that the designed system will have certain properties under certain conditions (assumptions). While developing a proof of such properties, the assumptions that define the bounds within which the system will function properly are generated. It often takes a lot of effort to perform - for each application - such a verification in the form of a formal analysis of when the system will function properly. For this reason verification is usually left out of the development process.

In structured development methods for knowledge-based systems such as CommonKADS (Schreiber et al., 2000) and DESIRE (Brazier et al., 1999, 2000), during design an important role is played by generic models for tasks, i.e., problem solving methods, and domain specific instantiations of such generic models. The use of generic models has the advantage that all effort spent on such a model is reusable in all applications of the model. Therefore, verification efforts related to generic models are much more efficient, as they can be performed for one generic model but applied in a large class of applications. The idea is that within the library of generic models, the formal analysis is included. When a generic model is applied, from the formal analysis it can be read which assumptions on the domain knowledge have to be fulfilled in order to obtain a proper functioning application.

In this paper, in Section 3, a structured verification method for compositional task models and problem solving methods (compositional generic models, for short) is introduced, called *compositional verification*. Roughly spoken, the requirements of the whole system are formally verified by deriving them from assumptions that themselves are properties of sub-components, which in their turn may be derived from assumptions on sub-sub-components, and so on. This process ends when primitive components are reached: components that are not composed, but specified by means of a knowledge base (or any other means).

The method introduced here is illustrated for a compositional generic (process) model for diagnostic reasoning. For this generic model, requirements are formulated (both the required static and dynamic properties), and a compositional specification is introduced in Section 4. The compositional specification is based on a process composition that specifies how the main process is composed of the processes hypothesis determination and hypothesis validation, and how the sub-process hypothesis validation is composed of the processes observation determination, observation execution and hypothesis evaluation. The compositional specification itself is expressed in the modelling framework of DESIRE, briefly described in Section 2. The

application of the compositional verification method to the diagnostic process model is presented in Section 5 (top level of the composition), Section 6 (lower level), and Section 7 (primitive components). The Appendix shows some of the details of the proofs.

## 2 Compositional Modelling of Knowledge-Based Systems

The generic diagnostic process model described in this paper has been modelled using the compositional design method DESIRE for knowledge-based systems and agent systems (DEsign and Specification of Interacting REasoning components; cf. (Brazier, Treur, Wijngaards and Willems, 1999). A number of compositional generic models for agents and tasks have been developed and used for a number of applications. The architectures upon which compositional specifications are based are the result of analysis of the tasks performed. Process compositions for a task include specifications of interaction between processes at each process abstraction level within a task. Models specified within DESIRE are defined according to the following compositional structure, which will be briefly discussed in Sections 2.1 to 2.3:

- process composition
  - identification of processes at different abstraction levels and task delegation
  - process composition relation: information exchange and task control
- knowledge composition
  - identification of information types and knowledge bases
  - knowledge composition relation between information types and knowledge bases
- relation between process and knowledge composition

## 2.1 Process Composition

Process composition identifies the relevant processes at different levels of (process) abstraction, and describes how a process can be defined in terms of (is composed of) lower level processes.

### 2.1.1. Identification of Processes at Different Levels of Abstraction

Processes can be described at different levels of abstraction; for example, the process of the task as a whole, processes defined by specific sub-tasks, and so on. The identified processes are modelled as *components*. For each process the *input* and *output information types* are modelled. The identified levels of process abstraction are modelled as *abstraction/specialisation relations* between components: components may be *composed* of other components or they may be *primitive*. Primitive components may be either reasoning components (i.e., based on a knowledge base), or, components capable of performing tasks such as calculation, information retrieval, optimisation. These levels of process abstraction provide process hiding at each level.

**2.1.2. Composition of Processes** The way in which processes at one level of abstraction are composed of processes at the adjacent lower abstraction level is called *process composition*. This composition of processes is described by a specification of *information links*, i.e., the possibilities for information exchange between processes (*static view* on the composition), and a specification of *task control knowledge* used to control processes and information exchange (*dynamic view* on the composition). An essential element of the process composition is the set of information links that relate information at a level of process abstraction to the next higher level (called *mediating links*). The specification of these links (i.e., a kind of table) defines exactly which information can be exchanged from the lower to the higher level, and which information can be exchanged from the higher to the lower level of process abstraction.

## 2.2 Knowledge Composition

Knowledge composition identifies the knowledge structures at different levels of (knowledge) abstraction, and describes how a knowledge structure can be defined in terms of lower level knowledge structures. The knowledge

abstraction levels may correspond to the process abstraction levels, but this is often not the case.

**2.2.1. Identification of Knowledge Structures at Different Abstraction Levels** The two main structures used as building blocks to model knowledge are: *information types* and *knowledge bases*. Knowledge structures can be identified and described at different levels of abstraction. At higher levels details can be hidden. An *information type* defines an ontology (lexicon, vocabulary) to describe objects or terms, their sorts, and the relations or functions that can be defined on these objects. Information types can logically be represented in order-sorted predicate logic. A *knowledge base* defines a part of the knowledge that is used in one or more of the processes. Knowledge is represented by formulae in order-sorted predicate logic, which can be normalised by a standard transformation into rules.

**2.2.2. Composition of Knowledge Structures** Information types can be composed of more specific information types, following the principle of compositionality discussed above. Similarly, knowledge bases can be composed of more specific knowledge bases. The compositional structure is based on the different levels of knowledge abstraction distinguished, and results in information and knowledge hiding.

## 2.3 Relation between Process and Knowledge Composition

Each process in a process composition uses knowledge structures. Which knowledge structures are used for which processes is defined by the relation between process composition and knowledge composition.

The semantics of the modelling language are based on temporal logic (cf., Brazier, Treur, Wijngaards and Willems, 1999). Design is supported by graphical tools within the DESIRE software environment. Translation to an operational system is straightforward; the software environment includes implementation generators with which formal specifications can be translated into executable code. DESIRE has been successfully applied to design both single agent and multi-agent knowledge-based systems. Over the years, DESIRE has been used to design prototype knowledge-based systems for a wide variety of applications, often in projects paid by industry. For exam-

ple, knowledge-based systems for diagnosis of chemical (Nylon production) processes (Brazier et al., 2000), biochemical process control for penicillin production (Jonker and Treur, 1999), ecological monitoring (Beusekom et al., 1999), and design of sets of measures for environmental policy making (Brazier et al., 1996). Moreover, prototype multi-agent applications have been developed using DESIRE for, among others, distributed work flow scheduling for a Call Center (Brazier et al., 1999), negotiation for load balancing of electricity use (Brazier et al., 1998), multi-attribute negotiation in Electronic Commerce (Jonker and Treur, 2001), and information brokering (Jonker and Vollebregt, 2000). All of these applications have been designed in a compositional manner using DESIRE. However, only a few of them have been verified. The subsequent section describes a method to support such compositional verification.

### 3 Compositional Verification

The purpose of verification is to prove that, under a certain set of assumptions, a system will adhere to a certain set of properties, for example the design requirements. In our approach, this is done by a mathematical proof (i.e., a proof in the form mathematicians are accustomed to do) that the specification of the system together with the assumptions implies the properties that it needs to fulfill.

#### 3.1 The Compositional Verification Method

A compositional system can be viewed at different levels of abstraction. Viewed from the top level, denoted by  $L_0$ , the complete system is one component  $D$ , with interfaces, whereas internal information and processes are hidden (information and process hiding). At the next lower level of abstraction, the top level component  $D$  can be viewed as a composition of sub-components, information links, and task control. The compositional verification method takes into account this compositional structure. The primitive reasoning components can be verified using more traditional verification methods such as described in (Treur and Willems, 1994; Leemans, Treur and Willems, 1993). Verification of a composed component is done using properties of the sub-components it embeds and the task control knowledge. This introduces a form of compositionality in the verification process: the proof that a certain

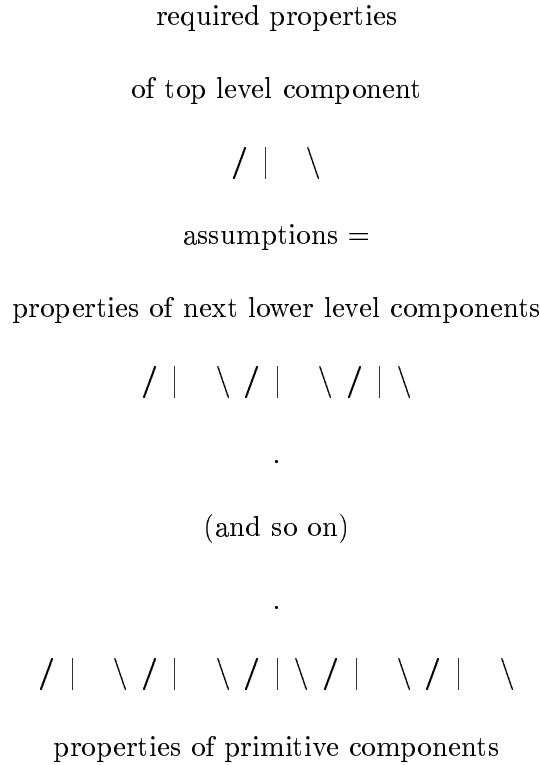


Figure 1: Hierarchical relations between properties in compositional verification

component adheres to a set of properties depends on the (assumed) properties of its sub-components. The assumptions under which the component functions properly, are the properties to be proven for its sub-components. This implies that properties at different levels of abstraction are involved in the verification process. These properties have hierarchical logical relations in the sense that at each level a property is logically implied by (a conjunction of) the lower level properties that relate to it in the hierarchy (see Fig. 1).

Often these properties are not given at the start of the verification process. Actually, the process of verification has two main aims:

- to find the properties
- to prove the higher level properties from lower level properties

The verification proofs that connect one abstraction level with the other are compositional in the following manner: any proof relating level  $i$  to level  $i+1$  can be combined with any proof relating level  $i-1$  to level  $i$ , as long as the same properties at level  $i$  are involved. This means, for example, that the whole compositional structure beneath level  $i$  can be replaced by a completely different design as long as the same properties at level  $i$  are achieved. After such a modification the proof from level  $i$  to level  $i-1$  can be reused; only the proof from level  $i+1$  to level  $i$  has to be adapted. In this sense the method supports reuse of verification. The *compositional verification method* can be formulated in more detail as follows:

**A. Verifying one abstraction level against the other**

For each abstraction level the following top-down procedure is followed:

1. Determine which properties are of interest (for the higher level).
2. Determine assumptions (at the lower level) that guarantee these properties.
3. Prove the properties on the basis of these assumptions.

**B. Verifying a primitive component**

For primitive knowledge-based components - with relatively small knowledge bases - a number of techniques exist in literature, see for example (Treur, Willems 1994; Leemans, Treur, Willems 1993). For primitive non-knowledge-based components, such as data bases, or neural networks, or optimization algorithms, verification techniques can be used that are especially tuned for that type of component.

**C. The overall verification process**

To verify the complete system

1. Determine the properties that are desired for the whole system.
2. Apply the above procedure **A** iteratively until primitive components are reached. In the iteration the desired properties of abstraction level  $L_i$  are either:
  - those determined in step **A1**, if  $i = 0$ , or
  - the assumptions made for the higher level  $L_{i-1}$ , if  $i > 0$

### 3. Verify the primitive components according to **B**.

*Notes:*

- The results of verification are (1) properties and assumptions at the different abstraction levels, and (2) logical relations between the properties of different abstraction levels
- Both static and dynamic properties and connections between them are covered
- Both the determination of the properties and assumptions and the proofs of the logical relations are made by hand (in the style of mathematicians' work)
- Reuse of verification results is supported: refining an existed verified compositional model by further decomposition, leads to a verification of the refined system in which the verification structure of the original system can be reused.
- Process and information hiding limits the complexity of verification per abstraction level.
- A requirement to apply the compositional verification method described above is the availability of an explicit specification of how the system description at an abstraction level  $L_i$  is composed from the descriptions at the lower abstraction level  $L_{i+1}$ ; the compositional modelling framework DESIRE is an instance of a modelling framework that fulfills this requirement.
- In principle, a similar, bottom-up procedure, can be formulated as well.
- For any set of assumptions obtained in A., if it is required that it does not contain superfluous elements, for each assumption in the set an example may be constructed in which the assumption does not hold, whereas the other assumptions in the set hold and one or more of the properties fail. If for one of the assumptions no example is possible, then try to eliminate it.

## 3.2 Language and semantics used by the Compositional Verification Method

In principle, verification is always relative to semantics of the system descriptions that are verified, and uses a specific language to express properties. For the Compositional Verification Method put forward in this paper, these semantics are based on compositional information states and time steps in which transitions from one state to the other occur. In this sub-section a brief overview of these assumed semantics and the language used to express (behavioural) properties is given.

To obtain a formalisation of behavioural properties different variants of temporal logic can be used, depending on the type of properties to be expressed. For example, linear or branching time temporal logic are appropriate to specify various agent (system) behavioural properties. Examples of the use of specification languages based on such variants of temporal logic are described, for example, in (Fisher and Wooldridge, 1997; Manna and Pnueli, 1995). However, to specify adaptive properties of task models or problem solving methods such as ‘exercise improves skill’ as well, a comparison between different histories has to be explicitly expressed. This requires a form of temporal logic language which is more expressive than those allowing to refer at each time point only to one history. An example of such a more expressive formal language in which different histories can be compared is the *temporal trace language* introduced in (Jonker and Treur, 1998); its language and semantics are defined as follows.

An *information state*  $I$  of a system or system component  $D$  (e.g., the overall system, or an input or output interface of a component) is an assignment of truth values  $\{\text{true}, \text{false}, \text{unknown}\}$  to the set of ground atoms describing the information within  $D$ . An information state  $I$  for a component  $D$  is structured according to the compositional structure of  $D$ , i.e., within  $I$  the input and output states of  $D$  (i.e., three-valued truth assignments to the ground atoms based on the input resp. output information types) can be distinguished, the input and output states of sub-components, task control information state, et cetera; see also (Brazier, Treur, Wijngaards and Willems, 1999). The set of all possible information states of  $D$  is denoted by  $IS(D)$ . A *trace*  $M$  of  $D$  is a sequence (over the natural numbers) of information states  $(I^t)_{t \in \mathbb{N}}$  in  $IS(D)$ . Given a trace  $M$  of  $D$ , the information state of the input interface of a component  $C$  at time point  $t$  is denoted by

$$\text{state}_D(M, t, \text{input}(C)),$$

where  $state_D$  and  $input$  are function symbols. Analogously,

$state_D(M, t, output(C))$

denotes the information state of the output interface of component  $C$  at time point  $t$  within system (component)  $D$ . The task control information state of component  $C'$  at time point  $t$  of the component  $C$  is denoted by  $state_C(M, t, tc(C'))$ , where  $C'$  is either  $C$  or a sub-component of  $C$ .

The information states can be related to statements via the formally defined satisfaction relation  $\models$ , comparable to the Holds-predicate in the Situation Calculus. Differences from the Situation Calculus approach are, however, that we

1. use an infix notation for the  $\models$  predicate instead of a prefix notation,
2. refer to a trace and time point instead of a single state, and
3. can focus on part of the system.

Based on these statements, behavioural properties can be formulated in a formal manner in a sorted first-order predicate logic with sorts  $\mathbf{T}$  for time points,  $\mathbf{Traces}$  for traces and  $\mathbf{F}$  for state formulae, using quantifiers over time and the usual first order logical connectives such as  $\neg$ ,  $\wedge$ ,  $\vee$ ,  $\Rightarrow$ ,  $\forall$ ,  $\exists$ . A simple example of such a statement is the following (other, less trivial examples of can be found in Sections 5 and 6 below). Consider the following informally expressed property for the dynamics of a task as a whole:

*Each query of component C transferred to component D must be followed by an answer of component D after a certain time.*

In a structured, semiformal manner, this property can be reformulated (and detailed) as follows:

*if at some point in time  
     component C outputs: a query for D,  
 then at a later point in time  
     component D outputs: an answer for C*

Using the formal language introduced above the following temporal formalisation is made of this example property:

$\forall M, t, r$   
 $[ state(M, t, output(A)) \models query\_for\_from(q, D, C)$   
 $\Rightarrow \exists p, t1 > t \ state(M, t1, output(B)) \models answer\_for\_from(a, q, C, D) ]$

Here the statement

$$\text{state}(M, t, \text{output}(A)) \models \text{query\_for\_from}(r, D, C)$$

means that within trace  $M$  at time point  $t$  a statement  $\text{query\_for\_from}(r, D, C)$  occurs in the output interface of component  $C$ , i.e. has truth value true in the output state of  $C$ .

To connect neighbouring levels of abstraction in a verification proof for a DESIRE specification, the following elements can be used:

- the assumptions of the sub-components specified within component  $D$
- the interactions between the sub-components of  $D$  and / or the interfaces of  $D$
- the input / output information states of the sub-components of  $D$
- the task control information states of the sub-components of  $D$
- the information states of component  $D$
- the task control information states of component  $D$

## 4 A Generic Diagnostic Process Model

The generic diagnostic process model described in this section is based on the generic model for diagnostic reasoning processes analysed in (Treur, 1993). Diagnostic reasoning processes aim at the analysis of the cause of a disturbed situation. In most of these situations not all relevant observational facts are known in advance. The process of acquisition of additional (observation) information is an essential part of most diagnostic processes (Treur, 1993). Therefore, dynamics play an important role in diagnosis. In general diagnostic reasoning consists of a number of sub-processes such as the determination of hypothesis, the choice of applicable tests, the performance of tests and the interpretation of the test results. Strategic information such as the suitability of a test, likeliness of a hypothesis being true and the cost and effect of a test play an important role. Variants of this model have been applied in different domains of application, for example, for soil sanitation and diagnosis of nylon production processes in chemical industry; see (Brazier, Jonker, Treur, and Wijngaards, 2000) for more details of the development of these applications.

In this section the generic process model of diagnosis to be verified is described. In Sections 4.1 to 4.3 below the processes at the different levels of

process abstraction are given, each process component is described, and the interaction between the components.

## 4.1 Process Composition

The processes at different abstraction levels are given in Fig. 2. The process Hypothesis Determination generates hypotheses that are validated by the process Hypothesis Validation.

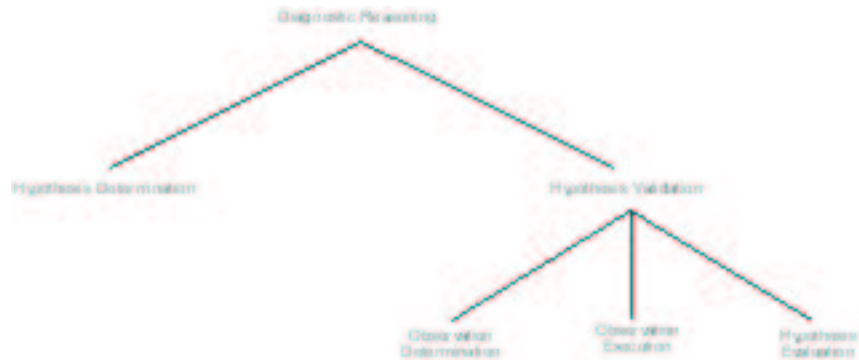


Figure 2: Processes at different abstraction levels in the diagnostic process model.

The two processes are described in the subsequent sections. In this section HYPs stands for the set of all (possible) hypotheses and OBS for the set of all (possible) observations.

## 4.2 Hypothesis Determination

The process Hypothesis Determination suggests hypotheses to be validated. This is done using information on which hypothesis have been rejected so far. The input and output interfaces are defined by

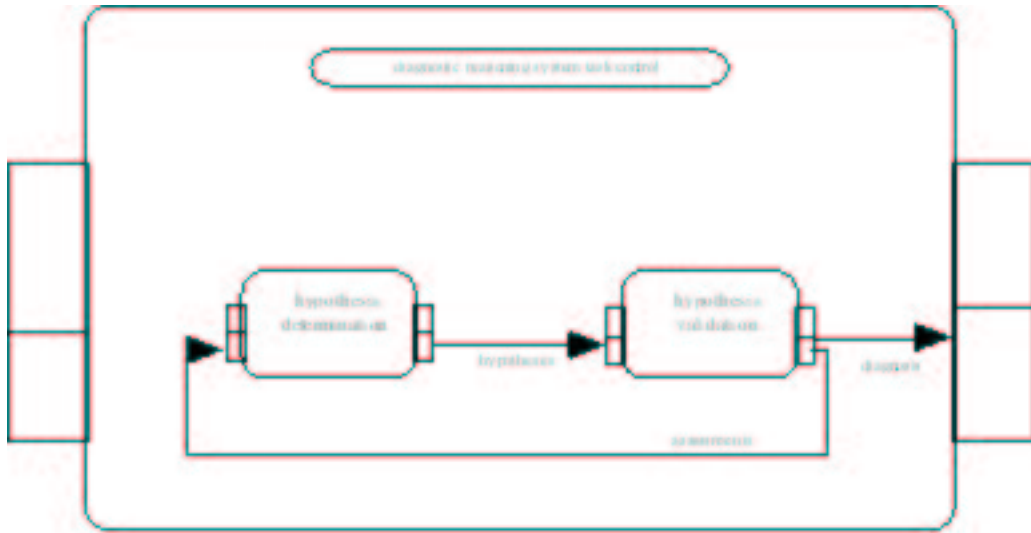


Figure 3: Process Composition: Top Level

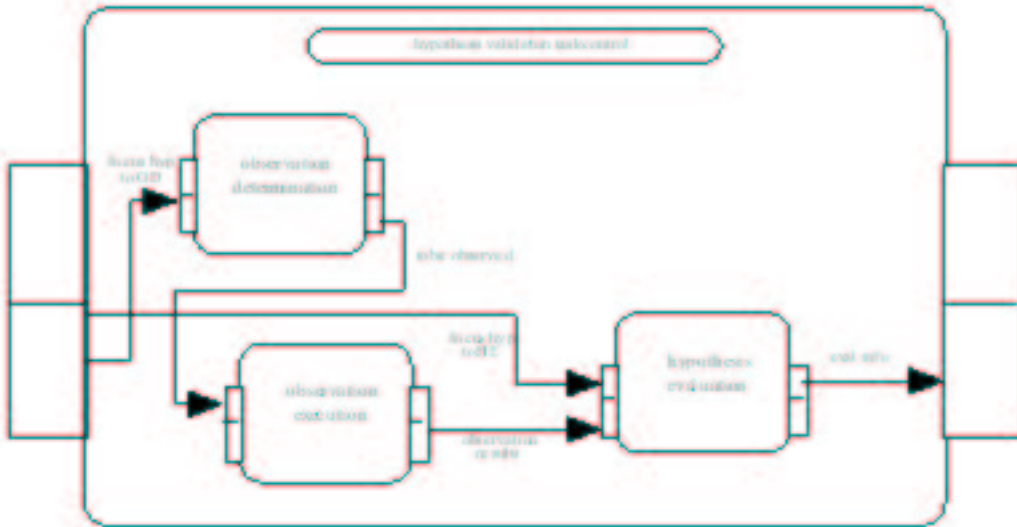


Figure 4: Process Composition: Hypothesis Validation

*input atoms*    rejected(h), confirmed(h) : h ∈ HYPS  
*output atoms*    focus(h) : h ∈ HYPS

Whenever an hypothesis has been rejected or confirmed, it should not be suggested as a focus. The selection of hypotheses for the focus could for example be based on the frequency at which the hypotheses occur. This component should select one or more hypotheses whenever not all hypotheses have been rejected. This process is specified as a primitive component in the example.

### 4.3 Hypothesis Validation

The main process of Hypothesis Validation is to determine whether the hypotheses of a given focus set are valid. In addition to that it keeps track of hypotheses that have already been validated. The interface and internal atoms for Hypothesis Validation are:

*input atoms*    focus(h) : h ∈ HYPS  
*internal atoms*    observed(o) : o ∈ OBS  
*output atoms*    rejected(h), confirmed(h) : h ∈ HYPS

The input is obtained from Hypothesis Determination. The process Hypothesis Validation is composed of three primitive processes. Each of these are described shortly in the following paragraphs.

**Observation Determination** To validate a hypothesis, observations have to be performed. These observations are selected by the sub-process Observation Determination. The knowledge required for this selection might include cost of doing observations, reliability, and so on. The information required by this process are the hypotheses that are in focus and observations that have already been performed. The interface of this process is

*input atoms*    focus(h) , observed(o) : h ∈ HYPS, o ∈ OBS  
*output atoms*    to\_be\_observed(o) : o ∈ OBS

**Observation Execution** The Observations are made in the sub-process Observation Execution. The information this process requires are the observations it needs to perform. The output consists of the results of those observations. The interface of this component is as follows:

*input atoms*    target(o) : o ∈ OBS  
*output atoms*    o : o ∈ OBS

Note that these atoms  $o$  are at the object level, whereas the atoms such as  $\text{to\_be\_observed}(o)$  are at a meta-level (in which case  $o$  is a term, naming the atom  $o$ ; for convenience we use the same notation). By the information link  $\text{to\_be\_observed}$  from Observation Determination to Observation Execution, the meta-atom  $\text{to\_be\_observed}(o)$  is linked to the meta-atom  $\text{target}(o)$ .

**Hypothesis Evaluation** Given observation results, the process Hypothesis Evaluation derives conclusions about which hypotheses are true. This process has the same level as Observation Execution since it uses the observations made there to derive truth values of hypothesis in focus by means of anti-causal knowledge. The interface of this process is

*input atoms*     $o : o \in \text{OBS}$   
*output atoms*    $h : h \in \text{HYPS}$

#### 4.4 Information Links

In Fig. 3 the interaction within the whole system  $S$  is shown. The link hypotheses transfers the hypotheses determined in Hypothesis Determination to Hypothesis Validation. The link assessments transfers the results from the evaluation in Hypothesis Evaluation to Hypothesis Determination so this component knows which hypothesis are rejected. The last link, diagnosis transfers the diagnosis determined by the system to the output interface of the main component; this mediating link, relates the top level to the lower level of process abstraction. Within the component Hypothesis Validation, the mediating links  $\text{focus hyp}$  to HE and  $\text{eval info}$  play a comparable role: they relate the level of Hypothesis Validation to the lower level of process abstraction.

## 5 Verification of the System at the Top Level

First the manner in which time points are attached to the reasoning process is discussed.

### 5.1 Time Points

For the verification of this system we need to introduce time points to reason about the dynamics of this system as explained in Section 3. For the

component S time points are defined as:

- Time point 1 corresponds to the termination time of the first activation of component Hypothesis Validation
- Time point  $t + 1$  is after the subsequent activations of Hypothesis Determination and Hypothesis Validation have been finished.

For the component Hypothesis Validation the time points are defined as:

- Time point 0 corresponds to no activation of the component.
- Time point  $t + 1$  is after Hypothesis Evaluation has been active, or Hypothesis Validation terminates.

## 5.2 Properties for the Top Level of the System

First, it is determined which properties the system as a whole should satisfy. Considering that the system S is a diagnostic reasoning system, it is expected that S produces output of the form `confirmed(h)` and/or `rejected(h)` for some hypotheses h. A first requirement is that output generated by the system in terms of assessments of hypotheses is correct, i.e., if the system derives that a hypothesis has been confirmed, it is true in the world situation, and if the system derives it is rejected, it is false in the world situation. Let the current world state be denoted by M. The following property relates the output of the system to the world state.

**Assessment correctness of S** The system S is called *assessment correct* if:

$$\begin{aligned}
 & (\forall M \in \text{Traces}(S) \forall t \forall h \\
 & \text{state}_S(M, t, \text{output}(S)) \models \text{confirmed}(h) \Rightarrow M \models h ) \wedge \\
 & (\forall M \in \text{Traces}(S) \forall t \forall h \\
 & \text{state}_S(M, t, \text{output}(S)) \models \text{rejected}(h) \Rightarrow M \models \neg h )
 \end{aligned}$$

Next, the system is required to be effective in generating assessments, i.e. in qualifying every hypothesis either as confirmed or as rejected: during the process it should derive at least some positive assessment output (of the form `confirmed(h)`), except in case all hypotheses are false; then the system should derive that all hypotheses are rejected (assessment output of the form `rejected(h)`):

### Assessment effectiveness of S

The system S is called *assessment effective* if:

$$\begin{aligned} & (\exists h M \models h \Rightarrow \\ & \forall M \in \text{Traces}(S) \exists t \exists h' \text{state}_S(M, t, \text{output}(S)) \models \text{confirmed}(h') ) \wedge \\ & (\forall h M \models \neg h \Rightarrow \\ & \forall M \in \text{Traces}(S) \exists t \forall h \text{state}_S(M, t, \text{output}(S)) \models \text{rejected}(h) ) \end{aligned}$$

It is undesirable (for a static world situation) that the system changes its mind during the process. Therefore the requirement is chosen that once an assessment has been derived, this is never revised:

### Assessment conservativity of S

The system S is called *assessment conservative* if:

- a)  $\forall M \in \text{Traces}(S) \forall t \forall h$   
[  $\text{state}_S(M, t, \text{output}(S)) \models \text{confirmed}(h) \Rightarrow \text{state}_S(M, t+1, \text{output}(S)) \models \text{confirmed}(h)$  ]
- b)  $\forall M \in \text{Traces}(S) \forall t \forall h$   
[  $\text{state}_S(M, t, \text{output}(S)) \models \text{rejected}(h) \Rightarrow \text{state}_S(M, t+1, \text{output}(S)) \models \text{rejected}(h)$  ]

The following property defines when the component Hypothesis Determination is efficient in providing a focus. Here, and in the sequel the abbreviation *assessed(h)* is used for  $\text{confirmed}(h) \vee \text{rejected}(h)$ .

### Focus efficiency of S

The component hypothesis determination is called *focus efficient* if:

$$\begin{aligned} & \forall M \in \text{Traces}(HD) \forall t \forall h \\ & [ \text{state}_{HD}(M, t, \text{input}(HD)) \models \text{assessed}(h) \Rightarrow \\ & \text{state}_{HD}(M, t, \text{output}(HD)) \not\models \text{focus}(h) ] \end{aligned}$$

The following property expresses a monitor on the system's progression.

**System shows progression** If a trace  $M \in \text{Traces}(S)$  and a timepoint  $t$  are given, system S shows *progression* from time point  $t$  to time point  $t+1$  if

$$\begin{aligned} & \exists h [ \text{state}_S(M, t, \text{output}(S)) \not\models \text{assessed}(h) \wedge \\ & \text{state}_S(M, t+1, \text{output}(S)) \models \text{assessed}(h) ] \end{aligned}$$

**Transfer Successfulness** A system property which often is used is transfer successfulness: if a system component C transfers information to another component D, then component D will receive at its input what component C did output. In particular this property is needed for the three information links at the top level, namely hypotheses, diagnosis and assessments.

*Transfer successfulness from HD to HV:*

$$\begin{aligned} & \forall M \in \text{Traces}(S) \forall h, t \\ & \text{state}_S(M, t, \text{output}(\text{HD})) \models \text{focus}(h) \Rightarrow \\ & \text{state}_S(M, t+1, \text{input}(\text{HV})) \models \text{focus}(h) \end{aligned}$$

*Transfer successfulness from HV to S:*

$$\begin{aligned} & \forall M \in \text{Traces}(S) \forall h, t \\ & \text{state}_S(M, t, \text{output}(\text{HV})) \models \text{confirmed}(h) \Rightarrow \\ & \text{state}_S(M, t+1, \text{output}(S)) \models \text{confirmed}(h) \\ & \forall M \in \text{Traces}(S) \forall h, t \\ & \text{state}_S(M, t, \text{output}(\text{HV})) \models \text{rejected}(h) \Rightarrow \\ & \text{state}_S(M, t+1, \text{output}(S)) \models \text{rejected}(h) \end{aligned}$$

*Transfer successfulness from HV to HD:*

$$\begin{aligned} & \forall M \in \text{Traces}(S) \forall h, t \\ & \text{state}_S(M, t, \text{output}(\text{HV})) \models \text{confirmed}(h) \Rightarrow \\ & \text{state}_S(M, t+1, \text{input}(\text{HD})) \models \text{confirmed}(h) \\ & \forall M \in \text{Traces}(S) \forall h, t \\ & \text{state}_S(M, t, \text{output}(\text{HV})) \models \text{rejected}(h) \Rightarrow \\ & \text{state}_S(M, t+1, \text{input}(\text{HD})) \models \text{rejected}(h) \end{aligned}$$

**Transfer Groundedness** Transfer successfulness only guarantees that a component receives what is output by another component. Sometimes it is also needed the other way around: that it guaranteed that a component does not receive information that was not output by another component. The property transfer groundedness is formulated for the three information links at the toplevel in the following manner:

*Transfer groundedness from HD to HV:*

$$\begin{aligned} & \forall M \in \text{Traces}(S) \forall h, t \\ & \text{state}_S(M, t+1, \text{input}(\text{HV})) \models \text{focus}(h) \Rightarrow \\ & \text{state}_S(M, t, \text{output}(\text{HD})) \models \text{focus}(h) \end{aligned}$$

*Transfer groundedness from HV to S:*

$$\begin{aligned}
& \forall M \in \text{Traces}(S) \forall h, t \\
& \text{state}_S(M, t+1, \text{output}(S)) \models \text{confirmed}(h) \Rightarrow \\
& \text{state}_S(M, t, \text{output}(HV)) \models \text{confirmed}(h) \\
& \forall M \in \text{Traces}(S) \forall h, t \\
& \text{state}_S(M, t+1, \text{output}(S)) \models \text{rejected}(h) \Rightarrow \\
& \text{state}_S(M, t, \text{output}(HV)) \models \text{rejected}(h)
\end{aligned}$$

*Transfer groundedness from HV to HD:*

$$\begin{aligned}
& \forall M \in \text{Traces}(S) \forall h, t \\
& \text{state}_S(M, t+1, \text{input}(HD)) \models \text{confirmed}(h) \Rightarrow \\
& \text{state}_S(M, t, \text{output}(HV)) \models \text{confirmed}(h) \\
& \forall M \in \text{Traces}(S) \forall h, t \\
& \text{state}_S(M, t+1, \text{input}(HD)) \models \text{rejected}(h) \Rightarrow \\
& \text{state}_S(M, t, \text{output}(HV)) \models \text{rejected}(h)
\end{aligned}$$

The above transfer successfulness and groundedness properties are assumptions for the whole system; they are assumed and used in the proofs of a number of properties.

### 5.3 Assumptions Needed to Prove the Properties of the Top Level

The required properties of the system have been proven from assumed properties of the components at one level lower. During this proof process these assumptions have been discovered.

**5.3.1 Assumptions on Hypothesis Validation** Some assumptions are quite straightforward. For example, assessment correctness simply inherits upward from Hypothesis Validation:

#### **Assessment correctness of HV**

The component hypothesis\_validation is called *assessment correct* if:

$$\begin{aligned}
& \text{a) } (\forall M \in \text{Traces}(S) \forall t \forall h \\
& \quad \text{state}_S(M, t, \text{output}(HV)) \models \text{confirmed}(h) \Rightarrow M \models h )
\end{aligned}$$



Figure 5: Logical relations between properties at different levels for the diagnostic process model

$$\begin{aligned} \text{b) } & (\forall M \in \text{Traces}(S) \forall t \forall h \\ & \text{state}_S(M, t, \text{output}(HV)) \models \text{rejected}(h) \Rightarrow M \models \neg h ) \end{aligned}$$

Similarly, for assessment conservation:

### **Assessment conservativity of HV**

The component hypothesis\_validation is called *assessment conservative* if:

$$\begin{aligned} \text{a) } & \forall M \in \text{Traces}(S) \forall t \forall h \\ & [ \text{state}_S(M, t, \text{output}(HV)) \models \text{rejected}(h) \Rightarrow \\ & \text{state}_S(M, t+1, \text{output}(HV)) \models \text{rejected}(h) ] \\ \text{b) } & \forall M \in \text{Traces}(S) \forall t \forall h \\ & [ \text{state}_S(M, t, \text{output}(HV)) \models \text{confirmed}(h) \Rightarrow \\ & \text{state}_S(M, t+1, \text{output}(HV)) \models \text{confirmed}(h) ] \end{aligned}$$

For effectiveness, the relationship is not one-to-one as in the case of correctness and conservativity. However, also in this case, at least one (among others) of the required assumptions on Hypothesis Validation is that it is effective in generating assessments, as long as focus hypotheses are provided to it.

### **Assessment effectiveness of HV**

The component hypothesis\_validation is called *assessment effective* if:

$$\begin{aligned} & (\forall M \in \text{Traces}(HV) \forall t \\ & [ \exists h \text{state}_{HV}(M, t, \text{input}(HV)) \models \text{focus}(h) ] \Rightarrow \\ & [ \exists h' \text{state}_{HV}(M, t, \text{input}(HV)) \models \text{focus}(h') \wedge \\ & \text{state}_{HV}(M, t, \text{output}(HV)) \models \text{assessed}(h') ] ) \end{aligned}$$

The following property will be used in the proofs.

**Component HV shows progression** If a trace  $M \in \text{Traces}(S)$  and a time-point  $t$  are given, component HV shows *progression* from time point  $t$  to time point  $t'$  if

$$\begin{aligned} & \exists h [ \text{state}_S(M, t, \text{output}(HV)) \not\models \text{assessed}(h) \wedge \\ & \text{state}_S(M, t', \text{output}(HV)) \models \text{assessed}(h) ] \end{aligned}$$

### Transfer successfulness and groundedness properties within HV

Similar to the transfer successfulness and groundedness properties at the level of the whole system, also within the component HV properties are needed that guarantee proper information transfer between the different sub-components. These properties can be specified in a form as Section 5.2. For example, proper transfer of to be observed information generated by OD to OE requires:

*Transfer successfulness from OD to OE:*

$$\begin{aligned} & \forall M \in \text{Traces}(\text{HV}) \forall t, o \\ & \text{state}_{\text{HV}}(M, t, \text{output}(\text{OD})) \models \text{to\_be\_observed}(o) \\ & \Rightarrow \text{state}_{\text{HV}}(M, t+1, \text{input}(\text{OE})) \models \text{to\_be\_observed}(o) \end{aligned}$$

*Transfer groundedness from OD to OE:*

$$\begin{aligned} & \forall M \in \text{Traces}(\text{HV}) \forall t, o \\ & \text{state}_{\text{HV}}(M, t+1, \text{input}(\text{OE})) \models \text{to\_be\_observed}(o) \\ & \Rightarrow \text{state}_{\text{HV}}(M, t, \text{output}(\text{OD})) \models \text{to\_be\_observed}(o) \end{aligned}$$

Similar transfer properties are used for all other information transfer between components within HV (as depicted in Figure 3, lower part). For the proofs these properties are assumptions, on the basis of the design specification: the transfer properties are the expression of the semantics of the information links in a design specification.

**5.3.2 Assumptions on Hypothesis Determination** For the component Hypothesis Determination the assumption is made that it is efficient and effective in generating focus hypotheses. Focus efficiency means that no hypotheses are chosen in focus that already have been assessed.

#### Focus efficiency of HD

The component hypothesis\_determination is called *focus efficient* if:

$$\begin{aligned} & \forall M \in \text{Traces}(\text{HD}) \forall t \forall h \\ & [ \text{state}_{\text{HD}}(M, t, \text{input}(\text{HD})) \models \text{assessed}(h) \Rightarrow \\ & \text{state}_{\text{HD}}(M, t, \text{output}(\text{HD})) \not\models \text{focus}(h) ] \end{aligned}$$

Focus effectiveness means that as long as not all hypotheses have been assessed, there will be generated focus hypotheses.

### Focus effectiveness of HD

The component hypothesis\_determination is called *focus effective* if:

$$\begin{aligned} & \forall M \in \text{Traces}(\text{HD}) \forall t \\ & [\exists h \text{ state}_{\text{HD}}(M, t, \text{input}(\text{HD})) \not\models \text{assessed}(h) \wedge \\ & \forall h \text{ state}_S(M, t', \text{input}(\text{HD})) \not\models \text{confirmed}(h)] \\ & \Rightarrow \exists h' \text{ state}_{\text{HD}}(M, t, \text{output}(\text{HD})) \models \text{focus}(h') \end{aligned}$$

**5.3.3 Domain Assumptions** The properties at the top level also need assumptions on the (domain) ontology and knowledge to be used by the process model. These are the assumptions of the type considered in (Fensel, 1995; Fensel and Benjamins, 1996).

**Finite number of hypotheses** The number of hypotheses is finite.

**Static world** The world state is static during the processing of system S.

## 5.4 Sketch of proofs of some of the Properties of the Top Level

In this section only a short sketch is given. For more details, see the Appendix. In Fig. 5 the logical connections between the properties at different levels are depicted. At each step that the system shows progression, due to assessment conservativity, the set of assessed hypotheses becomes strictly larger. The proofs follow the pattern that the assumptions guarantee that as long as not all hypotheses have been assessed the system will show progression. If the number of hypotheses is finite, say N, then within at most N steps the set of assessed hypotheses will be the set of all hypotheses, and the system terminates.

It can be noted that for a static world the property of assessment correctness implies assessment conservatism, so in the graph of Fig. 5 more logical relationships can be drawn. However, to avoid a complicated graph we did not attempt to give a complete account of all possible logical relationships in Fig. 5.

## 6 Assumptions to Prove the Properties of HV

The properties of Hypothesis Validation needed to prove the properties of the top level of the system were discussed in Section 5.3 (see Fig. 5). The assumed properties of the sub-components of Hypothesis Validation, needed to prove these properties can also be found in Fig. 5, at the lower level. For shortness, these properties are only explained informally.

The required properties of *Observation Determination* are:

**Observation efficiency of OD:** No observations are generated that already were performed.

$$\begin{aligned} & \forall M \in \text{Traces}(\text{OD}) \forall t \forall o \\ & [ \text{state}_{\text{OD}}(M, t, \text{input}(\text{OD})) \models \text{observed}(o) \Rightarrow \\ & \text{state}_{\text{OD}}(M, t, \text{output}(\text{OD})) \not\models \text{to\_be\_observed}(o) ] \end{aligned}$$

**Observation effectiveness of OD:**

If there exists at least one hypothesis in focus, and not all observations have been performed, then at least one observation is generated.

$$\begin{aligned} & \forall M \in \text{Traces}(\text{OD}) \forall t \forall h \\ & [ \exists o \text{state}_{\text{OD}}(M, t, \text{input}(\text{OD})) \not\models \text{observed}(o) \\ & \wedge \text{state}_{\text{OD}}(M, t, \text{input}(\text{OD})) \models \text{focus}(h) \Rightarrow \\ & \exists o' \text{state}_{\text{OD}}(M, t, \text{output}(\text{OD})) \models \text{to\_be\_observed}(o') ] \end{aligned}$$

**Execution effectiveness of OD:**

Every generated observation is performed.

$$\begin{aligned} & \forall M \in \text{Traces}(\text{OD}) \forall t \forall o \\ & [ \text{state}_{\text{OD}}(M, t, \text{output}(\text{OD})) \models \text{to\_be\_observed}(o) \\ & \Rightarrow \exists t' > t \text{state}_{\text{OD}}(M, t', \text{input}(\text{OD})) \models \text{observed}(o) ] \end{aligned}$$

This actually is an environment property of OD.

The required properties of *Observation Execution* are:

**Observation conservativity of OE:**

Once an observation result has been obtained, it persists.

$$\begin{aligned} & \forall M \in \text{Traces}(\text{S}) \forall t \forall o \\ & [ \text{state}_{\text{S}}(M, t, \text{output}(\text{OE})) \models o \Rightarrow \text{state}_{\text{S}}(M, t+1, \text{output}(\text{OE})) \models o ] \wedge \\ & [ \text{state}_{\text{S}}(M, t, \text{output}(\text{OE})) \models \neg o \Rightarrow \text{state}_{\text{S}}(M, t+1, \text{output}(\text{OE})) \models \neg o ] \end{aligned}$$

**Observation correctness of OE:**

Every observation result that is obtained is true in the world situation.

$$\begin{aligned} & \forall M \in \text{Traces}(\text{OE}) \forall t \forall o \\ & [ \text{state}_{\text{OE}}(M, t, \text{output}(\text{OE})) \models o \Rightarrow M \models o ] \wedge \\ & [ \text{state}_{\text{OE}}(M, t, \text{output}(\text{OE})) \models \neg o \Rightarrow M \models \neg o ] \end{aligned}$$

The required properties of *Hypothesis Evaluation* are:

**Assessment conservativity of HE:**

Once an hypothesis assessment has been derived, it persists.

- a)  $\forall M \in \text{Traces}(\text{S}) \forall t \forall h$   
 $[ \text{state}_{\text{S}}(M, t, \text{output}(\text{HE})) \models h \Rightarrow \text{state}_{\text{S}}(M, t+1, \text{output}(\text{HE})) \models h ]$
- b)  $\forall M \in \text{Traces}(\text{S}) \forall t \forall h$   
 $[ \text{state}_{\text{S}}(M, t, \text{output}(\text{HE})) \models \neg h \Rightarrow \text{state}_{\text{S}}(M, t+1, \text{output}(\text{HE})) \models \neg h ]$

**Assessment decisiveness of HE:**

If for all possible observations, observation results have been input, then for every hypothesis an assessment can be derived.

$$\begin{aligned} & \forall M \in \text{Traces}(\text{HE}) \forall t \\ & [ \forall o [ \text{state}_{\text{HE}}(M, t, \text{input}(\text{HE})) \models o \vee \text{state}_{\text{HE}}(M, t, \text{input}(\text{HE})) \models \neg o ] \\ & \Rightarrow \\ & \forall h [ \text{state}_{\text{HE}}(M, t, \text{output}(\text{HE})) \models h \vee \text{state}_{\text{HE}}(M, t, \text{output}(\text{HE})) \models \neg h ] \end{aligned}$$

**Assessment correctness of HE:**

If a hypothesis is derived, then it is true in the world situation; if the negation of a hypothesis is derived, then the hypothesis is false in the world situation.

- a)  $(\forall M \in \text{Traces}(\text{S}) \forall t \forall h$   
 $\text{state}_{\text{S}}(M, t, \text{output}(\text{HE})) \models h \Rightarrow M \models h )$
- b)  $(\forall M \in \text{Traces}(\text{S}) \forall t \forall h$   
 $\text{state}_{\text{S}}(M, t, \text{output}(\text{HE})) \models \neg h \Rightarrow M \models \neg h )$

In addition to the *domain assumptions* mentioned in Section 5.3.3, the following are needed:

**Empirically foundedness:** The hypotheses can be uniquely characterised by means of observations; in other words: if two world situations satisfy exactly the same observations, then they also satisfy exactly the same hypotheses; see (Treur and Willems, 1994).

$$\forall o [ M_1 \models o \Leftrightarrow M_2 \models o ] \Rightarrow \forall h [ M_1 \models h \Leftrightarrow M_2 \models h ]$$

**Finite number of observations:** The number of observations needs to be finite because (in the worst case) the system should be able to do all (relevant) observations to assess all hypotheses.

## 7 Verification of Properties of Primitive Components

In Sections 5 and 6 verification of the generic model was described, based on assumed properties of the primitive components. If the model is to be used, instances are required for the primitive components (e.g., containing domain knowledge), and for these instances it has to be verified whether they satisfy the required properties. The instances of primitive components can be verified – for not too large knowledge bases - making use of the more standard methods described in (Treur and Willems, 1994). This will be illustrated for the example model for diagnosis.

To start, the component Hypothesis Determination should satisfy focus efficiency and focus effectiveness. Focus efficiency means that no hypotheses are chosen in focus that already have been assessed. In the temporal trace language this is expressed in the following form. For all traces, at all time points if at the input the information is available that some hypothesis  $h$  already was assessed, then it will not be at the output that it is in focus:

$$\begin{aligned} & \forall M \in \text{Traces}(\text{HD}) \forall t \forall h \\ & [ \text{state}_{HD}(M, t, \text{input}(\text{HD})) \models \text{assessed}(h) \Rightarrow \\ & \text{state}_{HD}(M, t, \text{output}(\text{HD})) \not\models \text{focus}(h) ] \end{aligned}$$

The domain description  $W(\text{HD})$  is defined by some constraints (related to the required properties) on the set of complete models  $\text{CMod}(I(\text{HD}))$ , where  $I(\text{HD})$  is the combination of input and output information types of the component Hypothesis Determination. The constraint C1 related to the above property is expressed as follows:

$$\bigwedge_h \neg ( \text{assessed}(h) \wedge \text{focus}(h) )$$

Here  $\bigwedge_h$  means taking the conjunction over all hypotheses  $h$ ; similarly  $\bigvee_h$  will denote taking the disjunction. The second property to be satisfied by Hypothesis Determination is focus effectiveness; this means that as long as not all hypotheses have been assessed, and no hypothesis has been confirmed, focus hypotheses will be generated. In the temporal trace language this is expressed as follows. For all traces and time points, if there exists at least one hypothesis for which no information is at the input that it was assessed, and for no hypothesis there is information on the input that it was confirmed, then there exists at least one hypothesis such that on the output there is information that it is in focus:

$$\begin{aligned} & \forall M \in \text{Traces}(\text{HD}) \forall t \\ & [ \exists h \text{ state}_{HD}(M, t, \text{input}(\text{HD})) \not\models \text{assessed}(h) \wedge \\ & \forall h \text{ state}_S(M, t, \text{input}(\text{HD})) \not\models \text{confirmed}(h) ] \\ & \Rightarrow \exists h' \text{ state}_{HD}(M, t, \text{output}(\text{HD})) \models \text{focus}(h') ] \end{aligned}$$

This property can be reformulated to the following defining constraint C2 for  $W(\text{HD})$ :

$$\bigwedge_h \text{assessed}(h) \vee \bigvee_h \text{confirmed}(h) \vee \bigvee_h \text{focus}(h)$$

In conclusion, a component can safely be chosen to play the role of Hypothesis Determination in the diagnostic model if it is sound and strongly complete with respect to the domain description  $W(\text{HD})$  defined by the two constraints above (i.e., the set of complete models satisfying the constraints), related to the temporal properties, i.e.,

$$W(\text{HD}) = \{ M \in \text{CMod}(\text{I}(\text{HD})) \mid M \models C1 \wedge C2 \}$$

In a similar manner the properties ‘observation effectiveness’ and ‘observation efficiency’ of the component Observation Determination reduce to constraints (on the set of complete models) defining domain description  $W(\text{OD})$ . Observation efficiency means that no observations are generated that already were performed:

$$\begin{aligned} & \forall M \in \text{Traces}(\text{OD}) \forall t \forall o \\ & [ \text{state}_{OD}(M, t, \text{input}(\text{OD})) \models \text{observed}(o) \Rightarrow \\ & \text{state}_{OD}(M, t, \text{output}(\text{OD})) \not\models \text{to\_be\_observed}(o) ] \end{aligned}$$

This is reformulated as the following defining constraint C3 for  $W(\text{OD})$ :

$$\bigwedge_o \neg ( \text{observed}(o) \wedge \text{to\_be\_observed}(o) )$$

Observation effectiveness means that if there exists at least one hypothesis in focus, and not all observations have been performed, then at least one

observation is generated.

$$\begin{aligned} & \forall M \in \text{Traces}(\text{OD}) \forall t \forall h \\ & [ \exists o \text{ state}_{\text{OD}}(M, t, \text{input}(\text{OD})) \not\models \text{observed}(o) \\ & \wedge \text{state}_{\text{OD}}(M, t, \text{input}(\text{OD})) \models \text{focus}(h) \Rightarrow \\ & \exists o' \text{ state}_{\text{OD}}(M, t, \text{output}(\text{OD})) \models \text{to\_be\_observed}(o') ] \end{aligned}$$

This is reformulated as the following defining constraint C4 for  $W(\text{OD})$ :  
 $\bigwedge_h \neg \text{focus}(h) \vee \bigwedge_o \text{observed}(o) \vee \bigvee_o \text{to\_be\_observed}(o)$

Again, a proper choice for an instantiation of component Observation Determination is made if it satisfies soundness and strong completeness with respect to domain description  $W(\text{OD})$  defined by the two constraints:

$$W(\text{HD}) = \{ M \in \text{CMod}(\text{I}(\text{OD})) \mid M \models \text{C3} \wedge \text{C4} \}$$

One of the required properties of Hypothesis Evaluation is assessment decisiveness, which means that if for all possible observations, observation results have been input, then for every hypothesis an assessment can be derived:

$$\begin{aligned} & \forall M \in \text{Traces}(\text{HE}) \forall t \\ & [ \forall o [ \text{state}_{\text{HE}}(M, t, \text{input}(\text{HE})) \models o \vee \text{state}_{\text{HE}}(M, t, \text{input}(\text{HE})) \models \neg o ] \Rightarrow \\ & \forall h [ \text{state}_{\text{HE}}(M, t, \text{output}(\text{HE})) \models h \vee \text{state}_{\text{HE}}(M, t, \text{output}(\text{HE})) \models \neg h ] \end{aligned}$$

This can be reformulated into the property ‘empirically foundedness’ of the domain description  $W$  describing the world situations to which the system is applied.

This shows in terms of the diagnostic example how within a compositional verification process, required (possibly dynamic) properties of candidate primitive components for a generic model can be formulated as static properties of their domain description; cf. (Treur and Willems, 1994; Lee-mans, Treur and Willems, 1993).

## 8 Discussion

Within this discussion a number of issues are discussed in subsequent subsections.

## 8.1 Compositionality in Design and in Verification

The design method DESIRE is based on compositionality. The compositional verification method described in this paper fits well to DESIRE, but can also be useful to any other compositional modelling approach. The advantage of a compositional approach to design is to be able to reuse components and process models easily; the compositional verification method extends this to the *reuse of proofs* for properties of components that are reused. For example, the compositional diagnostic reasoning model described in this paper could be modified by changing the component Hypothesis Validation. If this changed component has the same properties as the current, the proof of the top level properties can be reused to show that the new system has the same properties as the original. This has high value for a library of generic models (i.e., problem solving methods), where the domain knowledge is not yet known. The verification of generic models forces one to find the assumptions under which the generic task model is applicable for the considered domain, as is also discussed in (Fensel, 1995, Fensel and Benjamins, 1996). A *library* of reusable components and task models will be set up, consisting of both specifications of the components and models, and their *design rationale*. The properties of the components and their logical relations (e.g., as represented in Fig. 5) are to be part of the design rationale of a model. When the model is applied, only the properties at the leaves of such a tree have to be verified against the domain knowledge used. This is what the current paper contributes to principled development of knowledge-based systems based on knowledge level models.

Due to the compositional nature of the verification method, a *distributed approach* to verification is facilitated. This implies that several persons can work on the verification of the same model at the same time, once the properties to be verified have been determined. Since the proof of properties of a composed component depends on the properties of its sub-components, it is only necessary to know or to agree on the properties of these sub-components. The verification method proposed in this paper is useful for compositional knowledge-based systems as well as compositional multi-agent systems.

## 8.2 Expressivity of the Temporal Trace Language

One of the approaches based on a standard temporal logic can be found in (Fisher and Wooldridge, 1997). A main difference in comparison to this

work is that our approach exploits compositionality in an iterated manner. An advantage of their approach is that they can make use of a temporal belief logic. It would be a challenge to extend the approach as referred to a compositional variant of temporal belief logic. A first step in this direction can be found in (Engelfriet, Jonker and Treur, 1999).

The reason why, instead of a standard temporal logic language, the temporal trace language was used to formalise the dynamic properties is that it is much more expressive in a number of respects. In the first place, it has *order-sorted predicate logic* expressivity, whereas most standard temporal logics are propositional. Secondly, the explicit reference to *time points and time durations* offers the possibility of modelling the dynamics of real-time phenomena, such as sensory activity patterns in relation to mental properties (cf. [Port and van Gelder, 1995]). Third, in our approach states are *three-valued*; the standard temporal logics are based on two-valued states, which implies that for a given trace a form of closed world assumption is imposed. This means that, for example, in Concurrent MetateM (cf., [Fisher, 1994]), if the executable temporal logic specification leaves some atom unspecified, during construction of a trace the semantics will force it to be false. To avoid this, an atom has to be split into a positive and a negative variant. In our approach this is not needed.

Fourth, the possibility to quantify over traces allows for specification of *more complex behaviours*. As within most temporal logics, reactiveness and pro-activeness properties can be specified. In addition, in our language also properties expressing different types of adaptive behaviour can be expressed. For example a property such as ‘exercise improves skill’, which is a relative property in the sense that it involves the comparison of two alternatives for the history (e.g., one with many exercises, and one with almost no exercises). This type of property can be expressed in our language, whereas in standard forms of temporal logic different alternative histories cannot be compared. Fifth, in our language it is possible to define *local languages for parts* of a system. For example, the distinction between internal, external and interface languages is crucial, and is supported by the language, which also entails the possibility to quantify over system parts; this allows for specification of system modification over time, cf. (Dastani, Jonker, and Treur, 2001). Sixth, since state properties are used as first class citizens in the temporal trace language, it is possible to explicitly refer to them, and to quantify over them, enabling the specification of what are sometimes called *second-order properties*.

### 8.3 Diagnosis and its Dynamics

The example model used to illustrate the compositional verification method in this paper is a generic process model for the *dynamics* of diagnostic reasoning. In other literature diagnosis has been related to different types of logical formalisations, such as diagnosis from first principles (Reiter, 1987), hypothetical or abductive reasoning (Console and Torasso, 1990, 1991), default logic and other nonmonotonic logics. In this literature the main focus is on the outcome of a diagnostic task and how this outcome relates to the input. The approaches abstract as much as possible from the dynamics of the intermediate diagnostic process, which involves, among others, the decisions about selection of hypotheses to focus on, and selection of observations to be performed for a given particular intermediate (information) state within the diagnostic process. As we wanted to test the compositional verification method on the aspect of internal dynamics of a model, our choice has been to consider a diagnostic process model in the tradition of (Treur, 1993), in which dynamic aspects such as hypothesis selection and observation selection are modelled in an explicit manner. Since these dynamic aspects play an important role, formalisations that only relate output of a diagnostic process to input are not appropriate: an approach is needed in which the *temporal aspects of (the choices within) intermediate processes* can be expressed. The temporal trace language supports such an approach.

An alternative formalisation of a diagnostic process that has this emphasis on the dynamics of the intermediate process in common is described in (Hoek, Meyer, and Treur, 1994). A difference is that diagnosis addressed in that paper is based on *causal knowledge*, whereas the current paper addresses diagnosis based on *anti-causal knowledge*. A more important difference is that the language used in (Hoek et al., 1994) is a form of (branching time) temporal logic in which different histories cannot be compared. Although for most of the relevant properties of diagnostic processes this may not be a problem, there are properties (e.g., relative adaptive properties) which cannot be expressed in a branching time temporal logic, as also discussed in Section 8.2 above.

Nevertheless, the diagnostic example model used to test the compositional verification method in this paper does not display all possibilities the compositional verification method offers for dynamics. One of the properties of the example domain is that the world is static; this is not a requirement for the method. Apart from the work reported here, a model for co-operative

information agents has been verified (Jonker and Treur, 1998) and a multi-agent system with agents negotiating about load balancing of electricity use, where the world can be dynamic (Brazier, Cornelissen, Gustavsson, Jonker, Lindeberg, Polak, and Treur, 1998).

## 8.4 Knowledge Level Models and their Assumptions

Some well-known knowledge level modelling approaches are generic tasks (Chandrasekaran, 1986), DESIRE (Brazier et al., 1999), and CommonKADS (Schreiber et al., 2000). In the context of such knowledge level modelling approaches the explicit and formal specification and verification of dynamic (behavioural) properties was never included. Work in this area that comes closest to the current paper is described in (Fensel, 1995, Fensel and Benjamins, 1996; Fensel et al, 1996).

A main difference of the current paper in comparison to this work on knowledge level modelling is that in our approach compositionality of the verification is addressed; in the work as referred only *domain assumptions* are taken into account, and no hierarchical relations between properties are defined. Compared to (Fensel and Benjamins, 1996), where also properties of diagnosis are identified, in the current paper the properties are formalised in formal semantical terms (they are expressed in terms of temporal formal semantics), whereas in the paper as referred the properties are not (yet) formalised. For example, the formalisation of the assumption ‘heuristic search knowledge’ (see Table 3 in Section 4 in the paper as mentioned) in terms of the semantics of the behaviour of the system might turn out far from trivial. Especially the semantical formalisation of such dynamic properties and their logical relationships is a challenge. Furthermore, assumptions on the dynamics of hypothesis determination and the heuristic knowledge involved, as presented in our paper, have been left out of consideration. On the other hand, the value of the paper is that it gives an extensive account on various assumptions for model-based diagnosis; this was left out of consideration in our paper. Besides compositionality, a difference of our approach with (Harmelen and Teije, 1997) is that in the latter approach only *static properties* of diagnosis are considered, whereas in our approach also dynamic properties are covered, formalised in temporal semantics.

## 8.5 Other Comparisons

Previous work on verification of compositional knowledge-based systems, described in (Treur and Willems, 1995), was based on the formulation of a *compositional verification principle* described in (Abadi and Lamport, 1993), applied to knowledge-based systems. This principle lifts properties of the sub-components to the component in which these are embedded. If all sub-components satisfy a certain property, and they are connected in the right manner, then the component as a whole will satisfy that property. The properties that can be verified in this way are the properties such as '*functions properly*'. However, for most real-world systems it gives more insight to explicit 'proper functioning' in the form of (task and domain) specific properties, as has been shown above. In this sense the current paper is a further development and refinement of the work described in (Treur and Willems, 1995).

Also in the area of *Petri nets* verification and compositionality are investigated; for example (Rambags, 1994). A difference with our approach is that composition is *not iterated*. Another difference is that in contrast to our approach no language is provided in terms of which properties can be expressed. This means that only very strict notions such as observation equivalence can be addressed, whereas in our case we can express as well restricted aspects of a system, leaving the rest of its behaviour unspecified. This gives more flexibility of description and avoids having to prove too strong properties.

The temporal trace language used in this paper has some similarity with the approach in *situation calculus* (McCarthy and Hayes, 1969). In (Reiter, 1993) proving properties in situation calculus is addressed. A difference to this work is that explicit references are made to temporal traces and that we can incorporate arbitrary time durations in the processes.

## 8.6 Further Work

A continuation of this work covers the case of multi-agent models; for some first results, see (Jonker and Treur, 1998). To support the handwork of verification it would be useful to have tools to assist in the creation of the proofs. Moreover, in future work the development of supporting tools for verification will be considered. Support can be distinguished into two types of tools: one type for verification of properties of primitive components, and another type to relate properties of composed components to their subcomponents. At

the moment tools of the first type exist for the verification of primitive components with knowledge bases of limited complexity. No tools of the second type, i.e., for the verification of composed components exist yet. To obtain such tools, one approach is formalizing the proofs of a verification process using a first order logic in which time and states are represented explicitly, and an interactive theorem prover for first order logic to support the proofs. An option is whether the tool KIV (based on dynamic logic) can be used; see (Reif, 1995). Some first, positive experiences with KIV for verification of an example model of a knowledge-based system are reported in (Fensel et al., 1996).

Within the area of automated reasoning and theorem proving the application of theorem provers and checkers to verification has been addressed for a number of systems. For example *HOL* (Alves-Foss and Levitt, 1991; Gordon and Melham, 1993; Melham, 1993), *Nqthm* (Bevier, Hunt Jr., Moore, and Young, 1989; Angelo, Verkest, Claesen, and De Man, 1993; Boyer, Kaufmann, and Moore, 1995), or *Coq* (Huet, Kahn, and Paulin-Mohring, 1997). Other relevant approaches in this area of automated reasoning and verification can be found in (Abadi and Lamport, 1993; Bundy, Giunchiglia, Villafiorita, and Walsh, 1997; Manna and Pnueli, 1995; Yoeli, 1990). It is one of the challenges for future work to find out if and how for the specific case of knowledge-based systems and the specific notion of compositionality that is used in our approach, one of these existing provers or checkers can be used as a basis for a dedicated supporting software environment. In our view, due to the conceptual and computational complexity of the applications aimed for, a verification process needs an essential contribution from a human verifier. Therefore the aim is to develop an interactive environment where the human user is responsible for the identification of the composition structure leading to different abstraction levels, for the properties at each of the abstraction levels, and perhaps also (per component) proof skeletons, that can be detailed and checked by the software. The identification of the composition structure and of the properties per abstraction level can be integrated with the requirements engineering and design process, as has been described in (Herlea, Jonker, Treur, and Wijngaards, 1999).

## 9 Acknowledgements

Dieter Fensel provided useful comments on an earlier version of this paper.

## 10 References

Abadi, M. and L. Lamport (1993). Composing Specifications, *ACM Transactions on Programming Languages and Systems*, Vol. 15, No. 1, 1993, pp. 73-132.

Alves-Foss, J., and Levitt, K., Verification of Secure Distributed Systems in Higher Order Logic: A Modular Approach Using Generic Components in *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy; Oakland, California*, 1991, pp. 122-35.

Angelo, C. M., Verkest, D., Claesen, L., and De Man, H., On the Comparison of HOL and Boyer-Moore for Formal Hardware Verification, *Formal Methods in System Design*, vol. 2, 1993, pp. 45-72.

Benjamins, R., Fensel, D., Straatman, R. (1996). Assumptions of problem-solving methods and their role in knowledge engineering. In: W. Wahlster (Ed.), *Proceedings of the Twelfth European Conference on Artificial Intelligence, ECAI'96*, John Wiley and Sons, 1996, pp. 408-412.

Beusekom, F. van, Brazier, F.M.T., Schipper, P., and Treur, J., Development of an ecological decision support system. In: A.P. del Pobil, J. Mira, and M. Ali (eds.), *Tasks and Methods in Applied Artificial Intelligence (Proceedings of the 11th International Conference on Industrial and Engineering Applications of AI and Expert Systems, IEA/AIE'98, vol. II)*. Lecture Notes in AI, vol. 1416, Springer Verlag, 1998, pp. 815-825

Bevier, W.R., Hunt Jr., W., A., Moore, J.S., Young, W.D., An Approach to Systems Verification. *Journal of Automated Reasoning* 5, 1989, pp. 411-428

Boyer, R.S., M. Kaufmann and J S. Moore, The Boyer-Moore Theorem Prover and Its Interactive Enhancement (Nqthm), *Computers and Mathematics with Applications*, Vol. 29, No. 2, pp. 27-62, 1995

Brazier, F.M.T., Dunin-Keplicz, B., Jennings, N.R. and Treur, J. (1995). Formal Specification of Multi-Agent Systems: a Real-World Case. In: V. Lesser (Ed.), *Proceedings of the First International Conference on Multi-Agent Systems, ICMAS'95*, MIT Press, Cambridge, MA, pp. 25-32. Extended version in: *International Journal of Cooperative Information Systems*, M. Huhns, M. Singh, (Eds.), special issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems, vol. 6, 1997, pp. 67-94.

Brazier, F.M.T., Jonker, C.M., Jungen, F.J., and Treur, J., Distributed Scheduling to Support a Call Centre: a Co-operative Multi-Agent Approach. *Applied Artificial Intelligence Journal*, vol. 13, 1999, pp. 65-90. H. S. Nwana

and D. T. Ndumu (eds.), Special Issue on Multi-Agent Systems.

Brazier, F.M.T., Jonker, C.M., Treur, J., and Wijngaards, N.J.E. (2000). On the Use of Shared Task Models in Knowledge Acquisition, Strategic User Interaction and Clarification Agents. *International Journal of Human-Computer Studies*, vol. 52, 2000, pp. 77-110.

Brazier, F.M.T., Treur, J., Wijngaards, N.J.E. and Willems, M. (1999). Temporal Semantics of Compositional Task Models and Problem Solving Methods. *Data and Knowledge Engineering*, vol. 29 (1), 1999, pp. 17-42. Preliminary version in: B.R. Gaines, M.A. Musen (Eds.), Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-based Systems workshop, KAW'96, Calgary: SRDG Publications, Department of Computer Science, University of Calgary, 1996, pp. 15/1-15/17.

Brazier F.M.T., Treur J., Wijngaards N.J.E. Modelling interaction with experts: the role of a shared task model. In: Wahlster, W. (ed.), *Proc. of the 12th European Conference on AI, ECAI'96*, Wiley and Sons, Chichester, pp. 241-245.

Bundy, A., Giunchiglia, F., Villafiorita, A., Walsh, T., Abstract Proof Checking: An Example Motivated by an Incompleteness Theorem. *Journal of Automated Reasoning*, vol. 19, 1997, pp. 319-346

Chandrasekaran, B. (1986). Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design. *IEEE Expert*, 1986, Vol. 1, pp. 23-30.

Console, L. and Torasso, P. (1990). Hypothetical reasoning in causal models. *Int. J. of Intelligent Systems*, vol. 5, pp. 83-124.

Console, L. and Torasso, P. (1991). A spectrum of logical definitions of model-based diagnosis. *Computational Intelligence*, vol. 7, pp. 133-141

Dastani, M., Jonker, C.M., and Treur, J. (2001). A Requirement Specification Language for Configuration Dynamics of Multi-Agent Systems. In: Wooldridge, M., Ciancarini, P., and Weiss, G. (eds.), *Proc. of the 2nd International Workshop on Agent-Oriented Software Engineering, AOSE'01*. Lecture Notes in Computer Science, Springer Verlag, to appear.

Engelfriet, J., Jonker, C.M. and Treur, J., Compositional Verification of Multi-Agent Systems in Temporal Multi-Epistemic Logic. In: J.P. Mueller, M.P. Singh, A.S. Rao (eds.), *Intelligent Agents V, Proc. of the Fifth International Workshop on Agent Theories, Architectures and Languages, ATAL'98*. Lecture Notes in AI, vol. 1555, Springer Verlag, 1999, pp. 177-194. Extended version to appear in: *Journal of Logic, Language and Information*.

Fensel, D. (1995). Assumptions and limitatons of a problem solving

method: a case study. In: B.R. Gaines, M.A. Musen (Eds.), Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-based Systems workshop, KAW'95, Calgary: SRDG Publications, Department of Computer Science, University of Calgary, 1995.

Fensel, D., Benjamins, R. (1996) Assumptions in model-based diagnosis. In: B.R. Gaines, M.A. Musen (Eds.), Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-based Systems workshop, KAW'96, Calgary: SRDG Publications, Department of Computer Science, University of Calgary, 1996, pp. 5/1-5/18.

Fensel, D., Schonegge, A., Groenboom, R., Wielinga, B. (1996). Specification and verification of knowledge-based systems. In: B.R. Gaines, M.A. Musen (Eds.), Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-based Systems workshop, KAW'96, Calgary: SRDG Publications, Department of Computer Science, University of Calgary, 1996, pp. 4/1-4/20.

Fisher, M., A survey of Concurrent METATEM — the language and its applications. In: D.M. Gabbay, H.J. Ohlbach (eds.), Temporal Logic — Proceedings of the First International Conference, Lecture Notes in AI, vol. 827, pp. 480–505.

Fisher, M., and Wooldridge, M., On the Formal Specification and Verification of Multi-Agent Systems. In: *International Journal of Cooperative Information Systems*, M. Huhns, M. Singh, (eds.), special issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems, vol. 6, 1997, pp. 67-94.

Gordon, M. J. C. and T. F. Melham (eds.), *Introduction to HOL: A theorem proving environment for higher order logic* (Cambridge University Press, 1993).

Harmelen, F. van, Teije, A. ten (1997). Validation and verification of diagnostic systems based on their conceptual model. In: *Proceedings of the Fourth European Symposium on the Validation and Verification of Knowledge-based Systems, EUROVAV'97*, 1997.

Herlea, D.E., Jonker, C.M., Treur, J., and Wijngaards, N.J.E., Integration of Behavioural Requirements Specification within a Knowledge Engineering Methodology. In: D. Fensel, R. Studer (eds.), *Knowledge Acquisition, Modelling and Management (Proceedings of the 11th European Workshop on Knowledge Acquisition, Modelling and Management, EKAW'99)*. Lecture Notes in AI, vol. 1621, Springer Verlag, 1999, pp. 173-190.

Hoek, W. van der, Meyer, J.-J.Ch., and Treur, J., (1994). Formal seman-

tics of temporal epistemic reflection. In: L. Fribourg and F. Turini (ed.), *Logic Program Synthesis and Transformation-Meta-Programming in Logic*, Proc. Fourth Int. Workshop on Meta-programming in Logic, META'94, Lecture Notes in Computer Science, vol. 883, Springer Verlag, pp. 332-352.

Huet, G. Kahn, G. Paulin-Mohring, C., *The Coq Proof Assistant - A tutorial*, Version 6.1, *rapport technique N°204*, INRIA, Août 1997.

Jonker, C.M. and Treur, J. (1998). *Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness*. In: W.P. de Roever, H. Langmaack, A. Pnueli (eds.), *Proceedings of the International Workshop on Compositionality, COMPOS'97*. Lecture Notes in Computer Science, vol. 1536, Springer Verlag, 1998, pp. 350-380.

Jonker, C.M., and Treur, J., *A Generic Process Control Model and its Application to the Control of Biochemical Processes*. In: I. Imam, Y. Kodratoff, A. El-Dessouki, and M. Ali (eds.), *Multiple Approaches to Intelligent Systems (Proc. of the 12th International Conference on Industrial and Engineering Applications of AI and Expert Systems, IEA/AIE'99)*. Lecture Notes in AI, vol. 1611, Springer Verlag, 1999, pp. 296-305

Jonker, C.M., Treur, J., *An Agent Architecture for Multi-Attribute Negotiation*. In: B. Nebel (ed.), *Proc. of the 17th International Joint Conference on AI, IJCAI'01*. To appear, 2001.

Jonker, C. M., and, Vollebregt, A. M., *ICEBERG: Exploiting Context in Information Brokering Agents*. In: M. Klusch, L. Kerschberg (eds.), *Cooperative Information Agents IV, Proceedings of the Fourth International Workshop on Cooperative Information Agents, CIA 2000*. Lecture Notes in Artificial Intelligence (LNAI 1860), Springer Verlag. pp. 27-38, 2000.

Leemans, P., J. Treur, and M. Willems (1993). *On the verification of knowledge-based reasoning modules*, Report IR-346, Department of Mathematics & Computer Science, Artificial Intelligence Group, Vrije Universiteit Amsterdam, 1993.

Melham, T., *Higher Order Logic and Hardware Verification*, Cambridge Tracts in Theoretical Computer Science 31 (Cambridge University Press, 1993).

Manna, Z., and Pnueli, A., *Temporal Verification of Reactive Systems: Safety*. Springer Verlag, 1995.

McCarthy, J., and Hayes, P.J., *Some Philosophical Problems from the Standpoint of Artificial Intelligence*, *Machine Intelligence*, vol. 4, 1969, pp. 463-502.

Port, R.F., Gelder, T. van (eds.) (1995). *Mind as Motion: Explorations*

in the Dynamics of Cognition. MIT Press, Cambridge, Mass.

Rambags, P.M.P., Decomposition and Protocols in High-Level Petri Nets. Ph.D. Thesis. Eindhoven University, 1994.

Reif, W. (1995). The KIV Approach to Software Engineering. In: M. Broy, S. Jänichen (eds.), *Methods, Languages, and Tools for the Construction of Correct Software*, Lecture Notes in Computer Science, vol. 1009, Springer Verlag, 1995.

Reiter, R. (1987). A Theory of Diagnosis from First Principles. *Artificial Intelligence*, vol. 32, pp. 57-95.

Reiter, R., (1993). Proving Properties of States in the Situation Calculus, *Artificial Intelligence*, vol. 64, 1993, pp. 337-351.

Schreiber, A. Th., Akkermans, J. M., Anjewierden, A. A., Hoog, R. de, Shadbolt, N. R. Velde, W. van de and Wielinga, B. J. (2000). *Knowledge Engineering and Management*. MIT Press.

Treur, J. (1993). Heuristic reasoning and relative incompleteness. *International Journal of Approximate Reasoning*, vol. 8, 1993, pp. 51-87.

Treur, J., and M. Willems (1994). A logical foundation for verification. In: *Proceedings of the Eleventh European Conference on Artificial Intelligence, ECAI'94*, A.G. Cohn (Ed.), John Wiley & Sons, Ltd., 1994, pp. 745-749.

Treur, J., and M. Willems (1995). Formal notions for verification of dynamics of knowledge-based systems. In: *Proceedings of the Third European Symposium on the Validation and Verification of Knowledge-based Systems, EUROVAV'95*, 1995, pp. 189-199.

Yoeli, M., Formal verification of hardware design. IEEE Computer Society Press, 1990.

# A On the proofs of the properties of the top level

In this Appendix as an illustration, some of the details are given of the proofs of the properties at the top level. Some of the details of the proofs of the following four properties are shown (see Figure 5) or sketched:

- system assessment correct  
(from: hypothesis validation assessment correct)
- system assessment conservative  
(from: hypothesis validation assessment conservative)
- system assessment effective  
(from: system assessment conservative, hypothesis determination focus efficient, hypothesis determination focus effective, and hypothesis validation assessment effective)
- system terminates  
(from: system assessment effective)

## A.1 The property system assessment correct

Assumption for the proof is hypothesis validation assessment correct:

$$\begin{aligned}
 & (\forall M \in \text{Traces}(S) \forall t \forall h \\
 & \text{state}_S(M, t, \text{output}(HV)) \models \text{confirmed}(h) \Rightarrow M \models h ) \\
 & \wedge \\
 & (\forall M \in \text{Traces}(S) \forall t \forall h \\
 & \text{state}_S(M, t, \text{output}(HV)) \models \text{rejected}(h) \Rightarrow M \models \neg h )
 \end{aligned}$$

The proof runs as follows. The system S is correct if

$$\begin{aligned}
 & (\forall M \in \text{Traces}(S) \forall t \forall h \\
 & \text{state}_S(M, t, \text{output}(S)) \models \text{confirmed}(h) \Rightarrow M \models h ) \wedge \\
 & (\forall M \in \text{Traces}(S) \forall t \forall h \\
 & \text{state}_S(M, t, \text{output}(S)) \models \text{rejected}(h) \Rightarrow M \models \neg h )
 \end{aligned}$$

The only information link to the output interface of S is diagnosis. The transfer groundedness property of this link expresses:

$$\forall M \in \text{Traces}(S) \forall t \forall h$$

$$\begin{aligned}
& \text{state}_S(M, t, \text{output}(S)) \models \text{confirmed}(h) \Rightarrow \\
& \text{state}_S(M, t+1, \text{output}(HV)) \models \text{confirmed}(h) \wedge \\
& \text{state}_S(M, t, \text{output}(S)) \models \text{rejected}(h) \Rightarrow \\
& \text{state}_S(M, t+1, \text{output}(HV)) \models \text{rejected}(h)
\end{aligned}$$

Suppose  $M, t, h$  are given with

$$\text{state}_S(M, t, \text{output}(S)) \models \text{confirmed}(h)$$

Then by the transfer groundedness property it holds

$$\text{state}_S(M, t+1, \text{output}(HV)) \models \text{confirmed}(h)$$

From assessment correctness of hypothesis validation it follows  $M \models h$ .

In a similar manner it is proved that

$$\text{state}_S(M, t, \text{output}(S)) \models \text{confirmed}(h)$$

implies  $M \models \neg h$ . This proves system correctness.

## A.2 The property system assessment conservative

The system  $S$  is called assessment conservative if:

- a)  $\forall M \in \text{Traces}(S) \forall t \forall h$   
 $[ \text{state}_S(M, t, \text{output}(S)) \models \text{confirmed}(h) \Rightarrow$   
 $\text{state}_S(M, t+1, \text{output}(S)) \models \text{confirmed}(h) ]$
- b)  $\forall M \in \text{Traces}(S) \forall t \forall h$   
 $[ \text{state}_S(M, t, \text{output}(S)) \models \text{rejected}(h) \Rightarrow$   
 $\text{state}_S(M, t+1, \text{output}(S)) \models \text{rejected}(h) ]$

Assumption for the proof: hypothesis validation assessment conservative.

The component hypothesis validation is called assessment conservative if:

- a)  $\forall M \in \text{Traces}(S) \forall t \forall h$   
 $[ \text{state}_S(M, t, \text{output}(HV)) \models \text{rejected}(h) \Rightarrow$   
 $\text{state}_S(M, t+1, \text{output}(HV)) \models \text{rejected}(h) ]$

$$\begin{aligned}
& \text{b) } \forall M \in \text{Traces}(S) \forall t \forall h \\
& \quad [ \text{state}_S(M, t, \text{output}(HV)) \models \text{confirmed}(h) \Rightarrow \\
& \quad \text{state}_S(M, t+1, \text{output}(HV)) \models \text{confirmed}(h) ]
\end{aligned}$$

Suppose  $M, t > 0, h$  are given with  
 $\text{state}_S(M, t, \text{output}(S)) \models \text{confirmed}(h)$

By transfer groundedness it follows  
 $\text{state}_S(M, t-1, \text{output}(HV)) \models \text{confirmed}(h)$

From this by assessment conservatism of HV it follows  
 $\text{state}_S(M, t, \text{output}(HV)) \models \text{confirmed}(h)$

By transfer successfulness it follows  
 $\text{state}_S(M, t+1, \text{output}(S)) \models \text{confirmed}(h)$

The same proof can be followed with  $\text{rejected}(h)$  instead of  $\text{confirmed}(h)$ . This proves conservatism of the system.

### A.3 The property system assessment effective

Assumptions for the proof of the system's assessment effectiveness are: system assessment conservative, hypothesis determination focus efficient, hypothesis determination focus effective, and hypothesis validation assessment effective. A crucial step in the proof is the following intermediate proposition. It is shown how is proven first. In a trace  $M \in \text{Traces}(S)$  and a time point  $t$ , component HV shows progression from time point  $t$  to time point  $t+2$  if

$$\begin{aligned}
& \exists h [ \text{state}_S(M, t, \text{output}(HV)) \not\models \text{assessed}(h) \wedge \\
& \quad \text{state}_S(M, t+2, \text{output}(HV)) \models \text{assessed}(h) ]
\end{aligned}$$

**Proposition** The component HV will show progression as long as no hypothesis has been confirmed and there is still a hypothesis that was not rejected, i.e., for all traces  $M \in \text{Traces}(S)$  and time points  $t$  it holds:

$$\begin{aligned}
& [ \neg [ \exists h \text{state}_S(M, t, \text{output}(HV)) \models \text{confirmed}(h) ] \wedge \\
& \quad \neg [ \forall h \text{state}_S(M, t, \text{output}(HV)) \models \text{rejected}(h) ] ] \\
& \Rightarrow \exists h [ \text{state}_S(M, t, \text{output}(HV)) \not\models \text{assessed}(h) \wedge \\
& \quad \text{state}_S(M, t+2, \text{output}(HV)) \models \text{assessed}(h) ]
\end{aligned}$$

**Proof.** The proof runs as follows. Assume the conditions

$$\neg [ \exists h \text{ state}_S(M, t, \text{output(HV)}) \models \text{confirmed}(h) ] \wedge \\ \neg [ \forall h \text{ state}_S(M, t, \text{output(HV)}) \models \text{rejected}(h) ]$$

are satisfied. By transfer groundedness of the interaction from HV to HD it holds:

$$\neg [ \exists h \text{ state}_S(M, t+1, \text{input(HD)}) \models \text{confirmed}(h) ] \wedge \\ \neg [ \forall h \text{ state}_S(M, t+1, \text{input(HD)}) \models \text{rejected}(h) ]$$

From this it follows that:

$$\exists h \text{ state}_S(M, t', \text{input(HD)}) \not\models \text{assessed}(h) \wedge \\ \forall h \text{ state}_S(M, t', \text{input(HD)}) \not\models \text{confirmed}(h)$$

By focus effectiveness of HD it follows that

$$\exists h \text{ state}_S(M, t+1, \text{output(HD)}) \models \text{focus}(h)$$

By transfer successfulness from HD to HV it follows:

$$\exists h \text{ state}_S(M, t+2, \text{input(HV)}) \models \text{focus}(h)$$

Using assessment effectiveness for HV it follows that there exists an  $h'$  such that:

$$\text{state}_{HV}(M, t+2, \text{input(HV)}) \models \text{focus}(h') \wedge \\ \text{state}_{HV}(M, t', \text{output(HV)}) \models \text{assessed}(h')$$

From transfer groundedness it follows that

$$\text{state}_{HV}(M, t+1, \text{output(HD)}) \models \text{focus}(h')$$

and from focus efficiency of HD we have

$$\text{state}_{HV}(M, t+1, \text{input(HD)}) \not\models \text{assessed}(h')$$

By transfer groundedness it follows

$$\text{state}_{HV}(M, t, \text{output(HV)}) \not\models \text{assessed}(h')$$

This shows that the hypothesis  $h'$  was not assessed at time  $t$  but is assessed at time  $t+2$ . This proves the Proposition.

The proof of the property system assessment effective depends on this proposition. As soon as there is no progression anymore (which time point will eventually occur due to finiteness of the set of hypotheses and by conservativity), one of the two conditions

$$\neg [ \exists h \text{ state}_S(M, t, \text{output(HV)}) \models \text{confirmed}(h) ] \wedge \\ \neg [ \forall h \text{ state}_S(M, t, \text{output(HV)}) \models \text{rejected}(h) ]$$

in the Proposition has to be false. If the first one is false, then:

$$\exists h \text{ state}_S(M, t, \text{output}(\text{HV})) \models \text{confirmed}(h)$$

If the second one is false, then:

$$\forall h \text{ state}_S(M, t, \text{output}(\text{HV})) \models \text{rejected}(h)$$

From correctness, in the first case it follows that

$$\exists h M \models h,$$

and in the second case:

$$\forall h M \models \neg h.$$

This finishes the overview of the proof of assessment effectiveness of S.

#### **A.4 The property system terminates**

Finally, the property termination of S is proven on the basis of the proposition above, the assumptions assessment conservativity of HV, finite number of hypotheses, and static world. From the proposition and conservativity (if an hypothesis is rejected it will always stay rejected) it follows that for each time point t, if S does not terminate at time point t+1, the number of rejected hypotheses has been increased by at least one. Combining this result with domain assumption finite number of hypotheses it follows that system S will terminate after a finite number of time points.