Appears in *IEEE Transactions on Knowledge and Data Engineering*, Vol. 10, No. 5, pp. 706-726, 1988.

Goal-Directed Reasoning with ACE-SSM

Michel Benaroch
School of Management, Syracuse University,
Syracuse, NY 13244
Email: mbenaroc@syr.edu
URL: http://sominfo.syr.edu/facstaff/mbenaroc/benaroch.html
(315) 443-3492

ABSTRACT

Past research shows that the goal of knowledge-based systems (KBSs) is not only to produce a solution to a problem situation they face, but also to construct, implicitly or explicitly, a *situation-specific model* (SSM) that explicates the rationale behind that solution. This paper focuses on how KBSs can benefit from the availability of explicit *goal knowledge* that reflects the underlying structure (ontology) of SSMs constructed for an application task. It first shows how goal knowledge can be captured. Then, it explains how ACE-SSM -- an architecture for constructing explicit SSMs -- uses this knowledge to direct the construction of explicit SSMs. Finally, it discusses benefits that KBSs can derive from the availability of explicit SSMs and their underlying goal knowledge. Some of these benefits pertain to ways to simplify the construction and maintenance of KBSs through reuse, while others have to do with ways to endow KBSs with more robust problem-solving and explanation capabilities. These benefits are illustrated using concrete examples.

Index Terms: goal-directed reasoning, goal knowledge, knowledge-based systems, KBS architecture, situation-specific models.

1 INTRODUCTION

Research on knowledge-based systems (KBS) has focused on capturing knowledge explicitly in ways that enable these systems to produce explanations, avoid brittleness, etc. Three types of knowledge can be respectively distinguished: *goal knowledge*, ***G***, reflecting the goal of an application task, or what the task entails producing as solutions; *strategic knowledge*, ***S***, mirroring the control strategy applied to the task, or how the task is solved; and, *domain knowledge*, ***K***, that is the data used in the problem-solving process. Of these, goal knowledge has received the least attention.

Early research focused on making explicit *domain knowledge*, *K*. It produced systems like MYCIN, which captured *K* using production rules, separate from the rule interpreter [10]. In these systems, however, some strategic knowledge, *S*, for controlling the order of processing rules was often compiled into the physical order of rule clauses and of rules in *K*. Consequently, *K* and *S* were neither accessible nor interpretable for purposes of explanation, reuse, etc.

Later research focused on separating *strategic knowledge*, *S*, from *K*. It yielded systems like NEOMYCIN, which typically modeled an application task as a hierarchy of procedural subtasks, where high level subtasks capture *S* and use it to sequence lower level subtasks, all the way to primitive subtasks that directly use *K* [5]. Capturing *S* in this fashion helped to make certain control decisions explicit, and thus

enabled KBSs to explain some aspects of their behavior. More importantly, many subtasks capturing *S* became viewed as task-independent reusable modules, because they were found to be common to various applications. Chandrasekaran [4], who termed such modules *generic tasks* (*GTs*), showed that a complete inference procedure for an application can be functionally configured as a hierarchy of GTs that were specialized for the application. McDermott [18] relatedly found that some hierarchies of GTs form task-specific inference procedures he termed *problem-solving methods* (*PSMs*). He showed how these PSMs can be specialized into complete working KBS applications because they clearly identify the specific parts of *K* that they need during problem solving.

Despite progress made on *K* and *S*, *goal knowledge*, *G*, remains implicit in all known KBSs. This is surprising in light of Clancey's [6] observation — what all KBSs do to solve problem situations they face is construct (usually implicit) *situation-specific models* (*SSMs*) that explicate the rational underlying solutions they produce (e.g., in diagnosis an SSM could be a causal argument having the structure of a proof). This means that, although the behavior of many known KBSs is characterized as being goal-driven, these systems are not really aware of their goals. This observation applies even to systems like ABLE and NEOMYCIN, in which somewhat explicit SSMs emerge from interactions of the subtasks they use to capture *S* [5].

This paper respectively focuses on two issues. One is how to make goal knowledge, *G*, explicit as well. The paper specifically explains how *G* is represented and used with ACE-SSM — an architecture for constructing explicit SSMs. The second issue is how availability of *G* and the explicit SSMs it helps to construct can benefit KBSs in new ways that go beyond the benefits from having only *K* and *S* explicit. Informally put, *G* is knowledge that reflects the underlying structure (ontology) of SSMs that a system constructs to meet its goals. Respectively, we show how *G* helps to anchor ACE-SSM's reasoning in the desire to make these SSMs explicit. Explicit SSM are valuable in two senses, which will be illustrated later. They are each a complete record of what a system did to produce final and intermediate inference results comprising the solution to a specific problem situation. Their availability hence allows explaining solutions and the problem-solving behaviors that produced them, in ways of which typical KBSs are not yet capable. Additionally, explicit SSMs facilitate grounding all control decisions in the need to evolve these SSMs into a state with characteristics specified by *G*. This permits expressing *S* declaratively, in ontological terms pertaining to *G*, instead of embedding it in procedural subtasks or production rules. This in itself could allow a KBS to dynamically reflect on its problem-solving behavior (recorded in SSMs it constructs) and adjust its declarative *S* to attune this behavior. More importantly, availability of both *G* and *S* in a declarative form helps to endow a system with more robust reasoning capabilities. When the *G*'s reflecting the goals of different KBSs are accessible, they can be analyzed to determine whether they are complementary and/or compatible in some sense. If they are, these systems' domain knowledge (*K*) and reasoning (governed by *S*) can be integrated in ways that avoid most of the usual difficulties encountered in this context. Finally, we will also show that, since *G* is reusable across application tasks that construct SSMs similar in topology, *S* also becomes reusable because its declarative representation centers around the *G* with which it is used. In turn, this helps to simplify the construction of KBS applications involving the same task ontology.

The rest of this paper is organized as follows. Section 2 discusses the notion of goal-directed reasoning and how it applies in ACE-SSM compared to other approaches. Section 3 explains formally how goal knowledge, *G*, is represented and used in ACE-SSM for the purpose of constructing explicit SSMs. Section 4 shows why ACE-SSM is both domain- and task-independent as well as assesses its computational requirements. Section 5 elaborates on, and illustrates, the ways by which KBSs can benefit from the availability of *G* and explicit SSMs. Section 6 offers some concluding remarks and specific directions for future research.

2 GOAL KNOWLEDGE IN CONTEXT

Having said that goal knowledge, *G*, is related to the notion of goal-directed reasoning, this section

reviews common goal-directed reasoning approaches, to point out how they differ from ACE-SSM's approach. While most of these approaches use goal-directed reasoning mainly for control purpose, ACE-SSM's motivation behind goal-directedness is the desire to make SSMs explicit. To clarify this important difference, this section also uses an example from medical diagnosis to provide more intuition into the meaning of SSMs and how they relate to goal knowledge.

2.1 Goal-Directed Reasoning

Conventional rule-based systems (RBSs) apply the simplest approach to goal-directed reasoning [23]. They use information about their goals and relationships between them to guide control. This is apparent in backward-chaining RBSs, where problem-solving begins with the goal of deriving a particular data item. This item is matched against the consequent of rules, to determine which rules can derive it. The antecedents of rules that meet this criterion become then subgoals, which need to be achieved recursively. Thus, goals are built into object-level rules, to form a static AND/OR goal tree -- or *goal structure* -- for the application task. In each problem situation, only the relevant portion of the goal structure is instantiated by fired rules. Overall, RBSs' goal-directedness is rather implicit: there is no distinction between data elements serving as domain objects and those serving control purposes, and goal-directed control knowledge comprises only general-purpose heuristics built into procedural conflict resolution mechanisms.

GRAPES [24] is a RBS architecture that makes goal-directedness more explicit. Like in common RBSs, the goal structure for the task is hardwired into object-level rules, where rule antecedents match against goals and rule consequent post new goals. But, in GRAPES, the (sub)goals relevant to the problem situation solved are posted into a *goal memory*, which is distinct from the working memory. Goals in the goal memory are organized hierarchically, where the subgoals form AND/OR branches. The key benefit relative to control is that rules can monitor interactions between subgoals in the goal memory -- update their status (successful, inhibited, etc.), sequence them, record their failure for backtracking purposes, etc. -- as a way to avoid inefficient chaining of rules. Additionally, aside from basic RBS conflict resolution strategies, it is possible to use meta-rules capturing domain-specific control knowledge for selecting the subgoal to pursue in each inference cycle. Thus, in GRAPES, goal-directed control knowledge can reside in procedural conflict resolution mechanisms as well as declarative meta-rules.

In other RBS schemes, like AMORD [11] and GDD [22], a reasoning component posts goals and then a planning component determines the order of pursuing these goals. AMORD is a general-purpose solver that combines truth maintenance and planning facilities. It uses rules that post *inference goals* to its reasoner and rules that post *action goals* to its planner. The reasoner uses inference goals for meta-reasoning purposes, to avoid combinatorial forward-chaining and to manage backtracking. GDD, which builds on AMORD's approach, is a diagnosis-and-repair reasoner in which all goals are action goals built into object-level rules. GDD's notion of goal is derived from the attitude assignment (e.g., relevant, irrelevant) to a given action. Actions in GDD rules have truth (belief) and relevance values, both of which the reasoner updates based on the rules fired in each cycle. These rules can post the goals relevant to the problem situation solved, assign a relevance value to posted goals, inhibit non-contributing goals, etc. In each inference cycle, the list of all goals posted and judged relevant is sent to a planner, which generates a plan specifying the order in which they should be pursued. Hence, instead of using explicit goal-directed control knowledge, GDD and AMORD delegate control decisions to their planning component.

SOAR [20], a general-purpose solver built based on the problem-space paradigm, uses a different approach to goal-directed reasoning. In each inference cycle, given a problem space along with a description of some current state and a goal state in that space, SOAR fires all instantiations of the operator (rule) that applies to the current state. Within the context defined by the problem space, each instantiation inserts into the working memory actions -- new reachable states in the problem space and operators to apply to these states -- with preferences assigned to these actions. To choose which posted action to pursue in the next cycle, SOAR uses a *decision procedure* to analyze the preferences of posted actions.

Thus, preferences assigned to actions constitute domain-specific control knowledge that is built into rules. When the decision procedure cannot make a choice based on assigned preferences, SOAR runs into an *impasse*. To resolve the impasse, SOAR uses no conflict resolution mechanisms. Instead, it posts a new subgoal within a new lower-level subcontext corresponding to a different problem space. Thus, subgoals arise when SOAR does not know what to do next. For example, if SOAR cannot decide which of two actions to apply, a newly set subgoal leads SOAR to try both actions to find out which one is preferable. While this may seem an inefficient way to act on lack of control knowledge, its purpose is to facilitate learning. SOAR uses a *chuncking mechanism* to synthesize a new rule capturing the knowledge gained form resolving the impasse so that this impasse can be avoided in the future. Overall, SOAR's goal-directed reasoning approach is anchored in the desire to present an architecture of intelligence in which all knowledge is represented as declarative object-level rules (as opposed to meta-rules or procedural tasks).

Work on PSMs and GTs offers another approach to goal-directed reasoning [5]. This approach represents a task as a hierarchy of procedural subtasks, where each subtask is equated with a subgoal entailing the conversion of a given input into a desired output. While the goal structure is fixed into a subtasks hierarchy, goal-directedness is explicit in the sense that the goals and subgoals relevant to each problem situation are dynamically posted in, and managed using, a stack (i.e., conventional task stack). This approach is motivated by the desire to make control knowledge reusable across similar application tasks, by separating it from domain knowledge. On this premise, there have been developed some well known automated knowledge acquisition tools (e.g., SALT [17]), meta-level knowledge acquisition tools (e.g., PROTEGE-II [21]), and libraries of generic PSMs and GTs (e.g., TINA [2], DSPL/GT [15]).

So far we see that different goal-directed reasoning approaches vary on their motivation. Basic rule-based schemes and their variations use goal-directedness to improve control. SOAR uses goal-directedness mainly to facilitate learning when adequate control knowledge is not available. In PSM/GT-based approaches, goal-directedness helps to separate domain knowledge from control knowledge as a way to make the latter reusable across similar tasks.

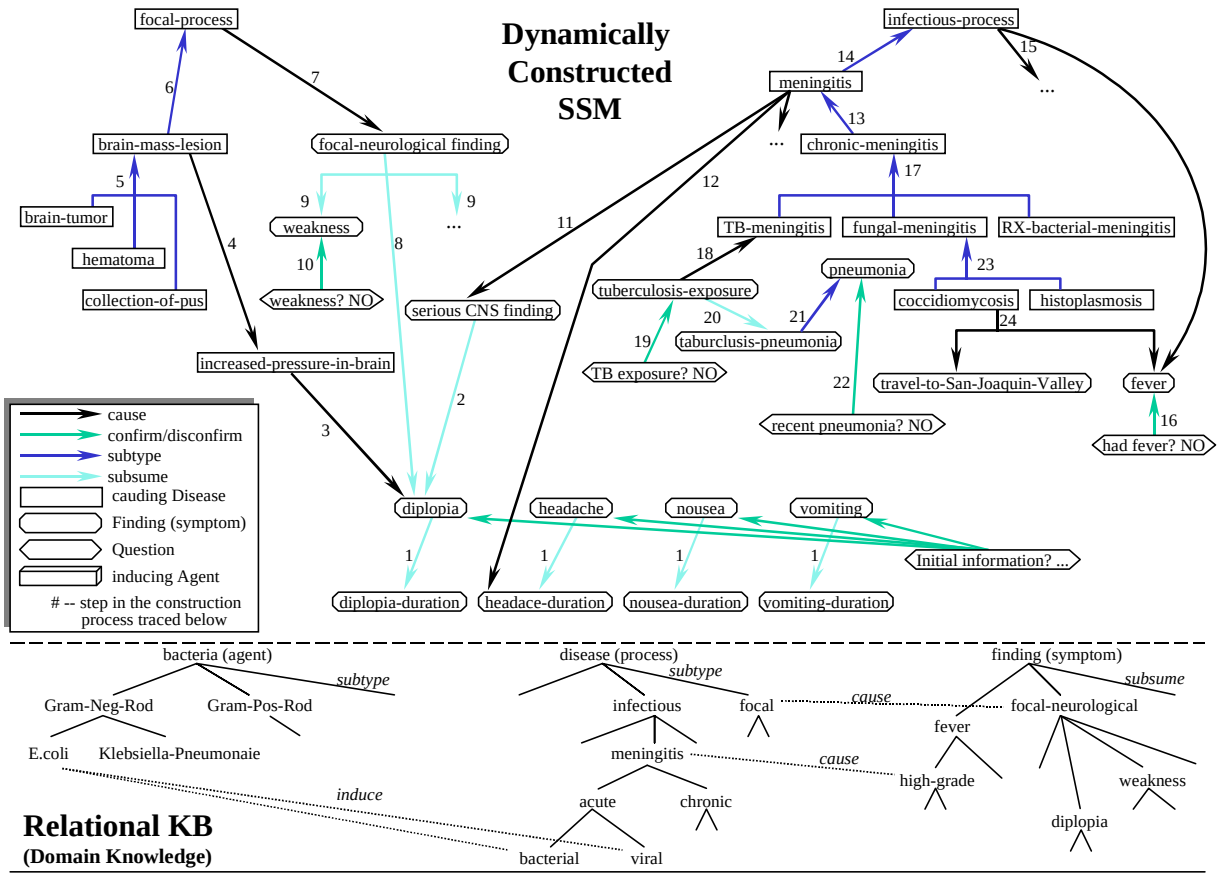
The motivation behind ACE-SSM's approach to goal-directed reasoning is different. In considering the goal of a KBS to be one of producing SSMs, its intention is to anchor the system's reasoning in the desire to make these SSMs explicit. For this purpose, ACE-SSM uses explicit goal knowledge describing the structure (ontology) of SSMs the system seeks to produce. To clearly see what this exactly means, we must first explain the notion of SSM.

2.2 Goal-Directedness and Situation-Specific Models

To understand the notion of SSM, it is useful to look at an example. Figure 1 shows a sample SSM constructed in the context of NEOMYCIN's medical diagnosis task [7].[1] On the surface, this sample SSM is simply a directed graph linking specific findings (symptoms), diseases, inducing agents, etc. that were instantiated because they were found to be relevant to the problem situation solved. In this sense, relative to RBSs, this SSM may appear to be nothing more than a complete and well-organized working memory that shows how its elements relate to each other as well as the order in which they were instantiated. Likewise, relative to a KB comprising explicit tangled relational networks, this SSM may appear to be just the portion of the KB that was instantiated by spreading activation.

However, a deeper look shows that this SSM is more than that. Where the goal of a KBS is to construct an SSM for every problem situation it solves, specific nodes and links are present in that SSM by virtue of meeting certain constraints — termed *goal constraints* — that dictate which particular relations between specific ontological objects must hold true in the application domain. For example, the SSM in Figure 1 grew a link from diplopia to pressure-in-the-brain due to a goal constraint stating that every abnormal finding must be explained by a causing disease. Generally, such goal constraints define the

[1] The graphical notation used in Figure 1 is not some new formalism – it only serves the purpose of illustrating the structure of the sample SSM.



Steps in the Construction Process (adapted from [7])

The patient is a 15-year-old female with a two-week history of headache, nausea, vomiting, and diplopia (double vision).

1. Based on the history, I map these findings to: headache-duration, nausea-duration, vomiting-duration, diplopia-duration.
2. I focus on diplopia first because it is an abnormal, serious-CNS (central nervous system)-finding.
3. Diplopia is caused by increased-pressure-in-the-brain. As pressure builds up, a nerve that enervates the eyes movement is impaired.
4. Increased-pressure-in-the-brain may be caused by a some type of a brain-mass-lesion.
5. A brain-mass-lesion could more specifically includes: brain-tumor, hematoma (collection of blood), collection of pus.
6. Brain-mass-lesion is a focal process.
7. A focal process always has focal-neurological manifestations.
8. We know that diplopia is one focal-manifestation.
9. Other focal neurological findings include one-sided hand or leg weakness.
10. Does the patient have any weakness anywhere in her body? One side weaker than the other? NO. So, the possibility of a focal process is ruled-out for now.
11. The serious-CNS-finding also provides evidence for meningitis.
12. Meningitis also causes headache-duration, which is already known to be one of the patient symptoms.
13. Given the long history, if it is meningitis it is a chronic-meningitis.
14. Chronic-meningitis is a systemic infectious-process,
15. and fever is one evidence for infection.
16. Has she had fevers? NO. This question is important to distinguish between an infectious and a non-infectious cause. But, no fever does not rule out an infection. We are dealing with is a chronic process, with which fever can sometimes be low or none at all.
17. Focusing on chronic-meningitis leads to several more specific possibilities -- TB-meningitis, fungal-meningitis, and partially rx-bacterial-meningitis.
18. I look at TB risk because we are facing an indolent (chronic) infection. Thinking of infections, chronic ones are most likely, even without any lab data. Tuberculosis-exposure can be one evidence for TB-meningitis.
19. Has she had any exposure to tuberculosis? No.
20. So, TB risk is negative. But we also know that TB risk subsumes tuberculosis-pneumonia,
21. and pneumonia subsumes tuberculosis-pneumonia.
22. No recent pneumonia that she knows of? No.
23. That's pretty solid evidence against TB-meningitis. Hence, I focus next on fungal-meningitis. Likely causes of fungal-meningitis are coccidiomycosis and histoplasmosis.
24. Among other things, fever and travel-to-San-Joaquin-Valley are evidence for coccodiomycosis. We already know that the patient has no fever, so coccodiomycosis is ruled-out.
25. Etc.

Figure 1: sample SSM and a trace of its construction process

semantics of domain phenomena modeled using SSMs. Specifically, in case of our sample SSM, once complete, this SSM would capture a causal argument showing what has happened to bring about the patient state of illness. More precisely, as we shall see in Section 3 (and Figure 3), the structure of this SSM reveals that NEOMYCIN's diagnosis task models a disease as a process that follows a specific causal script — an agent (e.g., E.coli) enters the body when given the opportunity (e.g., during surgery), moves to some organ system (e.g., meninges of the brains) where it proliferates, inducing a disease (e.g., bacterial meningitis) that causes symptoms (e.g., durable headache), directly or indirectly through certain pathologic states (e.g., inflated brain tissue).

Relative to this characteristic of SSMs, ACE-SSM interprets goal-directedness as follows. Given a set of goal constraints relevant to the application domain of a target task, the goal of the task in solving a specific problem situation is to produce an SSM that contains all given inputs and meets all the goal constraints. Here, a subgoal is equated with the violation of a certain goal constraint by a specific node in the SSM, and achieving a subgoal means adding to the SSM nodes and links that resolve the violated constraint, where the newly added nodes may yield new violations of goal constraints. Thus, ACE-SSM's goal-directedness serves the purpose of anchoring a system's reasoning in the desire to evolve SSMs into a state that meets all relevant goal constraints. Aside from the benefits derived from having explicit SSMs, benefits on which we elaborate later, ACE-SSM's approach to goal-directed reasoning offers two unique features. It makes explicit goal knowledge *G* describing the structure of SSMs (governed by goal constraints), thus allowing a KBS to be aware of its goals. In addition, it allows representing goal-directed control knowledge *S* declaratively, in terms of pertaining to *G*. Together, these features lead to several potential benefits, alluded to in Section 1 and further discussed in Section 5. To make these claims more concrete, we next provide a detailed description of ACE-SSM.

3 ACE-SSM: ARCHITECTURE FOR CONSTRUCTING EXPLICIT SSMS

In this section, we first provide a high-level overview of how ACE-SSM works, focusing on how it uses goal knowledge *G* to drive the construction of SSMs. We then explain how *G*, *K* and *S* are represented and used in ACE-SSM. To illustrate key ideas, while keeping the discussion on track, we use a somewhat simplified version of the sample medical diagnosis task mentioned earlier. We consider the task to involve:

(1) single-fault diagnosis;

(2) multiple findings, $|F\{D\})|>1$, where $F\{D\}$ is the set of findings (symptoms) covered by disease D;

(3) independence of findings, $F\{D_1, D_2\}=F\{D_1\}\cup F\{D_2\}$; and

(4) overlapping findings, $F\{D_1\}\cap F\{D_2\}\neq\varnothing$.

However, the discussion allows seeing how ACE-SSM would work under more complicated assumptions explained in [27, p. 689], such as: composite diagnoses, compensating findings (i.e., $F\{D_1, D_2\}\subset F\{D_1\}\cup F\{D_2\}$), and synergetic findings (i.e., $F\{D_1\}\cup F\{D_2\}\subset F\{D_1, D_2\}$).

3.1 Conceptual Underpinning

The reasoning process underlying the construction of SSMs can be explained using Newell's problem-space paradigm of intelligence [20]. This paradigm considers problem-solving to be a process that involves two nested searches -- problem search in the outer loop and knowledge search in the inner loop.

- *Problem search* assumes the existence of a problem space for the task solved. This implies the availability of: a goal state s_g that describes what needs to be known at the end of problem solving, an initial state s_i that describes what is known at the start of problem solving, and a set of operators *O* for generating new states in the problem space. Starting from the current state, $s_c=s_i$, problem search applies one operator in *O* at a time to generate a new state s_c', with the hope that s_c' will match, or at

least be closer to s_g. Thus, problem search constructs the problem space dynamically, as it uses operators in O to generate states that form a path from s_i to s_g.

- *Knowledge search* occurs in the inner loop within problem search. It guides problem search by using search control knowledge to select the operator in O to apply so as to generate the next state, s_c', on the path from s_i to s_g. Knowledge search usually occurs within a fix space whose structure (connectivity and how paths through it are described) pre-exists. For example, in RBSs, this search applies to the space defined by fix conflict resolution mechanisms. Only in a few approaches, such as SOAR's, is this space constructed dynamically.

Framing the construction process of SSMs within the problem-space paradigm requires reliance on several postulates. First, where the goal of solving a task is to create an SSM of every problem situation the task involves, let goal knowledge G reflect the structure of these SSMs in terms of the goal constraints that they must satisfy. Second, relative to problem search, we say that $\boldsymbol{G}$ implicitly describes the goal state, s_g, and the evolving SSM being created for a given situation explicitly describes the current state, s_c. Finally, let K be domain knowledge (data) used to populate SSMs, and S be the search control knowledge used in knowledge search.

With these postulates in mind, we summarize below the three-steps iterative process that ACE-SSM uses to construct explicit SSMs in terms of problem search and knowledge search (see Figure 2).

Step 1: In problem search, inspect the evolving SSM_j (current state s_c) to identify violations of goal constraints that correspond to gaps relative to G (goal state s_g), and then post the gaps in the SSM as subgoals. For example, consider an SSM with one abnormal finding node bound to diplopia, denoted (*F* diplopia). Here, violation of a constraint like "an abnormal finding must be linked to a causing disease" would identify one gap -- the lack of a disease node in the SSM that explains this finding. Respectively, a subgoal is posted in the SSM as an *unsatisfied node-chain*, denoted (*D* ?)→(*F* diplopia), where (*D* ?) is an unbound disease node. Thus, where $G(SSM_j)$ denotes the effect of applying goal constraints in G to SSM_j, what this step does can be written shortly as:

$$G(SSM_j) = SSM_j' = SSM_j \cup \{g \mid g \text{ is a subgoal posted for a violated goal constraint in } G\}.$$

Step 2: In knowledge search, given SSM_j' posting all identified gaps as unsatisfied subgoals, use search control knowledge S to choose which subgoal $g \in SSM_j$' to satisfy next by producing a new state that closes one of the gaps in SSM_j'. For example, suppose that the choice has to be between two subgoals represented by the unsatisfied node-chains (*D* ?)→(*F* diplopia) and (*Fg* ?)→(*F* diplopia), where (*Fg* ?) is an unbound finding node denoting a more general finding subsuming diplopia. Here, a search control principle like "generalize findings before searching for their causes" would dictate pursuing (*Fg* ?)→(*F* diplopia) first. Hence, where $S(SSM_j')$ denotes the result of applying S to SSM_j', and g is a subgoal in SSM_j', the output of this step is written shortly as:

$$S(SSM_j') = g.$$

Step 3: Back in problem search, apply a problem-space operator that is able to instantiate those elements in K needed to satisfy the subgoal g chosen in step 2. For example, for subgoal (*Fg* ?)→(*F* diplopia), a proper operator would search a taxonomy of findings in K to bind (*Fg* ?) to a more general subsuming finding such as serious-CNS-finding. Thus, where $K(g \in SSM_j')$ denotes the result of applying K to subgoal $g \in SSM_j'$, and g^* denotes g satisfied, what this steps does is written shortly as:

$$K(g) = g^*.$$

Upon updating SSM_j' with the results from step 3, ACE-SSM produces SSM_{j+1}, the next state in the problem space. Because SSM_{j+1} can still contain unsatisfied subgoals — old ones not yet pursued and new ones that the node(s) newly added to SSM_j' define relative to G — the three steps are repeated until the SSM corresponds to a state in the problem space that meets all goal constraints in G.

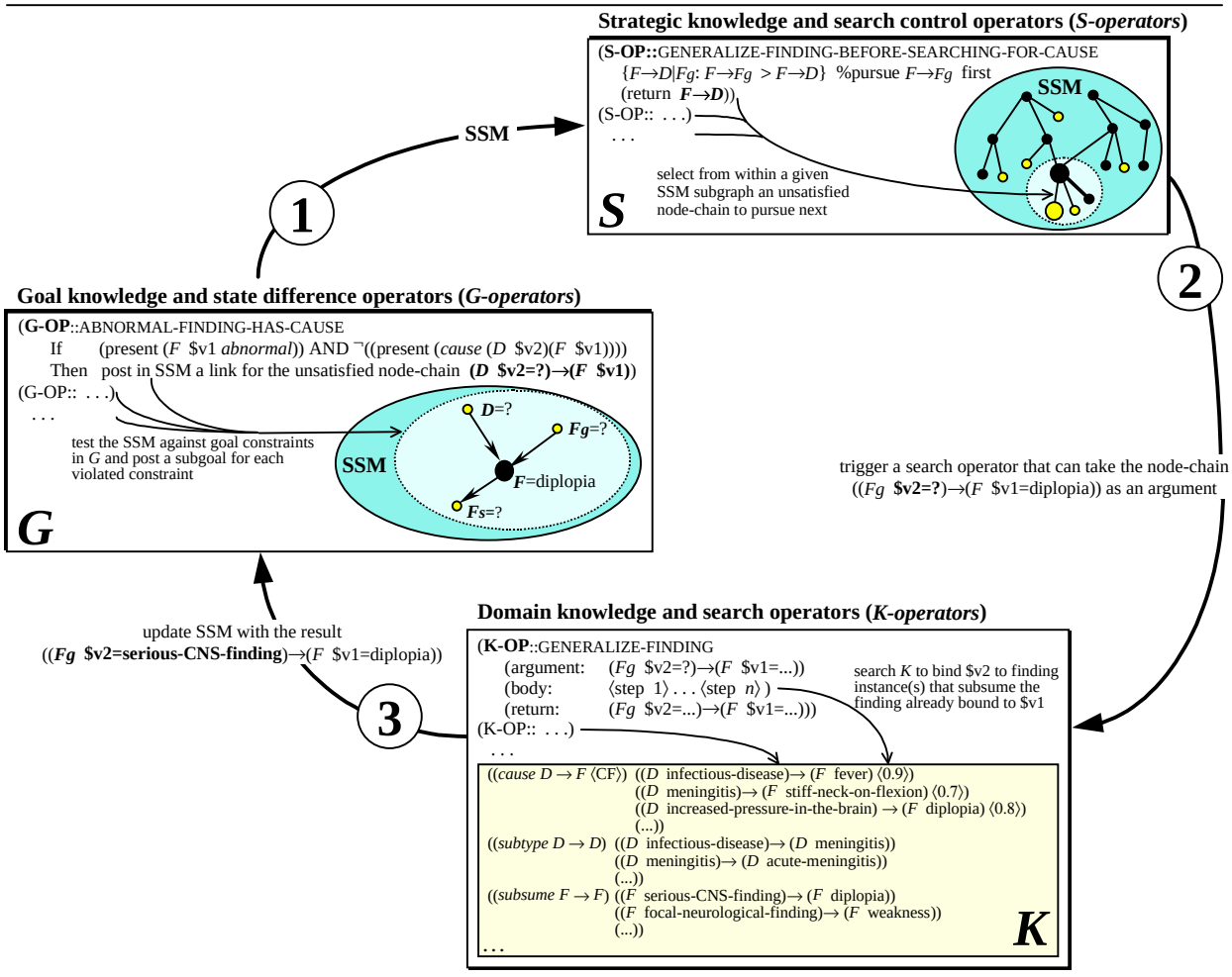


Figure 2: ACE-SSM's iterative construction process of an SSM

Several clarifications are in order regarding how ACE-SSM implements the above process. In particular, how are goal constraints comprising *G* represented and applied? how are subgoals posted in SSMs? how is domain knowledge *K* used to satisfy subgoals? and, how is search control knowledge *S* represented and used to guide the construction process of SSMs? The answer to these and other questions is provided in the rest of this section.

3.2 ACE-SSM's Approach Formalized

Before we explain the internal workings of ACE-SSM, we need to define the notion of SSM more precisely.

- An *SSM* is a digraph, ⟨*N*,*E*⟩, where *N* is a set of nodes and *E* is a set of directed edges connecting nodes in *N*.
- A *node* $n \in N$ is a tuple (*o i* [*a*] ⟨*t*⟩), where *o* is an ontological object, *i* is an instance of *o*, [*a*] is an optional attribute value of *i*, and ⟨*t*⟩ is a truth value or certainty factor (CF). For example, node (*F* diplopia abnormal ⟨1.0⟩) denotes an abnormal finding called diplopia with a 1.0 CF. A *terminal node* (or leaf) $n \in N$ is an unbound node, denoted (*o* ?). A node is unbound if its *i* value is unknown, *i*=?, its truth value is unknown, ⟨*t*⟩=?, or both.

- An *edge* $e \in E$ links two nodes, creating a *node-chain*, e.g., $(D \text{ infectious-disease } \langle 0.9 \rangle) \rightarrow (F \text{ fever } \langle 1.0 \rangle)$. An edge with a terminal node is equivalent to an *unsatisfied node-chain* with one unbound node, e.g., $(Fg \text{ ?}) \rightarrow (F \text{ diplopia } \langle 1.0 \rangle)$, from which the notion of *subgoal* is derived.

With these definitions, we are now ready to elaborate on ACE-SSM's internals.

3.2.1 Reasoning with Goal Constraints

Goal knowledge G describes the structure of the SSMs created for a target task. It is a set of goal constraints (axioms, assertions, assumptions) that these SSMs must satisfy. In G, we distinguish between *local* and *global* goal constraints, denoted G_L and G_g, respectively.

Local goal constraints in G_L define direct relations between types of domain objects that must hold true in every SSM created for the task. For our sample medical diagnosis task, two such constraints are:

(G_{L1}) every SSM node depicting an observed abnormal finding must be causally linked to a disease node that explains (covers) it, and

(G_{L2}) every SSM node depicting an inferred (non-observed) abnormal finding linked to a disease node must be verified with the patient (asked about).

Goal constraints in G_L are written formally as quantified first-order predicate formulae involving unary and binary function constants or relational constants. For example, the above constraints are written as:

(G_{L1}') $\forall F \text{ abnormal}(F) \wedge \text{observed}(F) \Rightarrow \exists D \ \textit{cause}(D,F)$

(G_{L2}') $\forall F \text{ abnormal}(F) \wedge \text{inferred}(F) \wedge (\exists D \ \textit{cause}(D,F)) \Rightarrow \exists Q \ \textit{confirm}(Q,F)$

where F is a finding object, D is a disease object, Q is a question (test) asked about F, abnormal is a unary function constant corresponding to an attribute of F, *cause* and *confirm* are binary relational constants, and observed and inferred are themselves formulae. Specifically, observed checks if there is a satisfied node-chain of the form $F \rightarrow Q$ (i.e., a finding confirmed by a question/test), and inferred checks if there is a satisfied node-chain of the form $F \rightarrow Fs$ (i.e., a finding established by a more specific subsumed finding).

Each goal constraint $\gamma \in G_L$ is applied by a dedicated *G-operator*. Where g_i is the unsatisfied node-chain or subgoal that G-operator i can post if $\gamma_i \in G_L$ is violated, applying this G-operator has the effect:

$$\text{G - operator}_i(\text{SSM}) = \begin{cases} \text{SSM} \cup g_i & \text{if the operator fired} \\ \text{SSM} & \text{otherwise} \end{cases}$$

Thus, a G-operator is a goal setting rule. It returns an unsatisfied node-chain corresponding to the consequent of constraint $\gamma \in G_L$, provided that γ is violated. For example, a G-operator for constraint (G_{L1}') is written in pseudo-code as:

```
(G_OP::DISEASE_CAUSE_FINDING(F $var1)            % argument: a bound Finding node in the SSM
    if (F $var1) is abnormal and                 % if goal constraint on F is violated
       (F $var1) is observed and
       (F $var1) is not linked to a bound disease node
    then return((D  ?)→(F $var1))).              % return an unsatisfied node-chain
```

If fired, this G-operator returns **(D ?)→(F $var1)**, which ACE-SSM uses to post a new subgoal by adding to the SSM a link between **(F $var1)** and the new unbound node **(D ?)**.

Based on the object types, relations and attributes that constraints in G_L identify, we can draw an entity-relationship (ER) diagram like the one in Figure 3. This diagram reveals three related things. First, it shows that, for an application task that is sufficiently understood, G_L captures the semantics of domain

phenomena with which the object reality of the task is concerned. By pursuing the direction of arrows in Figure 3, we see that our task models a disease as a bodily process that follows the specific causal script we described in Section 2.2. Second, the diagram reflects the task ontology – the explicit list and organization of terms that constitute a representational scheme for the specific reality with which the task is concerned [13]. Thus, it also reflects the topology of domain knowledge; every binary relation between object types directly maps to a relational network in *K*, as evident from the partial instantiation of the three networks seen in Figure 2. Finally, the diagram reveals the underlying structure of SSMs constructed for the task.

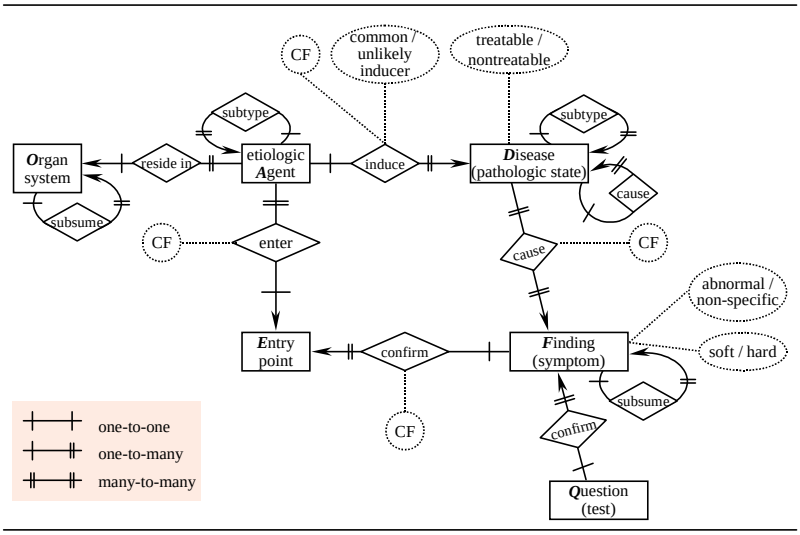


Figure 3: structure of SSMs reflecting the semantics of domain phenomena defined by goal knowledge

Relative to the semantics of domain phenomena G_L defines, G_g is a single *global goal constraint* reflecting termination conditions that an SSM must meet to qualify as the solution sought. This constraint relies on the meaning of SSM subgraphs. Because an SSM is a digraph that gradually grows edges, it could embody several subgraphs. For example, at some point the SSM in Figure 1 contains two subgraphs: one whose root is brain-mass-lesion, another whose root is meningitis. For our sample task, each SSM subgraph represents a competing, and possibly incomplete, candidate solution (or disease process description). Any such subgraph is the solution to a tackled problem situation, if it represents an instance of a causal process that implicates the diagnosed disease depicted by its root. Relative to this notion, termination conditions in G_g apply to the root of every candidate SSM subgraph, specifically:

(***G*****$_g$**) the root of an SSM subgraph depicting the disease process sought:

(1) must be the most specific disease possible (to allow prescribing the most correct cure),

(2) must cover (explain) every abnormal finding known to be present in the patient,

(3) etc.

According to condition (1), the roots of SSM subgraphs depicting competing candidate solutions form what medical diagnosis calls a *diagnostic differential* -- the set of most specific disease hypotheses. Clearly, this G_g assumes that our sample task is a single-fault diagnosis task. A different diagnosis task would require a different global goal constraint. For example, a task that seeks to find a minimal cover for the observed symptoms would require that SSM subgraphs spanned by a minimal number of *D* nodes cover all *F* nodes.

Like with constraints in G_L, G_g can be expressed as a first-order predicate formula, and it is captured using a G-operator. But, while applying constraints in G_L may result in posting subgoals,

applying G_g has the effect:

$$G_g(\text{SSM}) = \begin{cases} \textit{true} & \text{if all termination conditions in } G_g \text{ are met} \\ \textit{false} & \text{otherwise} \end{cases}.$$

All constraints in *G* share three important characteristics. First, they involve functions that access only the SSM, not *K*. Second, they can be expressed in domain-specific terms reflecting the semantics of SSM node-chains, or also using graph-oriented terms pertaining to the topology of SSM; e.g., *subtype*(·,·) can be replaced by *child*(·,·). Finally, together, constraints in *G* define a meta-level goal structure for the task -- a goal structure template based on which ACE-SSM dynamically constructs the goal structure relevant to each problem situation.

Evident from the discussion so far, ACE-SSM differs in two ways from the goal-directed reasoning approaches reviewed earlier. First, while these approaches (except for SOAR) attempt to be parsimonious in posting subgoals, ACE-SSM follows a different principle: *post all potentially relevant subgoals in the SSM and worry later about which of them to pursue and when*. Given a meta-level goal structure, ACE-SSM *must* post all potentially relevant subgoals; most often, before SSM nodes are bound, there is no way of knowing in each case which subgoal is truly necessary. Second, ACE-SSM does not need to manage subgoal interactions, largely due to the nature of subgoals posted in SSMs. (Recall that a subgoal is an unsatisfied node-chain entailing the most elementary inference action – binding an SSM node to an object instance, to a truth value, or to both.) Because goal-setting rules (G-operators) can have only a header that is both positive and atomic, subgoals are not hierarchical. Only in an indirect sense are all subgoals considered conjunctive relative to the global goal constraint G_g. Consequently, ACE-SSM treats subgoals as being independent, except for the fact that they compete for the same resources (agent's attention, time, etc.). In managing the allocation of resources, ACE-SSM assumes that all precedence relations between subgoals are captured by explicit strategic control principles in *S*. For example, the principle "generalize a finding before searching for its causes" implies pursuing subgoal ?*Fg*→*F* before subgoal ?*D*→*F*. We show later how *S* is represented and used. Hence, once subgoals are posted, ACE-SSM leaves it up to explicit *S* to determine which subgoals to pursue and when.

3.2.2 Using Object-Level Domain Knowledge

Assuming for now that ACE-SSM already decided on which subgoal in the SSM to pursue next, how is this subgoal achieved? There is a *K-operator* for satisfying every type of goal posted in the SSM -- there is a one-to-one mapping from G-operators to K-operators. Where G-operator$_i$(SSM)=g_i, and g_i* denotes g_i satisfied,

$$\text{K-operator}_i(\text{G-operator}_i(\text{SSM})) = g_i^*.$$

K-operators use *K* to derive a value to which to bind the unbound node in posted subgoals. For example, a K-operator for handling the subgoal (*D* ?)→(*F* $var1) is:

```
(K_OP::LINK_F_TO_CAUSING_D((D ?$var2)→(F $var1))   % argument: node-chain with unbound node
                                                   % body: code that instantiates elements
            < ... body ... >                       %    in K to bind $var2 to the disease(s)
                                                   %    causing the finding bound to $var1
      return((D  $var2)→(F $var1))).               % return: satisfied node-chain with the
                                                   %    unbound node bound to proper
                                                   %    instances
```

Two observations are in order relative to K-operators. First, when a K-operator cannot find instances in *K* to which to bind node *n*∈ SSM, it binds *n* to NIL, to let ACE-SSM know that an attempt to pursue a subgoal involving *n* was made. For example, trying to satisfy (*Dg* ?)→(*D d*), where *d* is the root of a disease taxonomy, would bind *Fg* to NIL. Second, the body of a K-operator can take different forms, depending on how *K* is coded. Where *K* comprises explicit relational networks, the body would apply graph-

oriented operations on *K* (e.g., get children, find father). If instead *K* comprises IF-THEN rules, for example, where the rules capture the ontological identity of object instances they reference, K-operators could activate a rule interpreter on a proper subset of the rules in *K*. Hence, all implementation details of *K* need to be known only to K-operators. Regardless, we assume hereafter that *K* is represented as relational networks.

For K-operators to do their job, links exist between *G* and *K* through common reference to the ontological identity of objects. When *K* is a set of relational networks, each network has the signature:

$$\pi([\wedge/\vee]\ n \rightarrow n\ [\langle p \rangle]),$$

where π is a relation, $[\wedge/\vee]$ is an optional logical operator indicating whether sentences in the network are conjunctive or disjunctive, *n* is a tuple node of the form (*o i* [*a*]), and $[\langle p \rangle]$ is an optional a-priori probability (i.e., CF). For example, the sentence *cause*((*D* bacterial-meningitis)→(*F* high-grade-fever abnormal) ⟨0.9⟩) means that high-grade-fever is an abnormal finding caused by bacterial-meningitis with a 0.9 CF. We say that every sentence coincides with a specific *node-chain* instance, e.g., (*D* bacterial-meningitis)→(*F* high-grade-fever) for the last example. The following shows a hypothetical conjunctive network linking diseases and findings:

$$\begin{aligned}&((cause \wedge (D \rightarrow F\ \langle cf \rangle))\\ &\quad ((D\ d) \rightarrow (F\ f_1)\ \langle 0.7 \rangle)\\ &\quad ((D\ d) \rightarrow (F\ f_2)\ \langle 0.3 \rangle)\\ &\quad (\ldots)).\end{aligned}$$

Here, if only the first two sentences apply to *d*, conjunctively they imply that *d* has a 0.79 CF (i.e., 0.7+0.3*(1–0.7)).

Having mentioned CFs, ACE-SSM uses these to manage uncertainty like in conventional RBSs (e.g., see [10]). For example, consider the above hypothetical network that links diseases and finding. Given the unsatisfied node-chain (*D d*)→(*F* ?), a K-operator that addresses this kind of node-chain would return:

$$((D\ d) \rightarrow (((F\ f_1)\ \langle cf_1 \rangle) \wedge ((F\ f_2)\ \langle cf_2 \rangle))),$$

based on which ACE-SSM would update the CF of (*D d*) in the SSM to be $cf_1+cf_2*(1-cf_2)$. Likewise, for a disjunctive relational network linking diseases and their inducing agents, a K-operator for addressing (*A* ?)→(*D d*) would return:

$$(((A\ a_1)\ \langle cf_1 \rangle) \vee \ldots \vee ((A\ a_m)\ \langle cf_m \rangle)) \rightarrow (D\ d)),$$

and ACE-SSM would update the CF of node (*D d*) to be $\max(cf_j)_{j=1\ldots m}$. It is easy to see how ACE-SSM can be augmented to handle uncertainty using a multi-valued logic [12], for example, one which allows every proposition (and SSM node) to have a truth value in {*true*, *false*, *unknown*}.

3.2.3 Applying Strategic Control Principles

During the discussion of how subgoals are posted and achieved, we assumed that ACE-SSM uses explicit search control knowledge, *S*, to decide on which subgoal to pursue and when. *S* comprises strategic principles that yield the abstract control pattern shown in Figure 4. This figure distinguishes *local strategic principles*, S_L, from *global strategic principles*, S_G. It does so on the premise that an SSM is a digraph embodying several subgraphs whose edges correspond to node-chain instances. For example, as Figure 1 shows, the diplopia finding is eventually mapped to two causing diseases – brain-mass-lesion and meningitis – that become the roots of different SSM subgraphs depicting two competing candidate solutions. The role of S_G is to focus attention to a specific SSM subgraph, or candidate solution, denoted:

$$S_G(\text{SSM}) = \text{SSM subgraph.}$$

Principles in S_G take affect only occasionally, when certain changes to the SSM require to divert attention

from one SSM subgraph to another. By contrast, the role of S_L is to determine at every moment which subgoal $g \in S_G(\text{SSM})$ to pursue next, denoted:

$$S_L(S_G(\text{SSM})) = g.$$

Principles in S_L direct problem-solving most of the time, as $S_G(\text{SSM})$ continues to change by gradually growing new edges due to spreading activation of elements in K.

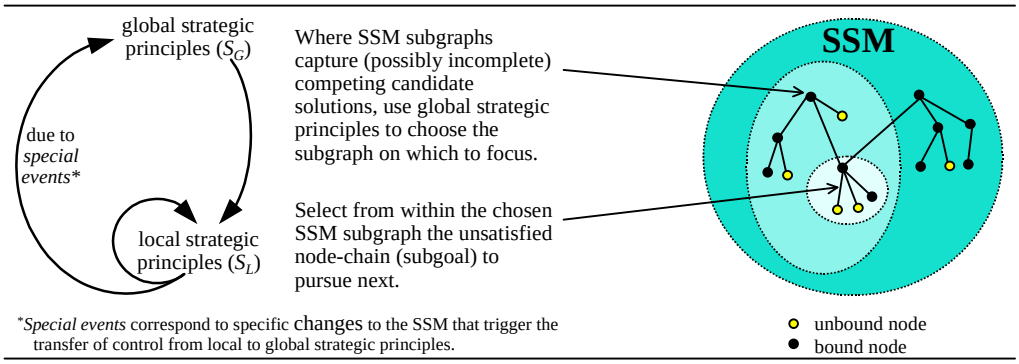


Figure 4: roles of strategic control principles

Before we explain how principles in S_L work, we need to clarify certain points. First, at any moment ACE-SSM focuses on one specific bound node in $S_G(\text{SSM})$, termed the *focus node* and denoted f (e.g., diplopia). Second, granted that f is the tuple $(o\ i\ [a]\ \langle t \rangle)$, we define object($f$) to be a function that returns the ontological identity of f,

$$\text{object}(f) = o,$$

and define $G(\text{object}(f))$ to be

$$G(\text{object}(f)) = \{g \mid g \text{ is a subgoal associated with object}(f)\}.$$

For example, if object(f)=D, $G(D)=\{?Dg \rightarrow D,\ D \rightarrow ?Ds,\ D \rightarrow ?F,\ ?A \rightarrow D\}$, where $?Dg \rightarrow D$ maps f to a more general disease of which it is a subtype, $D \rightarrow ?Ds$ maps f to more specific diseases of its type, $D \rightarrow ?F$ maps f to findings it causes, and $?A \rightarrow D$ maps f to its inducing agents (see Figure 3).

Principles in S_L prescribe the order of pursuing subgoals in $G(o)$, for every ontological object o the task involves. For instance, for o=D, a principle like "test a disease hypothesis before refining it" prescribes pursuing $D \rightarrow ?F$ before $D \rightarrow ?Ds$, denoted $\{D \rightarrow ?F > D \rightarrow ?Ds\}$, where $>$ is a binary order (precedence) relation. Thus, where object(f)=o, every principle $\sigma_i \in S_L$ pertaining to object o imposes order on a subset $G'(o)$ of the elements in $G(o)$, denoted:

$$\sigma_i\,(G(o)) =_{\text{def}} >_{e \in G'(o) \subseteq G(o)}.$$

Respectively, if we define $S_L(o) \subseteq S_L$ to be $S_L(o)=\{\sigma \mid \sigma \in S_L \text{ applies to object type } o\}$, then

$$S_L(G(o)) =_{\text{def}} >_{e \in G(o)}.$$

For instance, for the above example, $S_L(G(D))=\{D \rightarrow ?F > ?Dg \rightarrow D > D \rightarrow ?Ds > ?A \rightarrow D\}$.

In the same spirit, where f is the focus node, we define attributes(object(f)) as:

$$\text{attributes}(\text{object}(f)) = \{\text{object}(f)_{[a]} \mid a \text{ is an attribute of object}(f)\}.$$

For example, if object(f)=F, attributes(F)=$\{F_{[\text{abnormal}]},\ F_{[\text{non-specific}]},\ F_{[\text{soft}]},\ F_{[\text{hard}]}\}$. Note that, depending on the domain, findings can have other attributes, like *utility* or *cost*, which reflect the value or effort associated with confirming them. The attributes of a focus node f are needed when f is in multiple node-chains of the

same type. For example, given a set of known findings $\{F_i\}$ and assuming that S_L dictates pursuing $?D \rightarrow F$ next, which finding in $\{F_i\}$ to map first to its causing diseases? The answer is provided by a principle like "always pursue abnormal findings first", which uses the abnormality attribute of F to discriminate between the subgoals considered, $\{?D \rightarrow F_i\}$. (In the sample SSM in Figure 1, this principle is used in line 2 to dictate pursuing diplopia first.) Thus, for such local strategic principles we can write:

$$S_L(\text{attributes}(o)) =_{\text{def}} >_{o[a] \in \{a \mid a \text{ is an attribute of object type } o\}}.$$

For example, $S_L(F)=\{F_{[\text{abnormal}]} > F_{[\text{hard}]} > F_{[\text{soft}]} > F_{[\text{non-specific}]}\}$.

In summary, given a focus node, *f*, some principles in S_L, termed *object-centered principles* and denoted S_{Lo}, determine which subgoal $g \in G(\text{object}(f))$ to pursue next, while other principles in S_L, termed *attribute-centered principles* and denoted S_{La}, act as "tie breakers" when $G(\text{object}(f))$ involves several subgoals of the same type. More generally, principles in S_{Lo} can be contingent on attributes of *f* and vice versa. For example, if *f* is an observed (hard) finding, then $\{?Fg \rightarrow F > ?D \rightarrow F\}$; if *f* is an abnormal finding, then $\{?D \rightarrow F > ?Fg \rightarrow F\}$; etc.

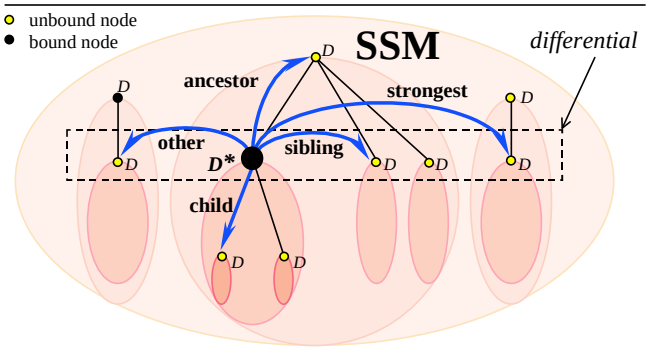
Having seen how local strategic principles impose order on subgoals, what is then the effect of applying S_L in each cycle? Given a focus node *f* and the set of subgoals associated with *f*, applying S_L returns the highest ranking subgoal, identifying it as the subgoal to pursue next. When S_L imposes only a partial order over subgoals, ACE-SSM must choose arbitrarily the next subgoal to pursue. For example, if $S_{Lo}(D)=\{D \rightarrow ?F > ?Dg \rightarrow D \geq D \rightarrow ?Ds > ?A \rightarrow D\}$, it can pursue $?Dg \rightarrow D$ before $D \rightarrow ?Ds$ or vice versa. Generally, the less order S_L imposes on subgoals, the less guided is ACE-SSM's search. In the extreme case where S_L imposes no order on subgoals, ACE-SSM could simply expand depth-first or breadth-first the SSM subgraph chosen by global strategic principles.

Speaking of global strategic principles, S_G, these play a different role. As an SSM evolves, it could embody several subgraphs that depict different candidate solutions. Assuming that the solver focuses on one candidate solution at a time, S_G tells which candidate solution should it be. Understanding exactly how S_G works requires understanding the topology of SSMs relative to the application task. For our sample task, the roots of all SSM subgraphs (and their subgraphs) are D nodes that are organized in the SSM following the topology of the portion of the disease taxonomy in *K* that was thus far instantiated. Of these, the most specific D nodes form the so-called diagnostic differential. Respectively, Figure 5 shows the possible relationships between the roots of competing SSM subgraphs. S_G takes affect conditional on specific changes to the SSM, and especially to the differential, that can occur after each time the SSM grows a new edge. More specifically, if the root of the currently pursued subgraph is termed the *solution focus* and denoted D^*, the following tells how global principles choose the solution focus:

$$S_G(\text{SSM}) = \begin{cases} \textit{strongest } D \in \text{differential} & (s_1)\ \text{if strength (CF) of the current } D^* \text{ dropped below that of another } D \in \text{differential, or} \\ & (s_2)\ \text{if a new } D \text{ was added to the differential that is not subsummed by any SSM subgraph} \\ \textit{child of } D^* & (s_3)\ \text{if the current } D^* \text{ was just specialized after doing } D \rightarrow ?Ds \\ \textit{sibling } D^* & (s_4)\ \text{if the current } D^* \text{ was pursued - tested by doing } D \rightarrow ?F \text{ and specialized by doing } D \rightarrow ?Ds \\ \textit{other } D \in \text{differential} & (s_5)\ \text{if the current } D^* \text{ and all its siblings were pursued} \\ \textit{anscestor of } D^* & (s_6)\ \text{if every } D \in \text{differential was pursued} \end{cases}$$

Each of these global principles pertains to a different change to the SSM. For example, principle (s_1) corresponds to a change in the strength of D^* (where strength can also measure, e.g., the number of abnormal Fs covered by the SSM subgraph D^* spans [3]), and principle (s_2) corresponds to the case where

a newly added *D* node is from a disease category not yet considered. These principles effect search in two general ways. Some narrow down the search within the diagnostic context defined by the currently pursued SSM subgraph, e.g., principle (s_3) keeps the solver focused on the last disease category considered. Other principles broaden the search to unexplored SSM subgraphs, e.g., principle (s_6) switches the solver to categorical reasoning involving a broader diagnostic context. In Figure 1, lines 13-14 and 23 in the SSM construction trace illustrate the effect of principles (s_3) and (s_4), respectively.



Where *D** is the current solution focus, or root of the currently pursued SSM subgraph, global strategic principles can divert attention to another SSM subgraph whose root is the:

- "strongest" candidate solution in the differential
- child of *D**
- sibling of *D**
- some other candidate in the differential
- an ancestor of candidates in the differential

Figure 5: relations between candidate solutions as the basis for global strategic principles

Global principles can be expressed using predicate calculus. For example, where solution-focus(*D*) identifies its argument as the current solution focus, the above principles labeled (s_1), (s_4) and (s_5) are written as:

(s_1') $\forall D$ solution-focus(D) $\wedge$ ($\exists D_1$ in-differential(D_1) $\wedge$ strongest-belief(D_1) $\wedge \neq(D, D_1)$) $\Rightarrow$ solution-focus(D_1),

(s_4') $\forall D$ solution-focus(D) $\wedge$ ($\exists D_1$ child(D_1, D) $\wedge$ $\neg$pursued(D_1)) $\Rightarrow$ solution-focus(D_1),

(s_5') $\forall D$ solution-focus(D) $\wedge$ pursued(D) $\wedge$ ($\exists D_1$ sibling(D_1, D) $\wedge$ $\neg$pursued(D_1)) $\Rightarrow$ solution-focus(D_1).

In these principles, several support functions are themselves formulae. For example:

$\forall D$ belief(D)$\geq$0.2 $\wedge$ ($\neg\exists D_1$ child(D_1, D)) $\Rightarrow$ in-differential(D),

$\forall D \exists F$ *cause*(D,F) $\wedge$ is-bound(F) $\wedge$ ($\exists D_1$ *subtype*(D,D_1) $\wedge$ is-bound(D_1)) $\Rightarrow$ pursued(D).

In the first example, a disease hypothesis is excluded from the differential, or ruled-out, by not having CF$\geq$0.2 or by not being the most specific *D*. In the second example, the pursued function checks if there was an attempt to both test and refine a disease hypothesis, and the is-bound function checks whether its argument is a bound SSM node.

When the antecedents of principles in S_G are not mutually exclusive, the order by which these principles are applied would necessarily compile control knowledge. For example, principle (s_1') will always fire before principle (s_5') above, unless the sibling of *D** is also the strongest candidate in the differential. To avoid this problem, these principles can be appropriately "forced" to be mutually

exclusive; e.g., ($a_1 \Rightarrow c_1$) and ($a_2 \Rightarrow c_2$) would be converted into ($a_1 \wedge \neg a_2 \Rightarrow c_1$) and ($a_2 \wedge \neg a_1 \Rightarrow c_2$), respectively.

S-operators are the representational constructs used to capture strategic principles on the symbol-level. Principles in S_L are captured as ordered sets, and are applied by a simple general-purpose S-operator. Given a pointer to a focus node *f* in the SSM, this S-operator returns one subgoal associated with *f* as the subgoal chosen to be pursued next. As to principles in S_G, these are captured using S-operators that are similar to G-operators, except that they reposition a pointer to a focus SSM subgraph instead of return an unsatisfied node-chain. For example, S-operators for principles (s_4') and (s_5') are written in pseudo-code as:

```
(S_OP::switch_to_child($p)        % $p = pointer to root of the current solution focus
    if   children($p,"D") <> NIL                  % if the root has D node children
    then return($p = first(children($p,"D"))))    % reposition pointer to the first child


(S_OP::switch_to_sibling($p)      % $p = pointer to root of the current solution focus
    if pursued($p)                                % if the root was pursued
        S = siblings($p)                          % S = siblings of the root
        do until S = NIL
            if not pursued($p1 = pop(S))          % reposition pointer to first sibling not
            then return($p1)                      %    yet pursued
    end-if
    return("fail"))                               % return: "inapplicable S-operator"
```

In these examples, **siblings($p)** and **children($p)** are general-purpose functions that are part of ACE-SSM. These functions treat SSMs purely as directed graphs, and hence apply graph-oriented operations to generate their results. They know nothing about the meaning of SSMs relative to the target application task.

4 COMPUTATIONAL TRAITS OF ACE-SSM

Having seen how ACE-SSM works, we next explain why ACE-SSM is both task- and domain-independent. We also compare ACE-SSM to common goal-directed reasoning architectures in terms of its computational requirements. While this comparison is rather informal, it shows that ACE-SSM is not necessarily more computationally intensive. Recall that, given the rather recent nature of ACE-SSM's approach to goal-directed reasoning, this paper's primary objective is to layout this approach, illustrate its plausibility, and point out unique potential benefits it offers.

4.1 Task- and Domain-Independence

ACE-SSM would work for any application task, as long as it is provided with what we call a microtheory of the task. We define a *microtheory* to be a solution to an application task, recognizing that the task can have several microtheories that reflect different viewpoints, levels of granularity, etc. A *knowledge-level microtheory* is the triplet ⟨*G,K,S*⟩ (where *K* is not yet populated). A *symbol-level microtheory* is the set of instantiated representational constructs capturing ⟨*G,K,S*⟩ -- K-operators, G-operators, S-operators, and relational networks. As Figure 6 shows, these constructs do not directly update the SSM. Instead, in response to their triggering, G-operators analyze the SSM and each return an unsatisfied node-chain (subgraph), S-operators analyze the SSM and return a pointer to an SSM subgraph and to an unsatisfied node-chain in that subgraph, and K-operators return each a satisfied node-chain. Only update operators that are part of ACE-SSM use these returned results to update the SSM. Since under the relational networks representation all SSMs are directed graphs, these update operators involve only general-purpose graph operations (add a node, link nodes, append disconnected subgraphs, etc.). They are task- and domain-independent in that they do not need to know the meaning of node-chains and SSM subgraphs on which they operate. This means that ACE-SSM would work for any task whose microtheory is committed to the relational network epistemology.

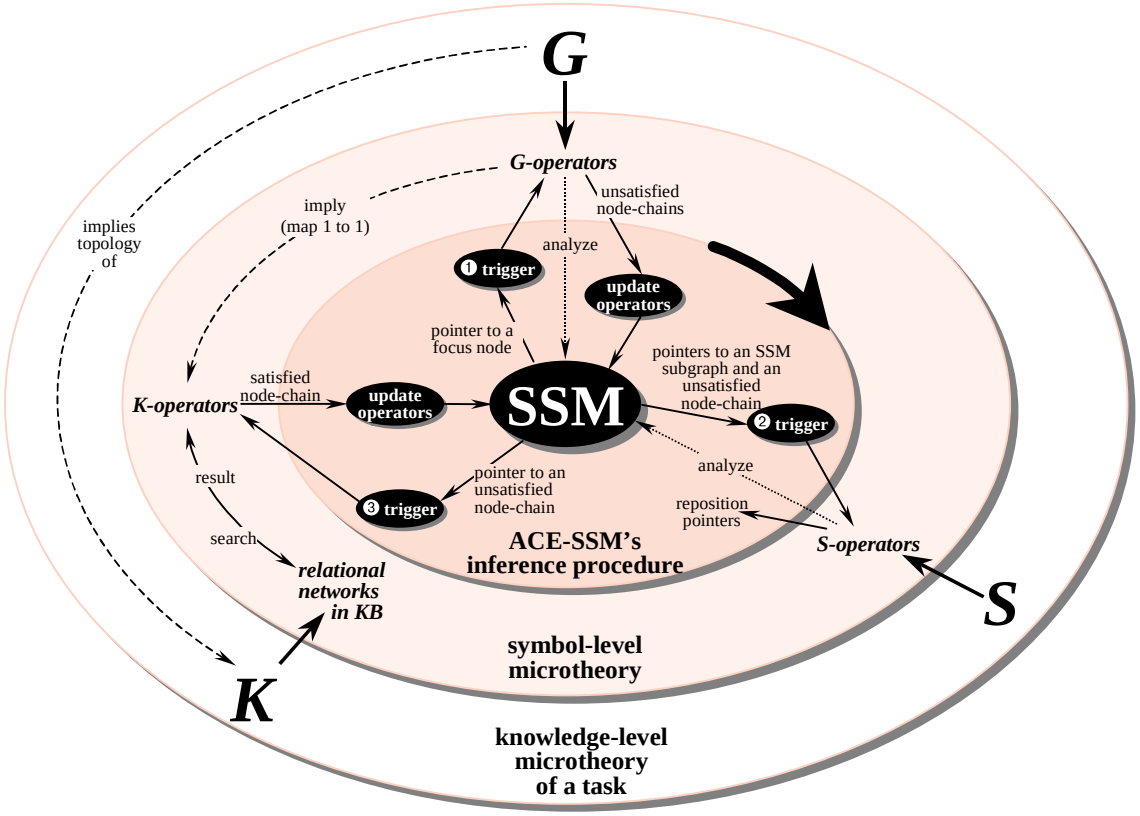


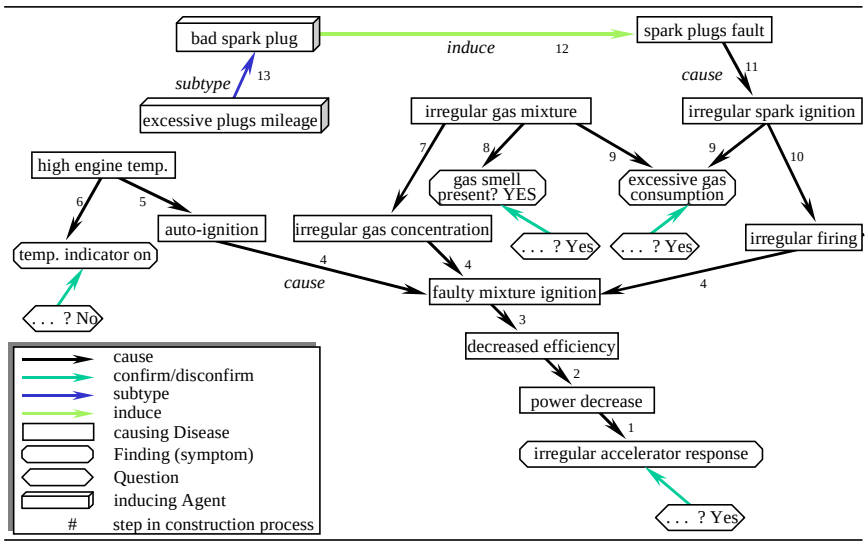
Figure 6: a deeper look into ACE-SSM's workings

To see our point more clearly, consider the following example. Figure 7A presents a sample SSM for the automobile diagnosis task solved in a KBS called CHECK [8]. The structural similarity of SSMs constructed for CHECK's task and our sample task is not coincidental. As Figure 7B indicates, except for some differences, both tasks have similar goal knowledge because they both model malfunctions (diseases, disorders) as abnormal processes. The impact of the differences can be explained by looking at the goal constraints governing the topology of SSMs, *G*, and the vocabulary used to express these constraints, *V*. The constraints in *G* can be divided into:

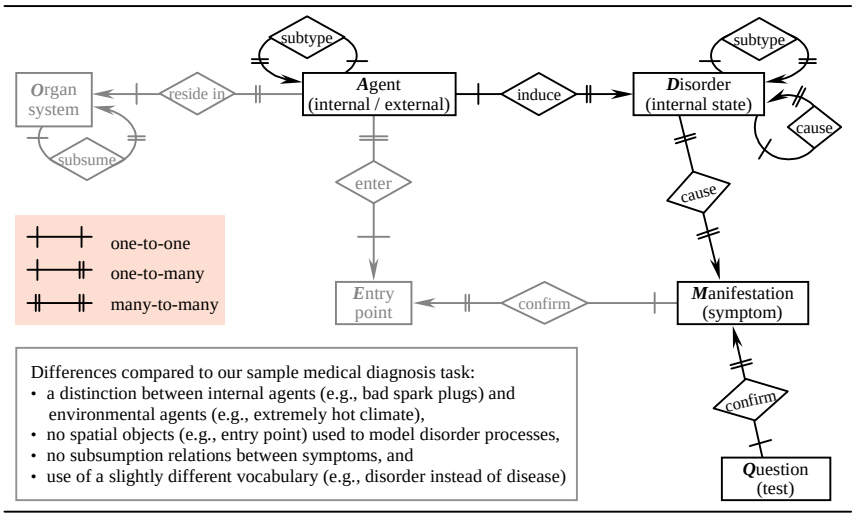
- G_o -- constraints pertaining directly to the task ontology, independent of the application domain (e.g., an abnormal manifestation must be linked to a causing malfunction); and
- G_d -- constraints involving domain-specific object attributes pertaining to the application domain (e.g., in our sample task, a finding can be *hard* (i.e., observed, measured) or *soft* (i.e., circumstantial evidence)).

Similarly, we can break down *V* into:

- V_o -- ontology-specific terms used to model task-specific phenomena (e.g., malfunction, manifestation);
- V_d -- domain-specific terms used as synonyms and attributes of terms in V_o (e.g., disease, finding); and
- V_e -- epistemology-specific terms pertaining to the graph representation of SSMs (e.g., root, sibling).



(A) sample SSM (adapted from [8])



(B) underlying structure of SSMs

Figure 7: SSM constructed for CHECK's automobile diagnosis task and its underlying goal model

If not for G_d and V_d, the G for our sample task would also apply to CHECK's task, and SSMs for both tasks would be expressed using terms only in V_o. Put differently, we could use terms only in V_e and V_o to express the symbol-level constructs (G-operators, K-operators, etc.) capturing $\langle G,K,S \rangle$, without impairing their functionality. Of course, to account for differences on G due to the use of G_d and V_d, we need to adapt our G_o and provide ACE-SSM with G_d and the corresponding symbol-level constructs, expressed also using terms in V_d. Nevertheless, because all of ACE-SSM's symbol-level constructs operate on node-chains (subgraphs), they can do their job without having to know the meaning of terms in V_o and V_d. In other words, regardless of the domain, upon adapting the G for our sample task to another task with the same ontology, ACE-SSM would be able to solve that task as well. This means that, under ACE-SSM, a microtheory can be generic in some sense to an entire class of tasks involving the same ontology.

4.2 Computational Requirements

Domain-independence aside, it is well known that making more knowledge explicit generally means more computation. To see the extent to which this assertion applies to ACE-SSM, which makes goal knowledge *G* explicit, we need to assess ACE-SSM's computational requirements relative to those of the approaches reviewed in Section 2.1. However, we emphasize that computational efficiency is only one measure of performance. Given that ACE-SSM and the other approaches vary on their motivation, computational issues must be weighed against the effort required from programmers as well as the benefits that each approach seeks to derive using goal-directedness. We defer discussion of those benefits to the next section, and focus here only on the programming and computational requirements. The latter are assessed in terms of the effort needed to deliberate about goal posting, deliberate about goal sequencing, and manage a working memory.

The effort needed to deliberate about which goals to post in each problem situation depends on the assumed goal structure. Except for GRAPES, the RBS schemes reviewed earlier require no such deliberation because they assume a static and pre-existing goal structure. They assume that much of the deliberation was done by the programmer during elicitation and coding of the goal structure into object-level rules. Coding goals into rules permits making the goal structure parsimonious in posting only the goals relevant to each problem situation; however, it hardwires domain-specific *S* into *K*, and leads to maintenance difficulties [6]. GRAPES allows deliberation about goal posting to be done using meta-rules. Unlike RBS schemes, the PSM/GT-based approach and ACE-SSM must deliberate about which goals to post in each problem situation, because they assume a meta-level goal structure captured as a hierarchy of procedural subtasks and as goal constraints, respectively. The PSM/GT-based approach deliberates by applying procedurally coded conditional statements, whereas ACE-SSM deliberates by applying goal constraints to the SSM. Compared to RBS schemes, both approaches offer two programming advantages. First, they keep *S* separate from *K* and thus simplify maintenance. Second, constructing a meta-level goal structure is easier than constructing a comprehensive goal structure in the style of RBSs'. As to SOAR, it falls between these two extremes. It assumes a static goal structure built into object-level rules. But, by using its decision procedure to dynamically evaluate preferences assigned to posted actions, it implicitly deliberates about when additional goals have to be posted to resolve impasses SOAR runs into (e.g., when the available *S* cannot decide which action to take next).

Deliberation about goal sequencing hinges on the control mechanism used and the *S* it employs. In basic RBSs, because domain-specific *S* is implicit, such deliberation is done using general-purpose conflict resolution mechanisms. In GRAPES, rules capturing domain-specific *S* explicitly deliberate about goal sequencing by analyzing the goal memory. GDD and AMORD delegate this deliberation effort to a planning facility, which uses little or no domain-specific *S*. SOAR too uses no conflict resolution mechanisms. Instead, domain-specific *S* in the form of preferences assigned to posted actions is used by its decision procedure to deliberated about goal sequencing. When it runs into an impasse, SOAR sets up a subgoal that leads it to try out all competing actions. (SOAR's chunking mechanism is supposed to learn over time how to avoid such inefficient deliberation.) In the PSM/GT-based approach, deliberation about goal sequencing is done both by procedural *S* captured in a hierarchy of subtasks, which can inspect dedicated data structures (e.g., a list of disease hypotheses in the diagnostic differential), and through the management of a task stack (e.g., track "end-conditions" of tasks to know when to fold-back completed tasks upwards in the stack) [7]. In principle, ACE-SSM is as efficient as the PSM/GT-based approach, because it can use the same domain-specific *S*, although it captures *S* declaratively. For instance, ACE-SSM captures the principle "test a disease hypothesis before refining it" as the order relation $\{D \rightarrow ?F > D \rightarrow ?Ds\}$, whereas NEOMYCIN hardwires it into a procedural subtask named Pursue-Hypothesis that always activates in the same order two lower-level subtasks named Test-Hypothesis and Refine-Hypothesis [7].

Of the compared approaches, only ACE-SSM captures *S* declaratively. This leads to three programming advantages. First, by being able to express S using predicate logic, ACE-SSM allows using a host of available techniques to formalize *S*, validate it, resue it, etc. Second, ACE-SSM allows binding of

control to occur at run-time, unlike in standard procedural languages where the binding occurs at the time the system is created. Both features mean that ACE-SSM requires less effort to code and update *S*. A third advantage relates to the elicitation of *S*. As Clancey [7] notes, given the structure of the logical argument a task entails modeling using SSMs, *S* can be rationally grounded in various cognitive and computational constraints. For example, depending on the width and depth of a disease taxonomy in *K*, *S* can be designed to induce a depth-first or breadth-first search of the taxonomy, to expedite problem solving. Thus, *S* must not be purely based on experts' reasoning -- it can also be normative in some sense. When *S* is declarative, it is easier to experiment with different control schemes.

Finally, let us look at the effort needed to manage a working memory. In RBS schemes, including AMORD and GDD, updating the working memory is the least costly. In GRAPES, goal-directedness requires managing a dedicated goal memory in addition. SOAR as well requires more effort because it manages a context stack – record working memory elements in a specific context, add contexts, remove subcontext defined relative to an achieved supergoal, etc. In the PSM/GT-based approach, similar effort is needed to manage dedicated data structures (e.g., differential list) and the task stack. In ACE-SSM, the SSM constitutes the working memory. Compared to RBSs, ACE-SSM requires only the extra effort needed to record the order by which elements are added to the SSM and to make their relationships explicit.

We see that, from the point of view of computational efficiency, ACE-SSM is not necessarily worse off compared to other approaches in which goal-directedness is explicit. Even if ACE-SSM involves somewhat more overhead cost, recall that computational efficiency is only one measure of performance. The next section presents other important benefits potentially offered by ACE-SSM's approach.

5 ADVANCES FACILITATED BY GOAL KNOWLEDGE

After seeing the way goal knowledge *G* is used to construct SSMs, this section examines the potential benefits from its availability and that of explicit SSMs. These benefits have to do with ways to simplify the development and maintenance of KBSs, and ways to enhance the robustness and the explanation capabilities of KBSs. We must point out that, given the recent nature of the line of research presented in this paper, discussion of the potential benefits must be regarded at this point as merely pointing out new avenues for future research. This discussion is intended to only provide the core ideas behind the merit and plausibility of these benefits. Based on these core ideas, we are currently developing some of the automated tools and capabilities needed to materialize these benefits.

5.1 Exploiting Microtheories in KBS Construction and Maintenance

To simplify the construction of applications for which knowledge is captured as relational networks, we can develop a KBS shell that stores generic microtheories of various task classes. The shell is an object-oriented (OO) hierarchy with nodes organized in layers corresponding to increasingly specialized task ontologies (see the top right corner of Figure 8). These could include, for example, the *abnormal chronological process* ontology used in diagnosis applications that utilize staged-failure models showing the time-ordered steps leading to a malfunction, as in CASTER [28], and the *abnormal interactive-historic process* ontology we discussed so far in the context of NEOMYCIN and CHECK. Nodes in the OO hierarchy hence store (or inherit parts of) specific knowledge-level (KL) microtheories and their symbol-level (SL) counterparts, each expressed using an ontology-specific vocabulary, V_o. Specifically, each node stores: a set of ontology-specific goal constraints, G_o, and corresponding instantiated G-operators and K-operators; templates for relational networks in *K* that follow from constraints in G_o, e.g., a constraint embedding the phrase *cause*(*D*, *F*) implies the need for a causal network relating diseases to findings; and strategic principles, *S*, and corresponding instantiated S-operators.

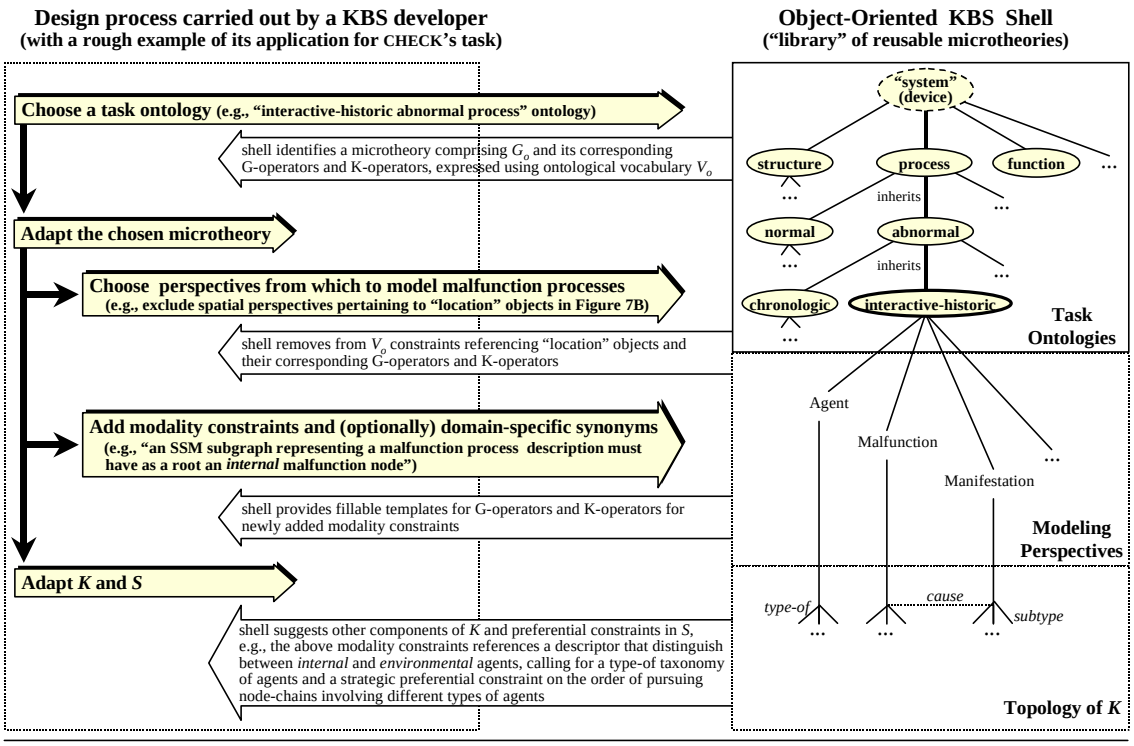


Figure 8: top-down KBS design through reuse of microtheories

Unlike common KBS shells which provide only representational constructs, or automated knowledge-acquisition shells that provide only one fix microtheory (e.g., SALT [17]), our proposed OO shell would provide generic microtheories of various task classes that can be adapted into specific applications. This shell is in the spirit of previous work [19] which proposed developing frameworks for handling task classes as top-level abstraction hierarchies and reusing these frameworks to create applications. Knowledge engineers could capitalize on the availability of reusable microtheories by adapting them following a rather structured top-down design process. Figure 8 outlines this process as well as illustrates its applications to CHECK's diagnosis task.

The logic of this structured design process helps to see how certain aspects of maintenance can also be simplified, when maintenance involves revisions of the *G* underlying a KBS. Like in knowledge modeling approaches such as COMMONKADS [25], we can exploit the link between the KL and the SL microtheories of a task, and require that a revision of the KBS design start with a revision of its underlying KL microtheory. Since in ACE-SSM there is a structure-preserving mapping between a KL microtheory and its SL counterpart, our proposed shell could detect automatically how changes in the KL microtheory impact specific SL constituents. More precisely, given a revised set of goal constraints, denoted *G'*, the shell can use differences between *G'* and the original *G* to recommend: what G-operators and K-operators to add, delete, or modify; what relational networks in *K* to add, remove, or restructure; and, which strategic principles in *S* to add, delete, or revise.

5.2 Lowering Brittleness and Increasing Novelty

Construction and maintenance aside, a common concern is how to enhance operational properties of KBSs. One important property is robustness -- the ability to perform well outside a narrow range of expertise. Simmons and Davis [26] show that a typical way to increase robustness is to improve domain coverage by providing a KBS with multiple knowledge sources, or *K*'s, reflecting different viewpoints (e.g.,

structural and functional models), levels of granularity (e.g., quantitative and qualitative models), etc. This endeavor involves several difficulties to be discussed shortly. These difficulties have to do mainly with how to dynamically integrate different *K*'s. Our approach can largely avoid these difficulties, by allowing to dynamically integrate *K*'s through their associated microtheories.

We can demonstrate this idea in the context of work by Liu and Farely [16] who discuss situations which require using *K*'s that are associated with different microtheories (see Figure 9). For example, one such problem can be: *given a circuit with a fix structure, how to lower the current on one of its resistors without changing the resistance of, or voltage across, that resistor?* Suppose that one of the available microtheories characterizes a circuit/component at the macro level using terms like voltage, current, and resistance. Its associated *K* comprises macro behavioral axioms like Ohm's and Kirchoff's laws. Since based on Ohm's law we cannot change current without changing resistance or voltage, solving the problem requires questioning the micro behavioral axioms of a resistor outside this microtheory. Therefore, suppose that we have available another microtheory that describes components at the micro level, in terms of their cross-sectional area, the velocity of electric charge-carriers, the field they create, etc. Here, the associated *K* comprises micro behavioral axioms like $\partial I = \partial C = \partial A + \partial v$, where ∂ denotes a qualitative change, *I* is current, *C* is charge flow, *A* is a component's cross-sectional area, and *v* is charge-carriers' velocity. To solve the problem, we can combine these two microtheories. This is possible because the intersection on the *G* (goal constrains) and *V* (vocabulary) from both microtheories is not empty; it includes, e.g., the "perturbation propagation" goal constraint: $\forall x \, \forall y \, perturbed(x) \wedge connected(x, y) \Rightarrow perturb(x, y)$, where *x* and *y* are parameters. As shown in Figure 9, the composite microtheory allows to derive a solution saying that we need to increase the physical length of the resistor in question.

We can explain based on this example how making explicit the *G*'s associated with diverse *K*'s helps to avoid two key difficulties commonly faced in the context of increasing robustness. One difficulty has to do with identifying which *K*'s can be integrated and how. This issue is typically addressed by using static task-specific "switching modules" to prescribe which and how specific *K*'s can be integrated [14][16]. In our approach, there is no need for such switching modules; given several *K*'s and their associated *G*'s, we can dynamically analyze the intersections of *G*'s (or actually of the task ontologies they reflect) to identify which *K*'s can be integrated and along what dimensions (e.g., common parameters). The other usual difficulty is how to integrate results and reasoning across *K*'s implemented using diverse representations. This difficulty is often addressed by using a dedicated logic-based intermediary representational language to propagate information and translate between representations (see [26] for details). Our approach avoids this added complexity and effort. First, propagation of information across integrated *K*'s is done through the combined SSMs constructed by the microtheories associated with these *K*'s (see Figure 9). Second, translation across representations is not necessary because ACE-SSM allows hiding in K-operators all the representational details of integrated *K*'s. Recalling that any *K* is applied only with the K-operators defined by its associated *G*, neither ACE-SSM nor the K-operators associated with other *K*'s being integrated need to know anything about the symbol-level representation of that specific *K*.

5.3 Exploiting SSMs for Explanation Purposes

Explanation is another useful operational property of KBSs. Availability of *G* and explicit SSMs helps to enhance various explanations associated with the three conceptual "planes" in ACE-SSM (see Figure 2). We next review this subject while focusing in particular on the more challenging explanations associated with the control strategy.

Explanations pertaining to *K* relate to *justifying* elements in *K*, based on a finer granularity *K'*. For example, a KBS can be asked to justify the assertion "the current through a resistor increases when voltage across it increases," derived based on Ohm's law. To explain this assertion, *G* permits the dynamic shifting across lumped-device models (*K*) and charge-carrier models (*K'*), in a task-independent manner, as we saw in the earlier example in Section 5.2.

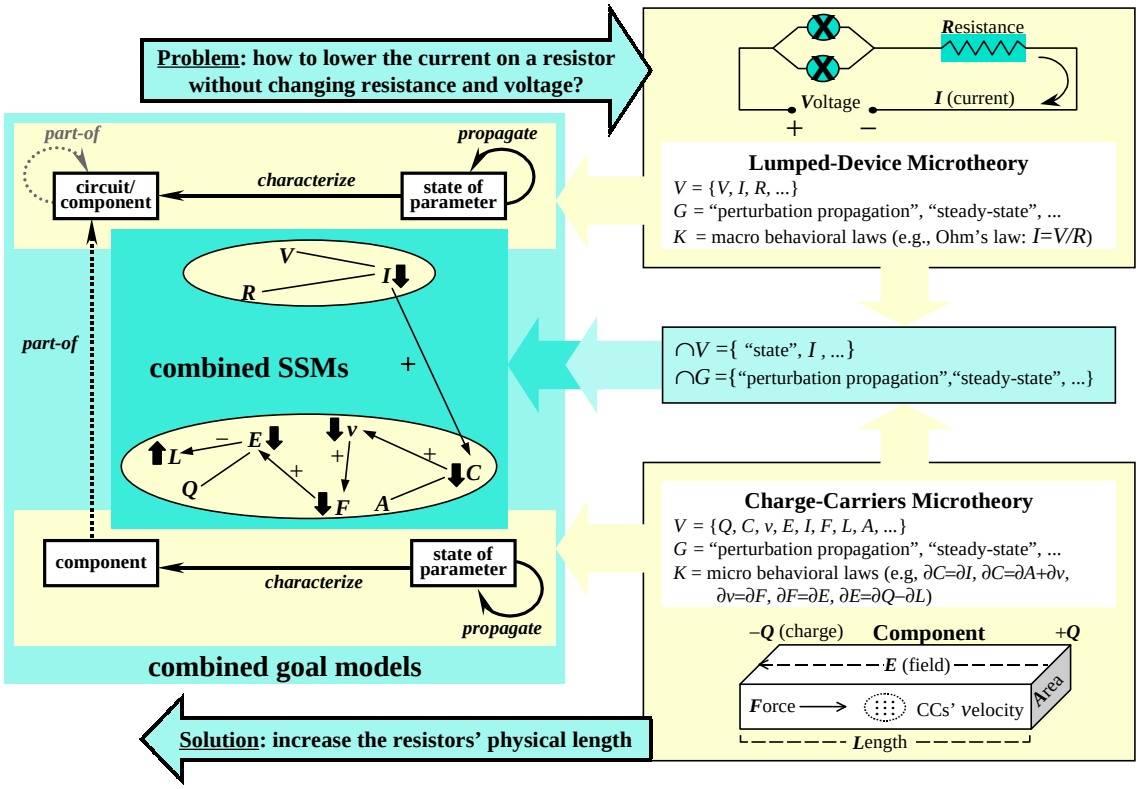


Figure 9: integrating microtheories dynamically

Relative to goal knowledge *G*, explanations seek to show *why* the solution produced by a system is good. Here, since the solution is captured in an SSM, we can show how the SSM meets the informational requirements of the task, defined explicitly by goal constraints in *G*. For example, in our sample diagnosis task, since *G* defines a causal script pertaining to a disease process, explaining a solution boils down to using the SSM to show that an instance of the causal script that implicates the diagnosed disease has occurred.

Explanations associated with the control strategy pertain to system behavior -- *how* was the solution derived (i.e., using what abstract lines of reasoning), and *why* was it derived that way. Generating *how* explanations of behavior requires interpreting the symbol-level reasoning process by mapping it back to knowledge-level terms. Granted that an SSM reflecting the order by which all its node-chains were added constitutes an explicit record of the system's behavior for a particular case, *how* explanations of behavior could be produced by collapsing node-chain sequences and mapping them to abstract lines of reasoning. In the example shown in Figure 10, two lines of reasoning -- *top-down refinement* followed by *categorical reasoning* -- are identified by knowing only that the ontological objects *Ds* and *Dg* syntactically denote a specialization and a generalization of *D*, respectively, and that *F* denotes a "feature" of *D*. Many such node-chain sequence patterns are probably common to other applications involving relational networks, and therefore it should be possible to identify and interpret them using task- and domain-independent explanation routines.

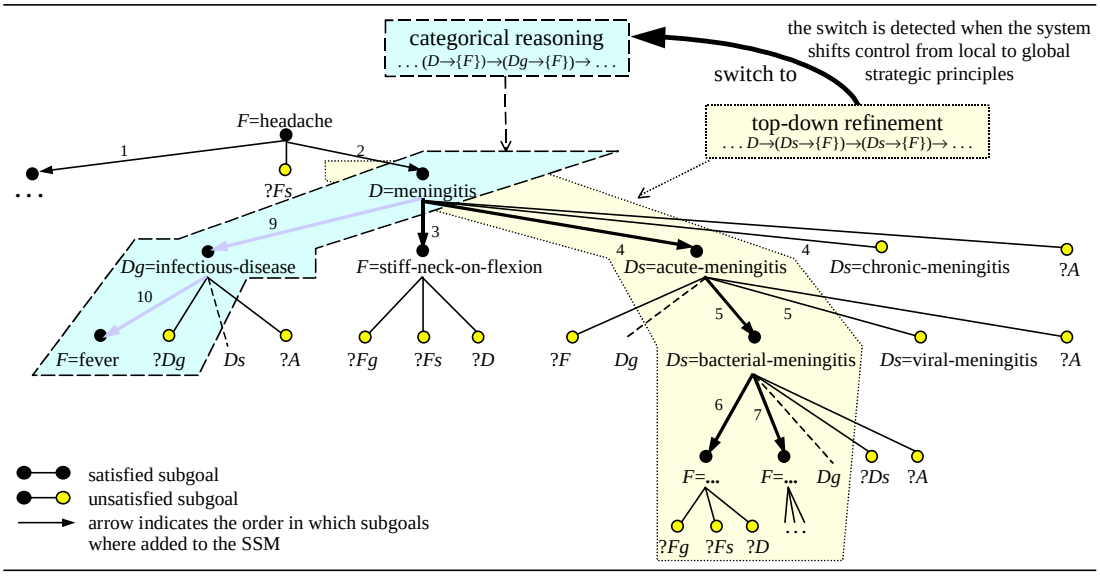


Figure 10: collapsing node-chain sequences to produce *how* explanation of behavior

Several existing KBSs that do not create explicit SSMs take a different approach to producing *how* explanations of behavior. Some produce such explanations by reporting on the order by which subtasks were applied based on the control strategy they embed. For example, DIVA [9] treats the stack of active subtasks as a record of behavior, and it produces *how* explanations by collapsing sequences of subtasks in the stack into abstract lines of reasoning; however, DIVA uses for this purpose special task-dependent explanation routines. Another example is REX [29], which takes the final conclusion and all intermediary conclusions produced during problem solving, and justifies the solution by assembling a plausible story that could deviate from what took place during problem solving. For example, in diagnosis, the story can reflect the general logical structure of diagnostic tasks, saying: (1) several diagnostic hypotheses that can explain the principle symptoms were identified, (2) some of these hypotheses were ruled out because they are implausible regardless of the symptoms they explain, and (3) the diagnostic conclusion is the best of the plausible hypotheses that can explain all the symptoms. Clearly, aside from not producing explanations that reflect the actual lines of reasoning the system followed, REX requires using task-specific explanation routines like in DIVA.

As to *why* explanations of behavior, which no known KBS is yet capable of producing, we propose using the following scheme. Assume the existence of a meta-system that creates its own SSM for modeling what the object system is doing during problem solving (see Figure 11). In the meta-SSM, each node corresponds to a node-chains sequence (line of reasoning) that has been started and may or may not be complete, and each link corresponds to a triggered global strategic principle in *S* that is the rational for the object system switching across two lines of reasoning. Such a meta-SSM would permit producing *why* explanation of behavior that provide the rationale for pursuing, interrupting, retracting, or resuming alternative lines of reasoning. Moreover, it would permit deliberating about which interrupted line of reasoning to resume, from the exact point where it was abandoned. This contrasts with systems like NEOMYCIN and DIVA, where end-conditions of active subtasks on the stack can interrupt ongoing lines of reasoning, and decisions concerning the state of the inference process are ad-hoc because the status of these lines of reasoning is nowhere recorded.

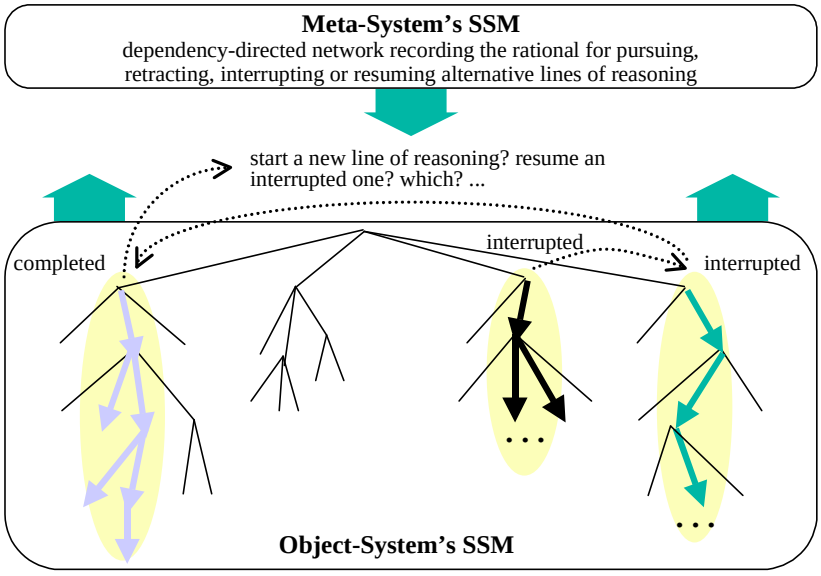


Figure 11: meta-system's SSM for generating *why* explanations of behavior

In fact, a meta-SSM could also help to attune the inference process. Intuitively put, it would allow analyzing how the object-system's SSM evolves, to determine how strategic principles in *S* can be adjusted so as to focus the search, avoid dead-ends, manage resource allocation, etc. For example, if a subgraph in the object-system's SSM coincides with a "wondering" top-down refinement line of reasoning, the order relations (over subgoals) defined by local strategic principles could be modified dynamically. Similarly, if frequent jumping from one SSM subgraph to another occurs because many lines of reasoning are interrupted, the global strategic principle(s) causing this behavior could be dynamically modified or deactivated. Capabilities of this kind are the subject of ongoing research.

6 CONCLUSION AND FUTURE RESEARCH

Whereas past research has focused on making explicit both domain knowledge, *K*, and strategic control knowledge, *S*, our work focuses on also making goal knowledge, *G*, explicit. We showed that *G* captures the ontology of SSMs that a particular task entails constructing. We explained how ACE-SSM, our task- and domain-independent architecture, uses *G* to direct the construction of explicit SSMs. Finally, we discussed several potential benefits that can be derived from the availability of *G* and SSMs in explicit form. These benefits have to do with promoting reuse of task ontologies in the development of KBS applications, simplifying these systems' maintenance, permitting them to possess more robust problem-solving capabilities, and enabling them to produce strong explanations. These benefits do not come at the cost of giving up any features present in current KBSs. In particular, not only do we keep *S* separate from *K*, like in known KBSs, we also anchor all control decisions in the need to evolve an SSM into a state with characteristics specified by *G*. This allows expressing *S* in a declarative form. Moreover, this allows ACE-SSM to view the process of problem solving as modeling, and reflect more of the real nature of problem-solving from a knowledge-level perspective. We showed that such a reflection could be beneficial for reuse, explanation, and robustness purposes.

Like with any new approach, much additional research is needed, mainly in four areas. One area pertains to the type of tasks for which ACE-SSM can be applied. We indicated in Sections 3.2.1 and 4.1 that ACE-SSM would work for every task that is understood sufficiently well, that is, every task for which it is possible to reveal the ontology of SSMs it entails constructing. So far ACE-SSM was applied to medical and mechanical diagnosis tasks, and it is now being applied to design tasks in the domains of financial risk management and elevator configuration. It is necessary to apply ACE-SSM to other types of

tasks so as to gain more insight into the strengths and limitations of its approach.

Second, being that ACE-SSM is currently a plain implementation of the conceptual SSM construction process described in Section 2.2, it should be possible to lower the overhead cost involved in applying goal constraints in *G* and strategic principles in *S*. One way to do so is to develop efficient graph-based algorithms for ACE-SSM's inference procedure. Another possibility is to implement ACE-SSM's approach using general-purpose solvers. For example, one can think of a production system that reasons with three different KBs: an object-level KB capturing *K*, a meta-level KB capturing goal constraints in *G*, and a control KB capturing declarative strategic principles in *S*. This architecture would allow taking advantage of existing algorithms (e.g., RETE [23]) that make RBSs efficient solvers.

Third, a fundamental issue is how to go about revealing the goal knowledge underlying a new KBS application. SSM-DKM [1] is a methodology that was recently developed for this purpose; however, it needs to be empirically tested.

The last area pertains to the potential benefits derived from availability of *G* and explicit SSMs. We showed that these benefits are quite significant, but more research work is needed to materialize them. We discussed and illustrated the core ideas behind what needs to be developed in this regard. In addition, we can identify several specific avenues where future work can benefit the line of research discussed in this paper. One avenue relates to the need to develop a typology of task ontologies that will allow reusing *G* following the scheme we presented in Section 5.1. A good starting point might be the extensive literature on role-limiting problem-solving methods (e.g., *propose-and-revise* [7]). Since all PSMs construct (implicit) SSMs through interactions of their comprising subtasks, it would be fruitful to analyze the inputs and outputs of these subtasks. Doing so for a PSM will not only reveal much of the ontology underlying the class of tasks for which the PSM was developed, but also serve as a first step toward expressing the *S* it captures using the declarative formalism we use in ACE-SSM. Other avenues for future research relate to robustness and explanation. Formal schemes and techniques are needed to enable an architecture like ACE-SSM to identify dynamically which task microtheories available to it can be combined and how, as a way to facilitate solving "new" problems of which no single microtheory is capable. Similarly, for explanation purposes, it is necessary to develop explanation routines that can identify within SSMs node-chain sequences that could be mapped to well-established abstract lines of reasoning (e.g., top-down refinement, linear search). Here too, it might be helpful to analyze known PSMs in terms of the order by which their subtasks are triggered to take specific node-chains as input and return other ones as output.

REFERENCES

[1] Benaroch, M., "Knowledge Modeling Driven by Situation-Specific Models," under review in *International Journal of Human-Computer Interaction*, 1997.

[2] Benjamins, V.R., "On the role of problem solving methods in knowledge acquisition -- experiments with diagnostic strategies," in Steels L., Schreiber G., Van de Velde W. (Eds.), *A Future for Knowledge Acquisition*, Springer-Verlag, London, 1994, pp. 137-157.

[3] Benjamins, V.R., and Jansweijer, W., "Toward a Competence Theory of Diagnosis," *IEEE Expert*, 9(5), pp. 43-52, October, 1994.

[4] Chandrasekaran, B., "Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design," *IEEE Expert*, 1(3), pp. 23-30, 1986.

[5] Chandrasekaran, B., and Jonhson T.R., "Generic Tasks and Task Structures: History, Critique and New Directions," in *Second Generation Expert Systems*, David J.M., Krivine J.P., and Simmons R. (Eds.), Springer-Verlag, London, pp. 233-272, 1993.

[6] Clancey, W.J., "Model Construction Operators," *Artificial Intelligence*, 53, pp. 1-115, 1992.

[7] Clancey, W.J., "Acquiring, Representing and Evaluating a Competence Model of Diagnostic Strategy," in *The Nature of Expertise*, Chi M., Glaser R., and Farr M.J. (Eds.), Lawrence Erlbaum Associated, Publishers, Hillsdale, NJ, pp. 343-418, 1988.

[8] Console, L., Fossa, M., and Torasso, P., "Acquisition of Causal Knowledge in the CHECK System," *Computers and Artificial Intelligence*, 8(4), pp. 323-345, 1989.

[9] David, J.M., and Krivine, J.P., "Explaining Reasoning from Knowledge Level Models," European Conference on Artificial Intelligence (ECAI-90), 1990.

[10] Davis, R., and Lenat, D., *Knowledge-Based Systems in Artificial Intelligence*, McGraw-Hill, 1982.

[11] deKleer, J., Doyle, J., Steele, G.L., and Sussman, G.J., "AMORD: A Deductive Procedure System," MIT AI Memo No. 435, 1978.

[12] Ginsberg, M.L., "Multivalued Logics: A Uniform Approach to Inference in Artificial Intelligence," *Computational Intelligence*, 4, pp. 265-316, 1988.

[13] Gruber, T.R., "A Translation Approach to Portable Ontology Specifications," *Knowledge Acquisition*, 5, pp. 199-220, 1993.

[14] Hunt, J.E., and Price, C.J., "Multiple-Model Diagnosis of Electro-Mechanical Subsystems," *Systems Engineering Journal*, 2(2), pp. 74-89, 1992.

[15] Josephson, J., Smetters, D., Fox, R., Oblinger, D., Welch, A., and Northurp, G., Integrated Generic Task Toolset – Fafner Release 1, Technical Report, Laboratory of AI Research, The Ohio State University, Columbus, OH, 1898.

[16] Liu, Z., and Farely, A., "Shifting Ontological Perspectives in Reasoning about Physical Systems," *Proceedings of AAAI-1991*, AAAI Press, pp. 395-400, 1991.

[17] Marcus, S., and McDermott, J., "SALT: A Knowledge Acquisition Tool for Propose-and-Revise Systems," *Artificial Intelligence*, 39(1), pp. 1-38, 1989.

[18] McDermott, J., "Preliminary Steps Toward a Taxonomy of Problem-Solving Methods," in *Automating Knowledge Acquisition for Expert Systems*, Marcus S. (Ed.), Kluwer Academic Publishers, pp. 225-256, 1988.

[19] Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., and Swartout, R.W., "Enabling Technology for Knowledge Sharing," *AI Magazine*, Fall, 1991.

[20] Newell, A., *Unified Theories of Cognition*, Harvard Press, Cambridge, MA, 1990.

[21] Puerta, A.R., Egar, J.W., Tu, S.W., and Musen, M.A., "A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools," *Knowledge Acquisition*, 4, pp. 171-196, 1992.

[22] Rymon, R., "Goal-Directed Diagnosis -- A Diagnostic Reasoning Framework for Exploratory-Corrective Domains," *Artificial Intelligence*, 84, pp. 257-297, July, 1996.

[23] Sauers, R., "Controlling Expert Systems," in *Expert System Applications*, Bloc L. and Coombs M.J. (Eds.), Springer-Verlag, Berlin, pp. 79-197, 1988.

[24] Sauers, R., and Farrell, R., *GRAPES User's Manual*, Technical Report ONR-82-3, Dept. of Psychology, Carnegie-Mellon University, 1982.

[25] Schreiber, G., Wielinga, B., and de Hoog, R., "CommonKADS: A Comprehensive Methodology for KBS Development," *IEEE Expert*, December, pp. 28-36, 1994.

[26] Simmons, R., and Davis, R., "The Roles of Knowledge and Representation in Problem Solving," in *Second Generation Expert Systems*, David J.M., Krivine J.P., and Simmons R. (Eds.), Springer-Verlag, London, pp. 27-45, 1993.

[27] Stefik, M., *Introduction to Knowledge Systems*, Morgan Kaufmann Publishers, San Francisco, CA, 1995.

[28] Thompson, T., and Clancey, W.J., "A Qualitative Modeling Shell for Process Diagnosis," *IEEE Software*, 3(2), pp. 6-15, 1985.

[29] Wick, M.R., "Reconstructive Expert System Explanation," *Artificial Intelligence*, 54, pp. 33-70, 1992.

ACKNOWLEDGEMENTS

The author thanks the anonymous referees for their careful reading of the manuscript and their constructive comments.

BIOGRAPHY

Michel Benaroch received the B.Sc. degree in Mathematics and Computer Science in 1983, the MBA degree in Information Systems in 1985, both from the Hebrew University of Jerusalem, and the Ph.D. degree in Information Systems in 1992 from the Stern School of Business at New York University. He is currently an associate professor in the School of Management at Syracuse University, New York. His research interests focus on the function of ontological knowledge in knowledge-based systems, the use of qualitative reasoning in economics, and the application of option pricing techniques to the evaluation of investments in information technology. His publications have appeared (and are forthcoming) in *Decision Support Systems*, *Decision Sciences*, *IEEE Transactions on Knowledge and Data Engineering*, *International Journal of Human-Computer Interaction*, *Information Systems Research*, *Journal of Economic Dynamics and Control*, and *Information Sciences*, among others. Benaroch is a member of the ACM, the IEEE, and the AIS.