Soar: A Comparison with Rule-based Systems

This paper compares and contrasts Soar with rule-based systems. In the current implementation of Soar, knowledge is represented, at the lowest level, in the form of production rules. This is largely due to the representational simplicity and modularity of productions, as well as the existence of efficient production-matching algorithms. However, the mechanisms that Soar uses in conjunction with these rules lead to systems quite distinct from typical rule-based systems. Researchers who work with Soar tend to call it a rule-based system, even acknowledging that the term has confusing connotations. Here, we take the opportunity to make explicit some of the important distinctions between Soar and "typical" rule-based systems. The following introduces the mechanisms of the latest version of Soar and compares them to the processes used in RBS.

- **Parallel, associative memory**

  In Soar, *all* knowledge relevant to the current problem is identified and brought to bear via an associative mechanism. Any relevant knowledge is activated, leading to further elaboration of agent belief, proposals to perform different tasks, taking action in the world, etc. Typical rule-based systems choose among matching rules rather than activating all of them. Conflict resolution is the process of choosing between alternative rules. Conflict resolution usually depends on syntactic features of rules; for example, when deciding between two matching rules, the rule-based system might choose the rule instantiated with the most recent memory elements or the largest number of them. Soar needs no conflict resolution for rule selection. All relevant knowledge is brought to bear in parallel.

- **Belief maintenance**

  Soar employs computationally inexpensive truth maintenance algorithms to update beliefs about the world. Belief maintenance ensures that agents are responsive to their environments. In typical rule-based systems, every item that is added to memory via a rule must be maintained via other rules. This means that every change to an agent's context must be the result of a deliberate commitment. This commitment is one source of the perception that rule-based systems are generally brittle and inflexible, because they over-commit to particular courses of action. Using automated belief maintenance, Soar does not require deliberation to maintain the consistency of its internal beliefs with the outside environment. All non-deliberate beliefs require no knowledge maintenance or additional commitment.

- **Preference-based deliberation**

  Deliberation in Soar is mediated by preferences, which allow agents to bring selection knowledge to bear for decisions. Preferences provide a mechanism for knowledge-mediated choice: the agent can express knowledge about which options it prefers in the current situation. Soar never chooses between rules to make a decision. If selection knowledge conflicts, Soar establishes a goal to resolve the conflict. Soar never reverts to a fixed procedure for choosing rules to apply. Instead, all decisions are determined by available knowledge.

  Successful deliberation in Soar results in the selection of an *operator*. An operator in artificial intelligence indicates a procedure with a precondition (what must be true for the operator to be activated) and action (what the operator does). In RBS, individual rules are the operators. In Soar, an operator is an abstract type, implemented as a collection of rules. The Soar decision procedure sorts the preferences for operators to determine what operator should be selected. Once an operator is selected, different rules specify the action. Because the pre-condition and action components of the operator are implemented as separate rules, Soar agents can easily recognize available options and reason about which option to take. This separation can also lead to *fewer* total rules because a single precondition rule can be matched with any number of action rules (and vice versa). In contrast, when pre-condition and action are combined in the same rule, as in RBS, the agent needs rules for every possible combination of pre-condition and action, leading to a potential combinatorial explosion in rules.

- **Automatic subgoaling**

  Soar recognizes conflicts (*impasses*) in selection knowledge and automatically creates a subgoal to resolve the impasse. A subgoal is a new state, and rule knowledge can be applied in both the superstates and substates. This contrasts with RBS, where there is typically only a single state. Further, automatic subgoaling gives Soar agents a meta-level reasoning capability, the ability to reason about their own reasoning.

- **Decomposition via problem spaces**

  Automatic subgoaling enables task decomposition. At each step in the decomposition, the agent is able to focus its knowledge on the particular options at just that level and ignore considerations at other levels. The process of decomposition narrows a potentially exponential number of considerations into a much smaller set of choices. Recognizing knowledge appropriate to the current problem is the essence of the problem space hypothesis. Automatic subgoaling leads to a hierarchy of distinct states; thus, agents can use multiple problem spaces simultaneously, and knowledge can be created that is specific to the unique states. This makes knowledge search simpler (leading to faster rule matching). Moreover, the knowledge base is naturally compartmentalized, providing a scalable infrastructure with which to build very large knowledge bases.

- **Adaptation via generalization of experience:**

  Once an agent comes to a decision that resolves an impasse, it summarizes and generalizes the reasoning during the impasse. This process results in new knowledge that will allow the agent to avoid an impasse when encountering a similar situation. Soar's learning mechanism is fully integrated (it is built into the architecture), pervasive (it automatically applies to all reasoning) and flexible (it can be used to realize a variety of different kinds of adaptation). Because the learning algorithm is an integral part of the overall system, Soar also provides a structure that addresses when learning occurs (when impasses are resolved), what is learned (a summarization of impasse processing), and why learning occurs (fast, rational reaction to the environment). The drawback of Soar's learning mechanism is all higher-level learning styles must be realized within the constraints imposed by Soar's basic learning mechanism. Thus, while Soar provides a lot of constraint towards integrating multiple learning methods with behavior, realizing any individual learning style is often more straightforward in a typical RBS.

To summarize, although Soar uses rules as its lowest-level representation language, the processes Soar uses in conjunction with the rules differ significantly from other rule-based systems. Soar also adds architectural support for operators and problems spaces; higher level, abstract representations not directly supported in other rule-based systems. The result is that Soar systems scale much better (in both performance and the manageability of the knowledge base) and require fewer rules for sufficiently complex applications relative to typical rule-based systems. Given the differences, together with the common preconceptions associated with rule-based systems, it might be appropriate simply to say that Soar is *not* a rule-based system. The alternative argument (historically made by Soar researchers) is that rule-based representations can be used in much more flexible ways than they typically have been.