# Resolving Contentions Between Initial and Learned Knowledge

Robert E. Wray, III and Randolph M. Jones
Soar Technology
317 North First Street
Ann Arbor, MI 48103
(734)327-8000
wrayre@acm.org, rjones@soartech.com

**Abstract** *This work describes a problem,* knowledge contention*, that arises when agent knowledge learned via knowledge compilation contends with the agent's original task decomposition knowledge in the execution of agent actions. We show the circumstances under which the problem occurs, present a solution that avoids knowledge contention, and provide empirical results that show that knowledge compilation under the proposed solution leads to agents that can integrate planning, plan execution, and learning without explicit knowledge design for learning.*

*Keywords:* integration of planning, learning and execution, knowledge compilation, EBL

## 1 Background and Motivation

Our interest is in integrated planning and execution systems that must interact with dynamic and quickly changing environments. For the purposes of this research, such systems may integrate "traditional" search-based planning with execution, or they may be purely reactive planners/executors (Agre and Chapman, 1987). In these systems, the time that it takes to plan can impact the system's performance and its ability to learn. The aim of this paper is two-fold: 1) detail some of the advantages of hierarchical plan representations for execution and learning, and 2) show how parallelism in the reasoning architecture leads to temporal contention between a system's initial knowledge and knowledge acquired via compilation. We propose a straightforward solution to this "knowledge contention." An empirical evaluation illustrates both the efficacy of the solution and the benefits of using knowledge compilation to improve performance in planning and execution systems that use hierarchical decompositions.

## 2 Compiling Hierarchically Decomposed Knowledge

One approach to exploiting the advantages of both hierarchical and flat representations is to encode agent knowledge initially in a hierarchical fashion, and then use knowledge compilation learning to allow the system to develop more efficient representations itself. Compilation caches the results of a hierarchical decomposition. If the agent compiles its plan and execution knowledge for a particular situation, the compiled knowledge will apply in a similar situation as soon as the knowledge can be retrieved, thus obviating the delay that occurs due to decomposition. Compilation thus provides experience-directed composition of complex behavior from simple subtasks.

Explanation-based learning (EBL) approaches have used goal regression techniques to operationalize problem solving knowledge in a number of domain classes. EBL uses a *domain theory* to generate an *explanation* of why some *training instance* is an example of a *goal concept* according to some *operationality criterion* (DeJong and Mooney, 1986). For an interactive agent, the goal concepts for

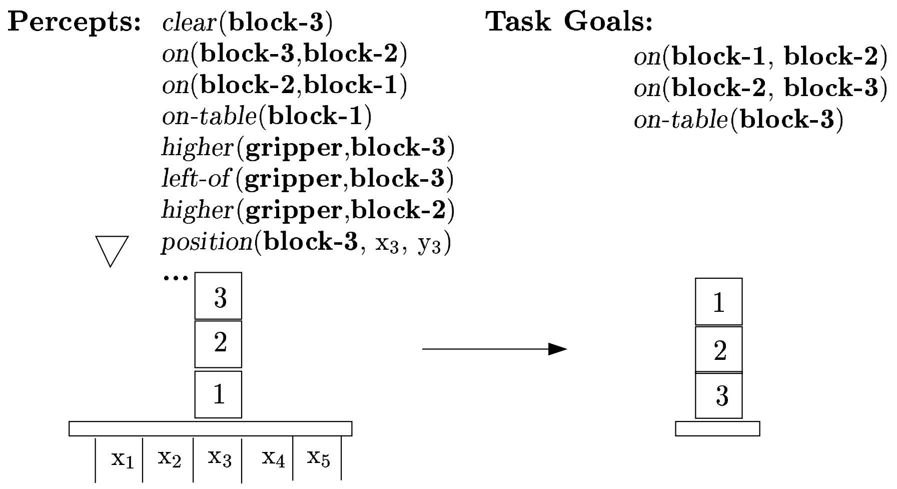
**Percepts:** *clear*(**block-3**)
*on*(**block-3**,**block-2**)
*on*(**block-2**,**block-1**)
*on-table*(**block-1**)
*higher*(**gripper**,**block-3**)
*left-of*(**gripper**,**block-3**)
*higher*(**gripper**,**block-2**)
*position*(**block-3**, $x_3$, $y_3$)
...
**Task Goals:**
*on*(**block-1**, **block-2**)
*on*(**block-2**, **block-3**)
*on-table*(**block-3**)
3
2
1
$x_1$
$x_2$
$x_3$
$x_4$
$x_5$
1
2
3

Figure 1: Building a tower in the Blocks World.

EBL are the situations in which a primitive operation should be generated in the current external state, *where the current external state is defined by the available percepts and task goals*. This definition of external state serves as the operationality criterion for execution domains: a condition is operational if it is a direct input from perception or a task goal. Explanation occurs over the hierarchical knowledge about the task domain, the domain theory. The training example is a current state description. Therefore, one way to use EBL in interactive environments is to operationalize the generation of a primitive action, which will thus avoid the decomposition process in the future.

Knowledge compilation has been used successfully in static domains and in dynamic domains where the learning occurs "off line" from the execution. EBL has been applied in dynamic domains in only limited cases (e.g., (Bresina et al., 1993; Laird and Rosenbloom, 1990; Mitchell, 1990)). In a dynamic domain, the training instance may change over time, making knowledge compilation methods potentially difficult to use. In particular, previous systems have been dependent upon specific representation schemes to avoid problems resulting from compilation when the data base of assertions could become inconsistent. These problems include compiling knowledge that includes features that never co-occur (the non-contemporaneous constraints problem (Wray et al., 1996)) and conflicts between compiled and original task knowledge (the knowledge contention problem).

## 3 Contention Between Compiled and Hierarchical Knowledge

*Knowledge contention* arises when duplicate plans specify the same action in two different levels of the hierarchy. Although duplication would be avoided by a knowledge designer, compilation does lead to duplicate plans (at different levels of generalization) for generating actions.

Consider, for example, a robot in an interactive variation of the Blocks World (Figure 1).[1] Assume that the agent has compiled a rule for executing a `step-right` action, compiling over subtasks such as `put-block-on-table` and `pick-up-block`, as in this rule:

[1]We use the the familiar, easy-to-understand Blocks World for illustration. The methods and ideas that follow should be applicable to all agent domains. The empirical evaluation will describe how these methods apply in a significantly more complex domain.

```
Rule 1:
    IF   Task-Goal(Tower(x,y,z))
         Not(On-Table(x))
         Clear(x)
         Left-of(gripper, x)
         Higher(gripper, x)
         Ready(gripper)
    THEN Execute(step-right(gripper))
```

If the gripper is *ready* in the Figure 1 state, Rule 1 would fire and execute the `step-right` action, achieving the desired result: the command to move the gripper is generated without referencing the intermediate goals (and thus there is no delay while creating them).

What does the agent do if the gripper is not *ready*? It begins to create a hierarchical plan by decomposing the problem. A `put-on-table(3)` goal is created, followed by further decomposition to a `pick-up` goal. At this point, if we assume that the agent has no direct means of making the gripper ready, the agent can progress no further; the agent now waits to receive the *ready* signal from the external environment. When it is perceived, both the compiled knowledge (Rule 1) and the original decomposition knowledge are immediately applicable. Depending on the specific implementation of the agent's motor system, the agent may step once to the right, twice, or not at all.

This example illustrates the potential of contention between an agent's original task knowledge and its compiled knowledge. The agent now has two different knowledge sources that specify the same action in the same state and, as the example shows, in some cases the architecture may try to apply both of these actions for the same task goal. Solutions to this problem can also become complicated by the time it takes to perform hierarchical decomposition. For example, the agent should also deal correctly with situations where the gripper becomes ready in the middle of creating subgoals.

## 4 A Solution to Knowledge Contention

Knowledge contention arises from parallelism in the reasoning architecture. In complex, real-time tasks, parallelism is an advantage because an agent can pursue different threads of reasoning that can be executed independently of one another (for example, moving a gripper and opening a gripper at the same time). However, allowing unrestricted parallelism can lead to inconsistencies in an agent's asserted knowledge (Wray, 1998). Unrestricted parallelism also allows contending pieces of knowledge to execute simultaneously.

One way to maintain "good" parallelism while avoiding many of the problems is to serialize the reasoning between subtasks while maintaining parallelism within a subtask. We call this serialization Subtask-limited Reasoning (SLR). Ideally, serialization would occur only when knowledge contention could be detected, but dependency computations are expensive (Almasi and Gottlieb, 1989). SLR uses a heuristic notion of dependency, allowing reasoning to progress from the top of the plan/execution hierarchy to the lowest level. This imposed serialization determines which levels of knowledge get first priority (the heuristic being that knowledge closer to the base of the hierarchy is presumably more efficiently represented).

Determining which assertions are associated with particular levels of the hierarchy is, in the worst case, linear in the number of pending assertions (Wray, 1998). Thus, SLR avoids expensive dependency computations. However, SLR delays *all* assertions in lower levels of the hierarchy, regardless of whether they are dependent on the reasoning causing the delay. Thus, there is some potential unnecessary loss in parallelism.

SLR provides a partial solution to knowledge contention. In the example above, both compiled and decomposition knowledge match simultaneously. With SLR, the agent would always invoke the compiled knowledge before the original knowledge, because the compiled knowledge applies at a higher level. SLR thus provides conflict resolution between compiled and original task knowledge, always preferring the compiled knowledge because it necessarily matches higher in the hierarchy.

SLR alone is not a complete solution. It prevents the rules from firing simultaneously, but would not prevent the decomposition knowledge from firing after Rule 1. However, the agent's assertion of an output command provides new information, in the form of the issued command. It is possible to tag such commands so that compiled knowledge will include the tag when issuing the command. For example, assume the agent creates a *Command* tag whenever it begins execution of a primitive. The original decomposition knowledge could test that this tag was not asserted before executing the command by adding Condition 2 to the knowledge that issued the primitive:

```
Condition 2:
Not(Command(step-right(gripper)))
```

Absence of the tag can then be used as a precondition for issuing the command at all levels of the hierarchy. Further, compilation would capture this condition in learned rules. Thus, Condition 2 would now be an additional condition in Rule 1. If a learned rule is not fully operationalized (at the top level of the hierarchy), the presence of the tag will prevent that rule from firing. Thus, the tagging solution with SLR provides a complete solution for all knowledge contention between knowledge at different levels of hierarchy, whether the contending pieces of knowledge are part of the original task knowledge or learned.

In a sense, such tagging can be viewed as an explicit way to mark resources (external actuators) as being busy or free. Making resource use explicit in this way is probably a good idea anyway, from a software engineering standpoint. Although some knowledge representations make resource usage decisions implicit, the problem of knowledge contention during learning demands a solution that makes such marking explicit.

## 5 Serialized Integration of Planning, Execution, and Learning

We have run a variety of tests to verify that the proposed architecture results in agent systems that can compile their knowledge unproblematically. We have instantiated the proposal in a variation of the Soar architecture, because Soar integrates hierarchical execution and planning, uses compilation as its learning mechanism, and allows unrestricted parallelism in the application of knowledge. Agents implemented using the original Soar architecture (Laird et al., 1987; Newell, 1990) require extensive knowledge design/re-design when the architecture's compilation mechanism is used in interactive environments (especially real-time environments). The additional knowledge engineering cost arises from knowledge contention as well as temporal inconsistencies in compiled knowledge (Wray et al., 1996). This section briefly relates the results of using compilation in the new architecture. We used previously existing agents, not specifically designed for learning, and the experiments demonstrate that the new architecture improves agent performance through compilation, and avoids knowledge contention without expensive knowledge re-design.

We first applied SLR to a Blocks World simulation, a testbed created to explore issues in architecture design. Agents in this simulation of the blocks world do no explicit planning, but rather have execution knowledge that allows the agent to manipulate any initial configuration of blocks into an ordered tower. Although this domain is simple, agents developed solely for the integrated planning and execution task, but not for learning, were not able to complete a single task in this domain when learning was enabled. The primary cause of failure was knowledge contention. The agents compiled a rule similar to the one shown previously, and

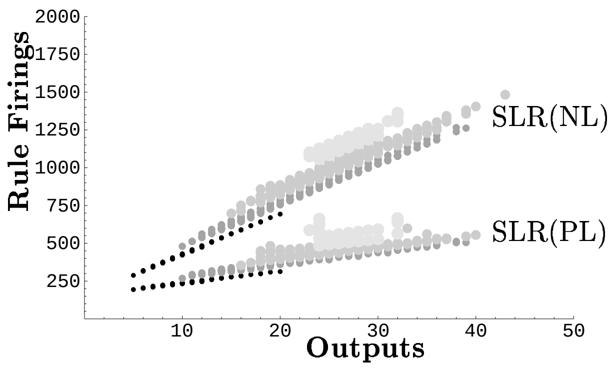

Figure 2: Output Commands *vs.* rule firings in the SLR agent for both post-learning (PL) and non-learning (NL) runs. The plot for the SLR learning agent does not include data from the first 100 cases (during which most compilation occurs). The shade and size of each datum signifies the the number of blocks moved for the run: (smallest, black) $\rightarrow$ 1 block; (larger, dark gray) $\rightarrow$ 2 blocks; (larger, medium gray) $\rightarrow$ 3 blocks; (largest, light gray) $\rightarrow$ 4 blocks.

the resulting knowledge contention debilitated the agent. Thus, the results do not include learning data without SLR because non-SLR agents could not perform the task due failures deriving from knowledge contention.[2]

Figure 2 summarizes the results of using SLR with the original agents. The agents were able to learn without difficulty and without significant knowledge re-design. The figure shows the relationships of production firings to output commands and number of blocks to move in the non-learning and learning agent. The figure demonstrates that learning reduces the knowledge (measured in production rules) that the agent must apply in order to perform its task. While compilation cannot reduce the number of outputs nor number of blocks that must be moved for any initial configuration of blocks, it does improve knowledge access performance by about 50%, which results in less overall time to execute the task. Thus, knowledge compilation achieves the goal of operationalizing knowledge to a more efficient form, while SLR eliminates the knowledge contention problems normally produced by real-time plan decomposition and parallelism.

It is not surprising that compilation could be successful in a simple, relatively static domain like the Blocks World, so we have also run tests in a much more complex domain. TacAir-Soar agents pilot virtual military aircraft in a real-time computer simulation of tactical combat (Jones et al., 1999; Tambe et al., 1995). The combat aircraft domain is only indirectly accessible (each agent uses realistic aircraft sensor models and can thus perceive only what a pilot in a real aircraft would sense), nondeterministic (the behavior of other agents cannot be strictly predicted or anticipated), nonepisodic (the decisions an agent makes early in the simulation can impact later options and capabilities), dynamic (the world changes in real time while the agent is reasoning), and continuous (individual inputs have continuous values). Domains with these characteristics are the most difficult ones in which to create and apply agents (Russell and Norvig, 1995).

Complete analysis of the compilation results from TacAir-Soar agents is beyond the space constraints of this paper.[3] In general, the TacAir-Soar agents required a modest amount of knowledge modification for the new architecture, although the changes were motivated by improvements in the task decomposition rather than learning (Wray and Laird, 1998). No further changes were necessary to overcome existing knowledge contention problems. Agents were able to compile their knowledge for efficient execution in this dynamic, complex domain.

The TacAir-Soar agents demonstrate that responsiveness can improve with compilation. One time-critical task for these agents is the launch of a missile; once a target aircraft is in range, the pilot should, as quickly as possible, push a fire button to launch a selected missile. Figure 3 illustrates the change in this average

[2]Significant knowledge re-design of the original agents could also allow unproblematic performance with learning.

[3]See (Wray, 1998) for further details.

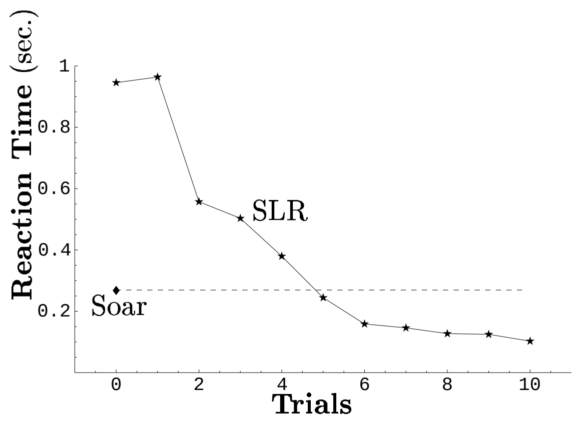

Figure 3: Improvement in reaction time with learning.

reaction time with learning. As in the Blocks World, the original Soar agents cannot use the learning capability of the architecture. Because TacAir-Soar uses a task decomposition that is several levels deep, the original agents would fire several missiles at once when compilation created contending pieces of knowledge for firing. Thus, the datum for the original agents is constant; they cannot improve their reaction time.

Initially, the agents under the new solution respond more slowly than their counterparts under the old architecture. Under the complete SLR solution, knowledge for different (sub)tasks must be expressed in individual units, which causes the initial decrease in reaction time. However, with additional learning trials, average reaction time decreases. By the fifth learning trial, reaction time in the new architecture has improved in comparison to the original. In the final learning trial, reaction time has decreased to about one-tenth of a second, a 61% improvement in comparison to the original agent.

The improvement in reaction time is attributable to two different factors. First, after compilation, the agent recognizes the conditions under which a missile should be launched without having to regenerate the plan hierarchy. Thus, the agent has gained more operational knowledge of when to fire a missile. Second, individual actions have been composed higher in the hierarchy and no longer require individual selection and application. Thus, multiple, independent actions can be initiated within a single agent reasoning cycle (an example of "good" parallelism). These results show that responsiveness can improve substantially with compilation, improving the quality of agent behavior. Again, these results were achieved without significant explicit knowledge design to handle learning. The SLR solution provides the infrastructure for non-problematic compilation.

## 6 Conclusion

When compiling over a knowledge hierarchy, knowledge contention can essentially create *race conditions* when one or more operationalized chunks of knowledge initiates an external action at the same time as the original domain knowledge. A race condition is simply a situation in which the result of some computation is dependent on the order in which the knowledge applies. Together with tagging of actions, SLR eliminates knowledge contention by assuming that assertions higher in the hierarchy have greater priority than local assertions. This is a reasonable assumption because the original plan/execution hierarchy is also built from top to bottom; an agent can be certain that a subtask is still valid in the current context only after the higher level context as been fully elaborated. Parallelism is still desirable and maintained for independent pieces of knowledge within each level of the hierarchy.

## References

Agre, P. E. and Chapman, D. (1987). Pengi: An implementation of a theory of activity. In *Proceedings of the National Conference on Artificial Intelligence*, pages 196–201, Seattle, Washington.

Almasi, G. S. and Gottlieb, A. (1989). *Highly Parallel Computing.* Benjamin/Cummings, Redwood City, California.

Bresina, J., Drummond, M., and Kedar, S. (1993). Reactive, integrated systems pose new problems for machine learning. In Minton, S., editor, *Machine Learning Methods for Planning*, chapter 6, pages 159–195. Morgan Kaufmann.

DeJong, G. and Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176.

Jones, R. M., Laird, J. E., Neilsen, P. E., Coulter, K. J., Kenny, P., and Koss, F. V. (1999). Automated intelligent pilots for combat flight simulation. *AI Magazine*, pages 27–41.

Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64.

Laird, J. E. and Rosenbloom, P. S. (1990). Integrating execution, planning, and learning in Soar for external environments. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1022–1029, Boston, Massachusetts.

Mitchell, T. M. (1990). Becoming increasingly reactive. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1051–1058, Boston, Massachusetts.

Newell, A. (1990). *Unified Theories of Cognition.* Harvard University Press.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach.* Prentice Hall.

Tambe, M., Johnson, W. L., Jones, R. M., Koss, F., Laird, J. E., Rosenbloom, P. S., and Schwamb, K. (1995). Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1):15–39.

Wray, R. E. (1998). *Ensuring Reasoning Consistency in Hierarchical Architectures.* PhD thesis, University of Michigan.

Wray, R. E. and Laird, J. (1998). Maintaining consistency in hierarchical reasoning. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 928–935, Madison, Wisconsin.

Wray, R. E., Laird, J., and Jones, R. M. (1996). Compilation of non-contemporaneous constraints. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 771–778, Portland, Oregon.