Resource-Bounded Reasoning for Complex Embedded Systems

Mark S. Boddy
Automated Reasoning Group
Honeywell Technology Center
3660 Technology Drive
Minneapolis, MN 55418
{boddy}@src.honeywell.com

1 Complex embedded systems

My current interest in resource-bounded reasoning primarily concerns applications to the control of complex embedded systems. Examples of such systems include manufacturing control systems, autonomous spacecraft, operations planning for an airline, or integrated avionics for commercial airliners. Most of the work to date on "applications" of resource-bounded reasoning has addressed abstract functions (e.g., heuristic search, probabilistic inference, vision processing). In some cases, these capabilities are then applied to stand-alone applications such as path planning, diagnosis, or user interface configuration. I do not mean by this description to denigrate either the generation of stand-alone applications or work on understanding how to implement specific forms of reasoning under resource bounds. These activities are essential.

However, there are some aspects of large, complex control systems which impose special requirements which will not be addressed by work on stand-alone applications. First and foremost is the fact that in such a system, the function you propose to add will have to be integrated with existing functions, which were designed in the absence of that capability. Worse, it may well be the case that the problem you propose to solve is sufficiently important that it is already being addressed, which imposes additional constraints based on the fact that in replacing the existing function, you must conform to interfaces designed for a different, and presumably less capable, implementation.

For example, consider the implementation of an automated system for tank management at a refinery. Tank management is already being done, most likely manually, perhaps with the aid of an Excel spreadsheet. Decisions about transfers from tank to tank interact with production planning decisions, made using large LP codes at several levels of granularity (years, quarters, or months), as well as with day-to-day operations (ships arriving either late or early), and on-line optimization of the refining process to address current weather conditions and the actual as opposed to estimated behavior of the system with a given input crude composition.

In an ideal world, it might be possible to construct a global optimizer that integrates tank management with production planning so as to maximize the profit the refinery will make by providing the most accurate possible model of capacity available over time, taking into account as well a first-principles model of refinery operation, and perhaps stochastic models for weather prediction and ship arrival times. In a slightly less unrealistic situation, perhaps we could implement a tank management system that could interact intelligently with production planning above, and refinery control below, adjusting these interactions as needed to cope with unforeseen events. This scenario is still overoptimistic: the reality is that current legacy code is fairly stupid. More to the point, subsystem interfaces are designed to reflect this fact.

The end result is that the insertion of new capabilities into complex systems is constrained to be an incremental process, in the sense that no more information or control is likely to be available than was initially assumed to be necessary. What's more, the "problem" to which you propose to provide a "solution" may very well have been engineered out of existence.

2 The need for resource-bounded reasoning for complex systems

Resource-bounded reasoning is already being done in complex embedded systems. As above: if a given capability is needed, it will be implemented in some form. Currently, solution quality is traded for time or other resources either through fixed resource bounds ("give me the best schedule you can by noon Monday"), or through simplifying the problem ("don't worry about maintenance, that's not important at this level"). These tradeoffs are almost invariably implicit, and buried either in a hunk of code somewhere, or in someone's head, depending on the level of automation involved.

These tradeoffs occur in many, many different places. Each one represents a possible opportunity to improve system operation by providing a more principled tradeoff decision, subject, of course, to the engineering difficulties outlined above. The next sections provides a brief set of examples of potential applications for resource-bounded reasoning as part of larger systems

3 Applications

3.1 Refineries

Refining operations have been automated from top-level production planning and purchasing decisions, down to minute-by-minute control of refining and product blending operations. At the level of refinery operations, there is a need for capabilities including:

- State estimation and diagnosis, using some combination of model-based and rule-based systems, sometimes both at the same time for different parts of the plant. This is complicated by the fact that sensors fail more often than any other part of the plant
- Recovery planning, given the current state estimation output.

- Predictive reasoning for evaluating planning decisions or control inputs (e.g., for refinery control or product blending).

3.2 Distributed Air Traffic Management

Distributed air traffic management involves making decisions on time scales ranging from minutes to hours, regarding what planes take off, where they will fly, and where and when they will land. The cost for making a tardy decision regarding some potential conflict is fairly high. The kinds of reasoning necessary here include:

- Conflict detection, including complex reasoning about 4-dimensional trajectories.
- Trajectory generation and optimization.
- Scheduling, for example ordering planes onto a runway.
- Communication and negotiation of commitments between adjacent air traffic control sectors.

3.3 Avionics and other integrated systems

Aircraft avionics are one example of a class of "integrated systems" in which the resource tradeoffs are more general. The people who design these integrated hardware/software systems recognize the interactions between hardware designs and software implementations, and as yet have no systematic approach to the problem.

This domain is somewhat less dynamic than the others listed above, in that frequently the tradeoffs being made are made at design time, not run time. However, it has a richer structure: the resource bounds in this case may involve (at the hardware level) any or all of power, weight, communication bandwidth, or cost. These constraints determine the limits on the system that can be built: is it more important to maximize the throughput of some set of general purpose processors, or to buy an expensive special-purpose chip? Given the current software function definitions, is the system memory-limited, CPU limited, or bus-limited?

Of course, the software design feeds back into this as well. It is hard enough to answer questions such as "given a fixed amount of memory, what is the best set of tradeoffs I can make among these various functions?" This problem is even worse: "given the space of possible tradeoffs, how much extra memory is it worth buying?"