Toward Practical Applications of Software Synthesis *

Douglas R. Smith
Cordell C. Green
Kestrel Institute
3260 Hillview Avenue
Palo Alto, California 94304
smith@kestrel.edu

Abstract

Formal methods are usually conceived as a way to obtain verifiably correct software, so many researchers have focused on applications requiring error-free code, such as safety-critical subsystems. There may be other paths to the ultimate success of formal methods. We argue that mechanized synthesis tools can have an impact in the production of high-performance algorithms. This thesis is supported by our work on the synthesis of transportation scheduling algorithms.

1 Introduction

We have been exploring automated tools for transforming formal specifications into efficient and correct programs. KIDS (Kestrel Interactive Development System) [8] serves as a testbed for our experiments and provides tools for performing deductive inference, algorithm design, expression simplification, finite differencing, partial evaluation, data type refinement, and other transformations. KIDS has been used to derive over 70 algorithms for a wide variety of application domains, including scheduling, combinatorial design, sorting and searching, computational geometry, pattern matching, and mathematical programming. The application of correctness-preserving transformations can allow programmers to synthesize complex codes embodying algorithms, data structures, and code-optimization techniques that might be too difficult to produce manually.

Formal methods need some dramatic success stories to spur the interest of industry and government (and even the U.S. academic community!). Our current strategy for selling the ultimate practicality of tools to support formal methods is to synthesize high-payoff algorithms, specifically scheduling algorithms. The intent is to show that automated algorithm

*This research was supported in part by ARPA/Rome Laboratories under Contract F30602-91-C-0043, in part by the Office of Naval Research under Grant N00014-93-C-0056.

design tools can economically generate families of high-performance scheduling codes. To this end we are producing a scheduling synthesis workstation that combines general-purpose synthesis tools with extensive knowledge about the scheduling domain and effective programming techniques for scheduling.

Tremendous benefits arise from having good scheduling systems. Many practical scheduling problems are NP-hard, so it is likely that there is no general and efficient solution method. The intrinsic combinatorial difficulty of scheduling practically requires heuristic algorithms for solving large-scale problems – optimal schedules can only be obtained for problems involving tens or hundreds of activities. However, the suboptimal schedules produced by many schedulers on large-scale practical problems mean that time, money, and resources are wasted. In practice, a good schedule found quickly is more valuable than an optimal schedule that may take days to compute. In any case, the schedule usually needs to be modified by users, who almost always have extra information that is not embodied in the scheduling model, and furthermore, during execution the schedule will need to be periodically revised to adapt to unanticipated circumstances.

As part of the ARPA/Rome Laboratory Planning and Scheduling Initiative, we have focused on the transformational development of transportation schedulers [12, 14]. Our approach involves several stages. The first step is to develop a formal model of the transportation scheduling domain, called a *domain theory*. Second, the constraints, objectives, and preferences of a particular scheduling problem are stated within a domain theory as a *problem specification*. Finally, an executable scheduler is synthesized semiautomatically by applying a sequence of *transformations* to the problem specification. The transformations embody programming knowledge about algorithms, data structures, program optimization techniques, etc. The result of the transformation process is executable code that is guaranteed to be consistent with the given problem specification. Furthermore, the resulting code can be extremely efficient.

In Sections 2 and 3 we describe the KIDS program synthesis system, and the programming knowledge that it uses to synthesize scheduling algorithms. In Section 4 we describe the application of KIDS to the synthesis of three different schedulers. The first, called KTS, is a strategic scheduler that runs several orders of magnitude faster than currently deployed (and manually written) schedulers. The second, called ITAS, is a laptop-based in-theater scheduler that has been delivered to PACAF at Hickham AFB in Hawaii and is regarded as being ready for operation during contingencies. The third is an ongoing joint effort to develop a scheduler for power plant outage activities.

2 KIDS model of program development

KIDS is a program transformation system – one applies a sequence of correctness-preserving transformations to an initial specification and achieves a correct and hopefully efficient program [8]. The system emphasizes the application of complex high-level transformations that perform significant and meaningful actions. From the user's point of view the system allows the user to make high-level design decisions like, "design a divide-and-conquer algorithm for that specification" or "simplify that expression in context". We hope that decisions at this level will be both intuitive to the user and be high-level enough that useful programs can be derived within a reasonable number of steps.

The user typically goes through the following steps in using KIDS for program development.

1. *Develop a domain theory* – An application domain is modeled by a domain theory (a collection of types, operations, laws, and inference rules). The domain theory specifies the concepts, operations, and relationships that characterize the application and supports reasoning about the domain via a deductive inference system. Our experience has been that distributive and monotonicity laws provide most of the laws that are needed to support design and optimization of code. KIDS has a theory development component that supports the automated derivation of various kinds of laws.

2. *Create a specification* – The user enters a problem specification stated in terms of the underlying domain theory.

3. *Apply a design tactic* – The user selects an algorithm design tactic from a menu and applies it to a specification. Currently KIDS has tactics for simple problem reduction (reducing a specification to a library routine), divide-and-conquer, global search (binary search, backtrack, branch-and-bound), problem reduction generators (dynamic programming, general branch-and-bound, and game-tree search algorithms), local search (hillclimbing algorithms), and others.

4. *Apply optimizations* – The KIDS system allows the application of optimization techniques such as expression simplification, partial evaluation, finite differencing, case analysis, and other transformations [8]. The user selects an optimization method from a menu and applies it by pointing at a program expression. Each of the optimization methods are fully automatic and, with the exception of simplification (which may be arbitrarily hard), take only a few seconds.

5. *Apply data type refinements* – The user can select implementations for the high-level data types in the program. Data type refinement rules carry out the details of constructing the implementation.

6. *Compile* – The resulting code is compiled to executable form. In a sense, KIDS can be regarded as a front-end to a conventional compiler.

Actually, the user is free to apply any subset of the KIDS operations in any order – the above sequence is typical of our experiments in algorithm design. A new system, called Specware, is currently under construction at Kestrel arising out of our theoretical investigations and experience with KIDS and other Kestrel systems. Specware is based on concepts of higher-order specifications, morphisms, and categorical constructions [16, 11, 13].

3 Synthesizing Schedulers

In this section we informally describe some of the algorithmic knowledge that is used to synthesize schedulers. Formal presentation of this knowledge and the deductive machinery necessary to apply it to concrete problem specifications may be found in [7, 14]. The general idea underlying the algorithm design tactics in KIDS is to represent abstract knowledge about a class of algorithms (such as divide-and-conquer or dynamic programming) as a theory, called an algorithm theory [9]. Models of an algorithm theory correspond to instances for a particular problem (e.g. quicksort and mergesort are two distinct models of divide-and-conquer theory). Constructing an algorithm for a formally specified problem corresponds to constructing an interpretation from an algorithm theory to the domain theory of the specification. Based on analysis of the algorithm design tactics in KIDS, we formalized several general techniques for constructing these interpretations in [11, 13]. These techniques are being implemented in SPECWARE.

There are two basic algorithmic approaches to computing a schedule: local and global. Local methods focus on individual schedules and similarity relationships between them. Once an initial schedule is obtained, it is iteratively improved by "moving" to similar neighboring schedules. Repair strategies [17, 5, 1], case-based reasoning, linear programming, and local search (hillclimbing) are examples of local methods.

Global methods focus on sets of schedules. A feasible or optimal schedule is found by repeatedly splitting an initial set of schedules into subsets until a feasible or optimal schedule can be easily extracted. Backtrack, constraint satisfaction, heuristic search, and branch-and-bound are all examples of global methods. We explore the application of global methods. In the following we briefly describe the notion of global search abstractly and show how it can be applied to synthesize a scheduler. Other projects taking a global approach include ISIS [2], OPIS [15], and MicroBoss [6] (all at CMU).

The basic idea of global search is to represent and manipulate sets of candidate solutions. The principal operations are to *extract* candidate solutions from a set and to *split* a set into subsets. Derived operations include *propagation* which is used to remove infeasible or nonoptimal solutions from sets. Global search algorithms work as follows: starting from an initial set that contains all solutions to the given problem instance, the algorithm repeatedly extracts solutions, splits sets, and reduces sets via propagation until no sets remain to be split. The process is often described as a tree (or DAG) search in which a node represents a set of candidates and an arc represents the split relationship between set and subset. Pruning is a special case of propagation in which all solutions from a set are eliminated, thereby allowing us to prune off that branch of the tree.

The sets of candidate solutions are often infinite and even when finite they are rarely represented extensionally. Thus global search algorithms are based on an abstract data type of intensional representations called *space descriptors*. In addition to the extraction and splitting operations mentioned above, the type also includes a predicate *satisfies* that determines when a candidate solution is in the set denoted by a descriptor. See [7] for a formal exposition of global search theory.

A simple global search theory of scheduling has the following form. Schedules are represented as maps from resources to sequences of trips, where each trip includes earliest-start-time, latest-start-time, port of embarkation, port of debarkation, and manifest (set of movement requirements). The type of schedules has the invariant (or subtype characteristic) that for each trip, the earliest-start-time is no later than the latest-start-time. A partial schedule is a schedule over a subset of the given movement records.

A set of schedules is represented by a partial schedule. The split operation extends the partial schedule in all possible ways. The initial set of schedules is described by the empty partial schedule – a map from each available resource to the empty sequence of trips. A partial schedule is extended by first selecting a movement record mvr to schedule, then selecting a resource r, and then a trip t on r (either an existing trip or a newly created one). Finally the extended schedule has mvr added to the manifest of trip t on resource r. The alternative ways that a partial schedule can be extended naturally gives rise to the branching structure underlying global search algorithms. The formal version of this global search theory of scheduling can be found in [10].

Propagation code is derived in several stages. First, necessary conditions on feasibility are derived. If these have the form of Horn-like constraints (a generalization of conditional inequalities), then they can be compiled into highly efficient, problem-specific constraint propagation code. More details may be found in [14].

4 Scheduling Applications

Transportation scheduling tools currently used by the U.S. government are based on models of the transportation domain that few people understand [4]. Consequently, users often do not trust that the scheduling results reflect their particular needs. Our approach tries to address this issue by making the domain model and scheduling problem explicit and clear. If a scheduling situation arises which is not treated by existing scheduling tools, the user can specify the problem and generate an situation-specific scheduler.

There are several advantages to a transformational approach to scheduling. First, there is no one scheduling problem – there are *families* of related problems in any given scheduling situation. The problems can differ in the mix of constraints to satisfy, cost objectives to minimize, and preferences to take into account. A typical problem is to schedule a given collection of activities on given resources. Another kind of problem is to find an estimate of the resources needed to bring about a desired completion date. Another kind of problem is to work backwards from a given completion date to feasible start dates for individual activities. Another kind of problem is incremental or reactive scheduling. We believe that transformation systems such as KIDS will provide the most economical means for generating such families of schedulers. We have observed a great deal of reuse of concepts and laws from the underlying domain theory and of the programming knowledge in the transformations.

A second advantage is the reuse of best-practice programming knowledge. The systematic development of global search algorithms has helped us exploit problem structure in ways that other projects sometimes overlook.

4.1 Strategic Transportation Scheduling

The U.S. Transportation Command and the component service commands use a relational database scheme called a TPFDD (Time-Phased Force and Deployment Data) for specifying the transportation requirements of an operation, such as the Somalia relief effort. We developed a domain theory of TPFDD scheduling defining the concepts of this problem and developed laws for reasoning about them. KIDS was used to derive and optimize a variety of global search scheduling algorithms that are generically called KTS (Kestrel Transportation

Data Sets (Air only)	# of input TPFDD records (ULNs)	# of individual movements	# of scheduled items after splitting	Solution time	Msec per scheduled item
CDART		296	330	0.5 sec	1.5
CSRT01	1600	1261	3557	44 sec	12
096-KS	20400	4644	6183	86 sec	14
9002T Borneo	28900	10623	15119	290 sec	20

Figure 1: Scheduling Statistics

Scheduler). The KTS schedulers are extremely fast and accurate (see below).

The surprising efficiency of KTS stems from two sources. First, the derived pruning and propagation tests are surprisingly strong. The stronger the test, the smaller the size of the runtime search tree. In fact, on many of the TPFDD problems we've tried so far, KTS finds a complete feasible schedule without backtracking. The pruning and propagation tests are derived as necessary conditions on feasibility, but for this problem they are so strong as to be virtually sufficient conditions. The second reason for KTS' efficiency is the specialized representation of the problem constraints and the development of specialized and highly optimized constraint operations. The result is that KTS explores the runtime search tree at a rate of several hundred thousand nodes per second, almost all of which are quickly eliminated.

The chart in Figure 1 lists 4 TPFDD problems, and for each problem (1) the number of TPFDD lines (each requirement line contains up to several hundred fields), (2) the number of individual movement requirements obtained from the TPFDD line (each line can specify several individual movements requirements), (3) the number of movement requirements obtained after splitting (some requirements are too large to fit on a single aircraft or ship so they must be split), (4) the cpu time to generate a complete schedule on a SUN Sparcstation 2, and (5) time spent per scheduled movement. Similar results were obtained for sea movements. The largest problem, Borneo NEO, is harder to solve, because of the presence of 29 movement requirements that are inherently unschedulable: their due date comes before their availability date. Such inconsistencies must be expected and handled by a realistic system. KTS simply relaxes the due date the minimal amount necessary to obtain a feasible schedule.

We compared the performance of KTS with several other TPFDD scheduling systems. We do not have direct access to JFAST and FLOGEN, but these are (or were) operational tools at AMC (Airlift Mobility Command, Scott AFB). According to [4] and David Brown (retired military planner consulting with the Planning Initiative), on a typical TPFDD of about 10,000 movement records, JFAST takes several hours (on a Unix workstation) and FLOGEN about 36 hours (on a mainframe). KTS on a TPFDD of this size will produce a detailed schedule in *one to three minutes.* So KTS seems to be a factor of about 25 times faster than JFAST and over 250 times faster than FLOGEN. The currently operational ADANS system reportedly runs at about the same speed as FLOGEN (running on a Korbex machine). When comparing schedulers it is also important to compare the transportation models that they support. KTS has a richer model than JFAST (i.e. handles more constraints and problem features), but less rich than ADANS. The ITAS effort described in the next section reflects our efforts to synthesize schedulers that have at least the richness of the ADANS model.

4.2 Theater Transportation Scheduling

The PACAF (Pacific Air Force) Airlift Operations Center at Hickam AFB, Honolulu is tasked with in-theater scheduling of a fleet of 26 C-130 aircraft (plus assorted strategic aircraft on loan) throughout the Pacific region. Current scheduling practice is essentially manual; for example, the relief effort for Hurricane Iniki which struck the island of Kauai in September 1992 was sketched out on 2 sheets of legal paper and required hours of labor. Since Spring 1994 researchers from Kestrel Institute and BBN, Cambridge have been working with personnel from PACAF to model the in-theater scheduling problem. The resulting domain theory has been used to synthesize an increasingly rich series of schedulers generically called ITAS (In-Theater Airlift Scheduler). ITAS runs on a laptop computer (Macintosh Powerbook) which makes it useful for both field and command center operations. BBN has built the user interface based on the commercial Foxpro database package. ITAS schedules the Hurricane Iniki data in a few seconds.

To produce “flyable” schedules it has been necessary to model and schedule a variety of resources, including aircraft, air crews and their duty days, ground crews, parking space for aircraft, and other port restrictions. ITAS has been used in several exercises and was the sole scheduler used in an international exercise during September 1995. It is regarded as being ready to use for contingency purposes.

We have gone through many cycles of learning about the problem from the customer/end-user, elaborating our domain theory, generating new code, and observing PACAF personnel using the scheduler. Although this is a time-consuming process, it seems essential to developing an application that will be used. Nevertheless there has been significant payoff to us as researchers, since the problem features required by the end-user has forced us to generalize and deepen our theories of algorithm design.

4.3 Power Plant Outage Scheduling

We are continuing to develop new scheduling applications using KIDS. A joint project with the Electric Power Research Institute in Palo Alto, California and Rome Laboratory, focuses on the scheduling of maintenance activities during an outage period at nuclear power plants [3]. KIDS is being used to model the problem and to generate high-performance schedulers for maintenance activities. Current schedulers used by the utility industry are slow and handle only a small subset of the important features of the problem. Safety constraints are extremely important, as well as the efficiency of the schedule, since an outage period can costs millions of dollars per day.

5 Concluding Remarks

To conclude, it appears that there is an opportunity to demonstrate that formal software development tools could fill a real need for high-performance schedulers. We believe that part of the success of our approach has been that KIDS can generate complicated constraint propagation codes that may be intimidating to programmers.

To get around the problem that schedulers are usually embedded within a larger system, we develop "plug-compatible" interfaces to our derived schedulers. In this manner we have delivered KTS into the Common Prototyping Environment at Rome Laboratory and we have delivered a variant of KTS running on an Apple Powerbook to a U.S. government customer. The idea is to allow the substitution of our schedulers for existing schedulers and to perform comparative experiments. This is one tenable strategy for integrating formally developed code with legacy systems.

References

[1] Biefeld, E., and Cooper, L. Operations mission planner. Tech. Rep. JPL 90-16, Jet Propulsion Laboratory, March 1990.

[2] Fox, M. S., and Smith, S. F. ISIS – a knowledge-based system for factory scheduling. *Expert Systems 1*, 1 (July 1984), 25–49.

[3] Gomes, C., and Smith, D. R. Synthesis of outage schedulers. Tech. rep., Kestrel Institute, 1994. *submitted for publication*.

[4] John Schank et al. *A Review of Strategic Mobility Models and Analysis.* Rand Corporation, Santa Monica, CA, 1991.

[5] Minton, S., and Philips, A. B. Applying a heuristic repair method to the HST scheduling problem. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control* (San Diego, CA, November 5–8, 1990), DARPA, pp. 215–219.

[6] Sadeh, N. Look-ahead techniques for micro-opportunistic job shop scheduling. Tech. Rep. CMU-CS-91-102, Carenegie-Mellon University, March 1991.

[7] Smith, D. R. Structure and design of global search algorithms. Tech. Rep. KES.U.87.12, Kestrel Institute, November 1987.

[8] Smith, D. R. KIDS – a semi-automatic program development system. *IEEE Transactions on Software Engineering Special Issue on Formal Methods in Software Engineering 16*, 9 (September 1990), 1024–1043.

[9] Smith, D. R., and Lowry, M. R. Algorithm theories and design tactics. *Science of Computer Programming 14*, 2-3 (October 1990), 305–321.

[10] Smith, D. R. Transformational approach to scheduling. Tech. Rep. KES.U.92.2, Kestrel Institute, November 1992.

[11] Smith, D. R. Constructing specification morphisms. *Journal of Symbolic Computation, Special Issue on Automatic Programming 15*, 5-6 (May-June 1993), 571–606.

[12] Smith, D. R., and Parra, E. A. Transformational approach to transportation scheduling. In *Proceedings of the Eighth Knowledge-Based Software Engineering Conference* (Chicago, IL, September 1993), pp. 60–68.

[13] Smith, D. R. Classification approach to design. Tech. Rep. KES.U.93.4, Kestrel Institute, 1993.

[14] Smith, D. R., Parra, E. A., and Westfold, S. J. Synthesis of high-performance transportation schedulers. Tech. Rep. KES.U.95.6, Kestrel Institute, March 1995.

[15] Smith, S. F. The OPIS framework for modeling manufacturing systems. Tech. Rep. CMU-RI-TR-89-30, The Robotics Institute, Carenegie-Mellon University, December 1989.

[16] Srinivas, Y. V., and Jüllig, R. Specware:tm formal support for composing software. Tech. Rep. KES.U.94.5, Kestrel Institute, December 1994. To appear in Proceedings of the Conference on Mathematics of Program Construction, Kloster Irsee, Germany, July 1995.

[17] Zweben, M., Deale, M., and Gargan, R. Anytime rescheduling. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control* (San Diego, CA, November 5–8, 1990), DARPA, pp. 215–219.