A Machine-Checked Proof of the Optimality of a Real-Time Scheduling Policy

Matthew Wilding

The University of Texas at Austin*

Abstract. We describe a mechanically-checked proof of the optimality of earliest-deadline-first (EDF) schedulers on periodic tasks accomplished using the Nqthm theorem prover. We present a formalization of the theorem and discuss why the machine-checked proof is both more complex and more reliable than a corresponding informal proof.

1 Introduction

Real-time applications often have several required functions with different timing constraints. In a seminal paper for building real-time systems, Liu and Layland introduce abstractions that facilitate real-time application development [4]. Using a simple computation model, they exhibit different real-time scheduling policies that choose which of an application's various tasks to assign a processor and argue that these policies have certain useful properties. One scheduling policy is earliest-deadline-first (EDF), which assigns the processor to a task that has earliest deadline among the tasks that are currently running. An EDF scheduler is optimal for tasks requested periodically in the sense that if any schedule can meet the timing constraints on the requests then an EDF schedule will. The original proof of this theorem was flawed [4], but a subsequent published proof developed independently of the machine-checked proof presented in this paper appears correct [9].

EDF schedulers are rarely used because developers generally prefer static priority scheduling policies that ensure that critical tasks not miss a deadline because a less-critical task is selected. However, because EDF schedulers have a higher theoretical CPU utilization, if one is going to guarantee through formal analysis that no deadlines are missed by the tasks of an application it may be preferable to use an EDF scheduling policy.

Formal proofs – proofs constructed in a well-defined proof system where no step in the proof is skipped – are longer and harder to construct than standard proofs, but they allow mechanical checking. A proof checked mechanically by a trusted proof-checking program such as Nqthm [1] is very dependable, but such proofs require absolute precision in the statement of conjectures.

This paper describes a machine-checked proof of the EDF optimality theorem. Section 2 presents Nqthm definitions related to EDF schedules and periodic

* author's present address: Rockwell Collins Advanced Technology Center, Cedar Rapids, Iowa USA 52498. email: mmwildin@collins.rockwell.com

tasks used in the theorem. Section 3 presents the optimality theorem and an example of its application. The EDF optimality proof checked using Nqthm is discussed in Section 4.

2 Some Real-Time Scheduling Definitions

We first present some definitions in the Nqthm logic related to the scheduling of a processor to the tasks of a real-time application. The abstractions we define are similar to those in Liu and Layland [4]. A real-time application is composed of tasks each with a unique name. Occasionally there is a request for a task, which has four elements: the requested task's name, the time of the task request, the task request deadline, and the task's duration. Time is modeled with discrete units, tasks are interruptible, and we ignore task switch overhead.

A schedule is a list that represents a processor schedule assignment over time. A task name tk at location n in a schedule indicates that the processor is assigned to task tk at time n, and **nil** indicates that no task is assigned. A schedule fulfills a task request if during the period no earlier than the task request time and before the task request deadline it contains the task name a number of times equal to the requested task duration. A good schedule is one that fulfills each of a set of task requests.

We use functions provided in the Lisp-inspired Nqthm logic: cons $(a,\ l)$ constructs a new list with a as the first element and l as the rest, car (l) returns the first element of a list, cdr (l) returns all but the first element of a list, listp (l) is true when l is a non-empty list, firstn $(n,\ l)$ returns the first n elements of list l, nthcdr $(n,\ l)$ returns all but the first n elements of list l, append $(a,\ b)$ returns the concatenation of lists a and b, occurrences $(v,\ l)$ returns the number of occurrences of value v in list l. **t** abbreviates the constant TRUE and **f** abbreviates the constant FALSE. Each of the functions 'name', 'request-time', 'deadline', and 'duration' when applied to a task request returns the appropriate value associated with that request.

The property of being a good schedule s with respect to a request list r is easily formalized using Nqthm. (We use Nqthm infix notation for definitions [2].)

DEFINITION:
good-schedule $(s,\ r)$
$=$ **if** listp (r)
then (occurrences (name (car (r)),
firstn (deadline (car (r)) $-$ request-time (car (r)),
nthcdr (request-time (car (r)), s)))
$=$ duration (car (r)))
$\wedge$ good-schedule $(s$, cdr $(r))$
else t endif

Nqthm definitions are executable. In the following examples 'good-schedule' is applied to two example schedules and lists of task requests. Each task request is represented by a list of the task name, request time, deadline, and duration.

```
*(GOOD-SCHEDULE '(A NIL C A B A B C NIL NIL)
                '((A 0 7 3) (B 2 6 2) (C 1 5 1) (C 7 8 1)))
 F
*(GOOD-SCHEDULE '(A NIL C A B B A C NIL NIL)
                '((A 0 7 3) (B 2 6 2) (C 1 5 1) (C 7 8 1)))
 T
```

active-task-requests ($time$, r) returns the list of requests that are in r and for which $time$ is not less than the request time and is less than the deadline.

DEFINITION:
active-task-requests ($time$, r)
= **if** listp (r)
 then if ($time$ < deadline (car (r)))
  ∧ ($time \geq$ request-time (car (r)))
  then cons (car (r), active-task-requests ($time$, cdr (r)))
  else active-task-requests ($time$, cdr (r)) **endif**
 else nil endif

```
*(ACTIVE-TASK-REQUESTS 3 '((A 0 7 3) (B 2 6 2) (C 1 5 1) (C 7 8 1)))
 '((A 0 7 3) (B 2 6 2) (C 1 5 1))
```

unfulfilled ($time$, s, r) returns the list of requests in r for which the task duration is not equal to the number of occurrences of the task name in s before $time$ but no earlier than the task request time.

DEFINITION:
unfulfilled ($time$, s, r)
= **if** listp (r)
 then if occurrences (name (car (r)),
   firstn ($time$ − request-time (car (r)),
    nthcdr (request-time (car (r)), s)))
  = duration (car (r)) **then** unfulfilled ($time$, s, cdr (r))
  else cons (car (r), unfulfilled ($time$, s, cdr (r))) **endif**
 else nil endif

```
*(UNFULFILLED 3 '(A NIL C A B B A C NIL NIL)
                '((A 0 7 3) (B 2 6 2) (C 1 5 1)))
 '((A 0 7 3) (B 2 6 2))
```

least-deadline (r) returns a request in r with least deadline.

DEFINITION:
least-deadline(r)
$=$ **if** listp(r)
then if listp(cdr(r))
then if deadline(car(r)) $<$ deadline(car(cdr(r)))
then least-deadline(cons(car(r), cdr(cdr(r))))
else least-deadline(cdr(r)) **endif**
else car(r) **endif**
else nil endif

```
*(LEAST-DEADLINE '((A 0 7 3) (B 2 6 2)))
 '(B 2 6 2)
```

The term edf(n, r) returns a schedule of length n for requests r such that an unfulfilled active task request with least deadline is chosen at each moment in the schedule if one exists or **nil** if there is no such request.

DEFINITION:
edf(*lengthschedule*, r)
$=$ **if** *lengthschedule* $\simeq$ 0 **then nil**
else let s **be** edf(*lengthschedule* $-$ 1, r)
in
let *unfulfilled* **be** unfulfilled(*lengthschedule* $-$ 1, s, active-task-requests(*lengthschedule* $-$ 1, r))
in
if listp(*unfulfilled*)
then append(s, list(name(least-deadline(*unfulfilled*))))
else append(s, list(**nil**)) **endif endlet endlet endif**

```
*(EDF 10 '((A 0 7 3) (B 2 6 2) (C 1 5 1) (C 7 8 1)))
 '(A C B B A A NIL C NIL NIL)
```

EDF scheduling would appear to be a sensible way to schedule task requests. Note that the EDF scheduler applied to the task requests of the previous examples generates a good schedule.

```
*(GOOD-SCHEDULE '(A C B B A A NIL C NIL NIL)
                '((A 0 7 3) (B 2 6 2) (C 1 5 1) (C 7 8 1)))
 T
```

Our model for tasks is that they are *periodic*, meaning that task requests for a particular task occur with constant frequency. The function 'periodic-tasksp' identifies valid lists of periodic tasks, each of which is a triple containing a unique name, a positive period, and a positive duration.

```
*(PERIODIC-TASKSP '((A 3 2) (B 0 3)))
 F
*(PERIODIC-TASKSP '((A 3 2) (B 9 3)))
 T
```

The term periodic-tasks-requests $(pts,\ n1,\ n2)$ generates task requests from periodic tasks list *pts*. A request is generated for each periodic task at every time less than *n2* that is the sum of *n1* and a multiple of the task's period. The deadline of a periodic task request is the next request of that task.

DEFINITION:
periodic-task-requests (*pt*, *starting-time*, *ending-time*)
= **if** periodic-taskp (*pt*)
  then if *starting-time* < *ending-time*
    then cons (list (tk-name (*pt*), *starting-time*,
          starting-time + tk-period (*pt*), tk-duration (*pt*)),
        periodic-task-requests (*pt*, *starting-time* + tk-period (*pt*),
              ending-time))
    else nil endif
  else nil endif

DEFINITION:
periodic-tasks-requests (*pts*, *starting-time*, *ending-time*)
= **if** periodic-tasksp (*pts*)
  then if listp (*pts*)
    then append (periodic-task-requests (car (*pts*), *starting-time*, *ending-time*),
          periodic-tasks-requests (cdr (*pts*), *starting-time*, *ending-time*))
    else nil endif
  else nil endif

```
*(PERIODIC-TASKS-REQUESTS '((A 3 2) (B 9 3)) 0 18)
'((A 0 3 2) (A 3 6 2) (A 6 9 2) (A 9 12 2)
  (A 12 15 2) (A 15 18 2) (B 0 9 3) (B 9 18 3))
```

Nqthm can reason about rationals [6], but it is simpler to introduce some functions that facilitate expressing the EDF theorem using integer arithmetic. big-period (*pts*) is the product of the task periods in *pts*. cpu-utilization (*pts*, *n*) returns the sum of $(n * duration) \div period$ for each task in *pts*.

DEFINITION:
big-period (*pts*)
= **if** listp (*pts*) **then** tk-period (car (*pts*)) ∗ big-period (cdr (*pts*))
  else 1 **endif**

DEFINITION:
cpu-utilization (*pts*, *bigp*)
= **if** listp (*pts*)
  then ((*bigp* ∗ tk-duration (car (*pts*))) ÷ tk-period (car (*pts*)))
    + cpu-utilization (cdr (*pts*), *bigp*)
  else 0 **endif**

```
*(BIG-PERIOD '((A 3 2) (B 9 3)))
 27
*(CPU-UTILIZATION '((A 3 2) (B 9 3)) 27)
 27
```

3 EDF Optimality for Periodic Tasks

We present the optimality theorem about EDF schedules on periodic tasks using the definitions of the previous section. If the *pts* is a list of periodic tasks such that $\sum_{tasks} duration/period \leq 1$ and n is a multiple of big-period(pts), then the EDF schedule satisfies the requests of *pts* through n time units.

$$
\begin{array}{ll}
\multicolumn{2}{l}{((\text{big-period}\,(pts) \geq \text{cpu-utilization}\,(pts,\ \text{big-period}\,(pts)))} \\
\wedge & \text{periodic-tasksp}\,(pts) \\
\wedge & ((n \ \mathbf{mod}\ \text{big-period}\,(pts)) = \mathtt{0})) \\
\rightarrow & \text{good-schedule}\,(\text{edf}\,(n,\ \text{periodic-tasks-requests}\,(pts,\ \mathtt{0},\ n)), \\
 & \qquad\qquad\qquad \text{periodic-tasks-requests}\,(pts,\ \mathtt{0},\ n))
\end{array}
$$

We call this an optimality theorem since the theorem shows that any set of periodic tasks for which there exists a good schedule can be scheduled using an EDF scheduler. We illustrate the application of this remarkable theorem on a small example. Assume task A has period 16 and duration 4, task B has period 5 and duration 2, and task C has period 3 and duration 1. If we let N be 240 and PTS the tasks described above, each of the hypotheses of the theorem is satisfied.

```
*(BIG-PERIOD PTS)
 240
*(CPU-UTILIZATION PTS 240)
 236
*(NOT (LESSP (BIG-PERIOD PTS) (CPU-UTILIZATION PTS (BIG-PERIOD PTS))))
 T
*(PERIODIC-TASKSP PTS)
 T
*(EQUAL (REMAINDER N (BIG-PERIOD PTS)) 0)
 T
```

Since we have proved the theorem and the hypotheses are satisfied, we know that the EDF schedule is a good schedule for the generated task requests.

```
*(GOOD-SCHEDULE
    (EDF N (PERIODIC-TASKS-REQUESTS PTS 0 N))
    (PERIODIC-TASKS-REQUESTS PTS 0 N))
 T
```

The first few assignments of the EDF schedule are calculated below, and the initial part of the schedule is displayed pictorially in Figure 1.

```
*(EDF N (PERIODIC-TASKS-REQUESTS PTS 0 N))
 '(C B B C A B C B A C B B C A A C B B C A B C B A C B B C A
    A C B B C A B C B A C B B C A A C B B C A B C B A C B B
    C A A C B B C A B C B A C B B C A A C B B C NIL B C B A
    ....
```

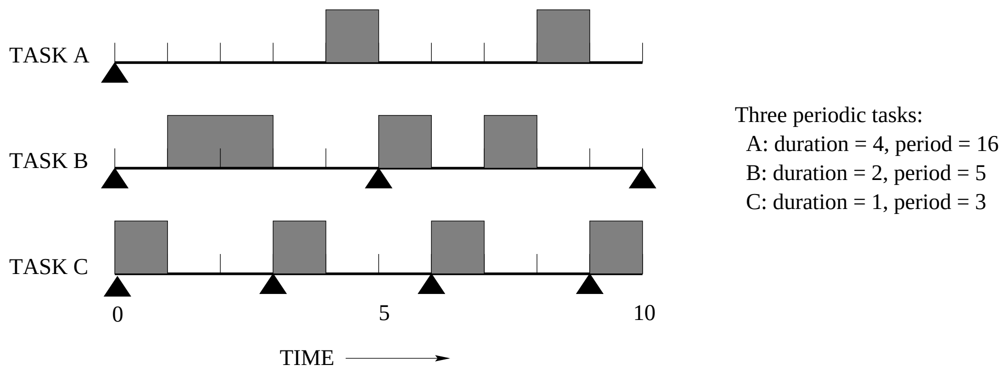

Fig. 1. An Example EDF Schedule

4 EDF Optimality Proof

4.1 An Informal Proof of EDF Optimality

Theorem 1. *Let R be a periodic request history corresponding to a periodic set P of m tasks. Let (tk_i, d_i, p_i) denote the ith element of P. $\sum_{i=1}^{m}(d_i/p_i) \leq 1$ implies the earliest deadline first schedule of R has no overflow.*

Proof: Let c be the product of the periods of P. Let P' be identical to P except let each duration d'_i and period p'_i in P be c times the corresponding values d_i and p_i in P. Let R' be a task request history of P'.

Let S' be a schedule for R' by repeating length(R) times the following:

$$(tk_1)^{d'_1/p_1}\ (tk_2)^{d'_2/p_2}\ \ldots (tk_m)^{d'_m/p_m}\ \mathbf{nil}^{c-\sum d'_i/p_i}$$

Since $\sum_{i=1}^{m}(d_i/p_i) \leq 1$, length($R$)*$c$ is the length of S' and S' does not overflow for R'.

We define swap(t, S, R) for time t, schedule S, and request history R as follows. Let tk be the least task with earliest absolute deadline unfulfilled at t. Return a schedule identical to S except that the task name at location t and the task name at the location of the first occurrence of tk no earlier than t are swapped. If schedule S does not overflow for request history R, then neither does swap(i, S, R).

Let $S'_0 = S'$ and $S'_{n+1} =$ swap(n, S'_n, R'). Let $S'' = S'_{length(S')}$. Note that S'' does not overflow for R' and that the element n in S'' is equal to the $n - (n \ \mathbf{mod}\ c)$ element of S''. Let S''' be composed of the elements of S'' in locations that are multiples of c. S''' is the earliest deadline first schedule of R and does not overflow.

Q.E.D.

4.2 The Nqthm Proof

The Nqthm proof of EDF optimality consists of approximately 500 "events" – not including a previously-proved library of arithmetic theorems upon which this proof depends [7] – that follow the informal proof described above. Nontrivial theorems are proved using Nqthm by proving lemmas that when applied automatically in later proofs lead to the desired proof. Nqthm provides the confidence associated with formal proof without the drudgery of writing the complete formal proof. Most of the 500 events of the EDF proof are lemmas designed to guide Nqthm to the proof of later theorems. The events are listed in [8]. The proof effort took approximately 3 man-months, and a comparison of the informal proof in Section 4.1 and the Nqthm proof in [8] makes obvious that the Nqthm proof required greater effort. Is this effort needed because formal proof is difficult, or is the difference simply due to difficulty using Nqthm?

We believe we faced greater difficulties proving this theorem using Nqthm because of the precision formal mathematics requires. Nqthm merely enforces the requirement that we be precise and not skip any details of the proof. On several occasions during the development of the proof the final theorem appeared nearly proved but was not. The complexity of doing formal mathematics is apparent in the complexity of the lemmas listed in [8]. Formal mathematics requires we be precise, and Nqthm succeeds in enforcing this requirement.

Many of the concepts needed in the proof of this theorem would not be introduced explicitly in an informal proof. As an example we examine 'firstn', which returns the first n elements of a list. This concept is defined in the Nqthm proof of the EDF optimality theorem because it is needed to define some of the notions in the proof, but a mathematician not doing formal proofs would not trouble himself to define it explicitly. In the Nqthm logic we define this concept as follows.

DEFINITION:
firstn (n, *list*)
$=$ **if** $n \simeq \texttt{0}$ **then nil**
else cons (car (*list*), firstn ($n - 1$, cdr (*list*))) **endif**

In order for Nqthm to reason about schedules and task requests many theorems are proved that "program" the theorem prover [1, 8]. Figure 2 presents some of the theorems involving 'firstn' that were proved in order to prove EDF optimality but which would probably be skipped in an informal proof. Each of these lemmas is proved using Nqthm and is applied subsequently in proofs involving 'firstn'. We believe that these facts about 'firstn' are needed in formal proofs involving 'firstn' whether or not one is using Nqthm. If one wishes to prove formally something that involves 'firstn' and 'append', for example, there is no getting around the requirement to prove theorem (2). An Nqthm user is forced to prove something akin to this theorem as a part of the proof of EDF optimality because any formal proof of EDF optimality requires it.

General lemmas like these about 'firstn' can be reused in future Nqthm proofs that involve 'firstn', because the collection of theorems proved about subsidiary

1. $\text{length}\,(\text{firstn}\,(n,\ \mathit{list})) = \text{fix}\,(n)$
2. $\text{firstn}\,(n,\ \text{append}\,(\mathit{l1},\ \mathit{l2}))$
   $=\ \ \textbf{if}\ \text{length}\,(\mathit{l1}) < n\ \ \textbf{then}\ \text{append}\,(\mathit{l1},\ \text{firstn}\,(n - \text{length}\,(\mathit{l1}),\ \mathit{l2}))$
   $\qquad \textbf{else}\ \text{firstn}\,(n,\ \mathit{l1})\ \textbf{endif}$
3. $\text{firstn}\,(\text{length}\,(x),\ x) = \text{plist}\,(x)$
4. $\text{plist}\,(\text{firstn}\,(n,\ l)) = \text{firstn}\,(n,\ l)$
5. $\text{nthcdr}\,(n,\ \text{firstn}\,(n + x,\ s)) = \text{firstn}\,(x,\ \text{nthcdr}\,(n,\ s))$
6. $\text{firstn}\,(n,\ \text{replace-nth}\,(i,\ v,\ l))$
   $=\ \ \textbf{if}\ i < n\ \ \textbf{then}\ \text{replace-nth}\,(i,\ v,\ \text{firstn}\,(n,\ l))$
   $\qquad \textbf{else}\ \text{firstn}\,(n,\ l)\ \textbf{endif}$
7. $\text{nth}\,(n,\ \text{firstn}\,(\mathit{n2},\ s))$
   $=\ \ \textbf{if}\ n < \mathit{n2}\ \ \textbf{then}\ \text{nth}\,(n,\ s)$
   $\qquad \textbf{else}\ \mathsf{0}\ \textbf{endif}$
8. $\text{firstn}\,(a,\ \text{firstn}\,(b,\ x))$
   $=\ \ \textbf{if}\ b < a\ \ \textbf{then}\ \text{append}\,(\text{firstn}\,(b,\ x),\ \text{repeat}\,(a - b,\ \mathsf{0}))$
   $\qquad \textbf{else}\ \text{firstn}\,(a,\ x)\ \textbf{endif}$
9. $\text{firstn}\,(\mathsf{1},\ s) = \text{list}\,(\text{car}\,(s))$
10. $(i < j) \rightarrow (\text{nth}\,(i,\ s) \in \text{firstn}\,(j,\ s))$
11. $(\text{car}\,(x) \in \text{firstn}\,(n,\ x)) = (n \not\simeq \mathsf{0})$
12. $((i \geq y) \wedge (i < (x + y))) \rightarrow (\text{nth}\,(i,\ s) \in \text{firstn}\,(x,\ \text{nthcdr}\,(y,\ s)))$
13. $(x \notin l) \rightarrow ((x \in \text{firstn}\,(n,\ l)) = ((x = \mathsf{0}) \wedge (\text{length}\,(l) < n)))$
14. $\text{firstn}\,(\mathit{n1},\ \text{repeat}\,(\mathit{n2},\ v))$
   $=\ \ \textbf{if}\ \mathit{n2} < \mathit{n1}\ \ \textbf{then}\ \text{append}\,(\text{repeat}\,(\mathit{n2},\ v),\ \text{repeat}\,(\mathit{n1} - \mathit{n2},\ \mathsf{0}))$
   $\qquad \textbf{else}\ \text{repeat}\,(\mathit{n1},\ v)\ \textbf{endif}$
15. $\text{nthcdr}\,(n,\ \text{firstn}\,(n,\ l)) = \textbf{nil}$
16. $\text{listp}\,(\text{firstn}\,(n,\ l)) = (n \not\simeq \mathsf{0})$
17. $(\textbf{nil} = \text{firstn}\,(n,\ l)) = (n \simeq \mathsf{0})$
18. $n \geq \text{occurrences}\,(v,\ \text{firstn}\,(n,\ l))$
19. $\text{firstn}\,(a + b,\ l) = \text{append}\,(\text{firstn}\,(b,\ l),\ \text{firstn}\,(a,\ \text{nthcdr}\,(b,\ l)))$

Fig. 2. Some theorems proved about 'firstn'

functions constitutes a strategy for proving theorems about them. This is not necessarily the case when using proof systems that encourage the development of less-automatic proofs, such as PC-Nqthm [3] or PVS [5]. The EDF optimality proof used previously-proved arithmetic facts developed for Nqthm-checked proofs of other theorems [7].

It is our belief that the difficulties we overcame to construct an Nqthm proof of this theorem using the approach presented in this paper result from the complexity and precision of formal mathematics rather than a deficiency in Nqthm.

5 Conclusions

Proofs of this kind lead to precise statements of theorems useful for computer system verification. Informal proofs often make intuitive leaps that are difficult to justify, or omit details that can lead to misinterpretation. The original proof of EDF optimality was flawed because a step in the informal proof was not valid, but if one trusts Nqthm and accepts the formalization presented in this paper one need not examine the proof to conclude that EDF optimality is a theorem.

References

1. R. S. Boyer and J S. Moore. *A Computational Logic Handbook.* Academic Press, Boston, 1988.
2. R. S. Boyer and J S. Moore. *Chapter 14 of A Computational Logic Handbook, 2nd edition.* Academic Press, Boston, 1997. (Distributed in Nqthm-1992 release available from Computational Logic, Inc.).
3. Matthew Kaufmann. A user's manual for an interactive enhancement to the Boyer-Moore theorem prover. Technical Report 19, Computational Logic, Inc., May 1988.
4. C.L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, 1973.
5. S. Owre, N. Shankar, and J. M. Rushby. *User Guide for the PVS Specification and Verification System (Beta Release).* Computer Science Laboratory, SRI International, Menlo Park, CA, February 1993.
6. Matthew Wilding. A mechanically-checked correctness proof of a floating-point search program. Technical Report 56, Computational Logic, Inc., May 1990.
7. Matthew Wilding. Proving Matijasevich's lemma with a default arithmetic strategy. *Journal of Automated Reasoning*, 7(3), September 1991.
8. Matthew Wilding. *Machine-Checked Real-Time System Verification.* PhD thesis, University of Texas, May 1996. ftp.cs.utexas.edu/pub/boyer/wilding-diss.ps.gz.
9. Zheng Yuhua and Zhou Chaochen. A formal proof of the deadline driven scheduler. *Formal Techniques in Real-Time and Fault-Tolerant Systems, Third International Symposium*, pages 756–775, 1994.

Acknowledgments: I thank Al Mok, Bob Boyer, Mathai Joseph, Matt Kaufmann, and Bill Young for many helpful suggestions about drafts of this paper. The views and conclusions contained in this document are those of the author. This work was supported in part at Computational Logic, Inc., by the Defense Advanced Research Projects Agency, ARPA Order 7406.