INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Analysis and Design of On-line Decision-Making Solutions for Time-Critical Planning and Scheduling Under Uncertainty

by

Lloyd Geoffrey Greenwald

B. S. E., University of Pennsylvania, 1988

Sc. M., Brown University, 1992

Thesis

Submitted in partial fulfillment of the requirements for the

Degree of Doctor of Philosophy in the Department of Computer Science

at Brown University

May 1997

UMI Number: 9738550

UMI Microform 9738550
Copyright 1997, by UMI Company. All rights reserved.

This microform edition is protected against unauthorized copying under Title 17, United States Code.

UMI
300 North Zeeb Road
Ann Arbor, MI 48103

Copyright
by
Lloyd Geoffrey Greenwald
1997

This dissertation by Lloyd Geoffrey Greenwald
is accepted in its present form
by the Department of Computer Science
as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date 10/30/96

Thomas L. Dean

Recommended to the Graduate Council

Date 25 October 96

Leslie Pack Kaelbling

Date 10/25/96

Shlomo Zilberstein
(University of Massachusetts at Amherst)

Approved by the Graduate Council

Date Dec. 13, 1996

Vita

I was born December 12, 1965 in Queens, NY, USA. I received my Bachelor of Science and Engineering in Computer Science and Engineering from the University of Pennsylvania in 1988, graduating *Magna Cum Laude*. I received my Master of Science in Computer Science from Brown University in 1992.

Acknowledgements

I thank Tom Dean for his help and guidance, and especially for taming my quixotic desire to solve all the problems of on-line decision making in this small space. I'm still amazed that we managed to mold this mass of ideas into a coherent work of which we can both be proud.

I thank Shlomo Zilberstein and Leslie Kaelbling for their efforts in improving this work through many thoughtful comments. I am also indebted to Leslie for providing me the opportunity to help design and teach the introductory robotics course at Brown.

So as not to be embarrassed by gaps in memory, I jointly acknowledge all the unnamed colleagues and friends who have touched my life during the past few years.

I thank my family, and especially my parents, for more than 30 years of love and support.

Most importantly, this thesis is dedicated to my two beautiful ladies.

To my little Jacelyn Rose, who turned me into a part-time student and full-time father when I needed it the most; and whose smile is my proudest achievement.

And to my wonderful Elaine. Whose unwavering friendship and love are certainties upon which I could always rely. Elaine, for constantly enriching my life with your presence, I thank you.

Contents

Vita iii

Acknowledgements iv

List of Figures x

1 Thesis Summary and Road Map 1

2 A Formalization of On-line Decision Making 15

2.1 Specifying a Decision-Making Environment 16

2.2 A Circuit Composition Formalization for Time-Critical Control System Design 17

2.3 Decision-Making Behavior 21

2.4 Discussion of Properties of Difficult Classes of On-line Decision-Making Problems 23

3 Solving On-line Decision-Making Problems by Sequencing Circuits 27

3.1 An On-line Decision-Making Solution Design Scenario 28

3.2 Decoupling Decision-Making Behavior and Time-Critical Control System Design 30

3.3 Evaluating Decision-Making Behavior that is Achieved by Sequencing a Collection of Pre-compiled Decision-Making Circuits 33

3.4 Circuit Paging 41

3.5 Combining a Time-Critical Control System and Decision-Making Environment to Derive Predictable Temporal Patterns of Situations 45
3.6 Discussion 50

4 Solving a Circuit Paging Problem **51**
4.1 Paging Arbitrary Collections of Executable Circuits 52
4.2 Existence of Solutions 53
4.3 Complexity 55
4.4 Approximation Algorithm 57
4.5 Executability 61
4.6 Easy Special Case 63
4.7 Developing an Exact Algorithm 65
4.8 Discussion 71

5 Arbitrary-Circuit Paging: Real-time Avionics Application **72**
5.1 Real-time Avionics 72
5.2 Conditional Real-time Avionics Scheduling 76
5.3 Guaranteeing Behavior Under Conditional Real-time Avionics Scheduling Solutions 79
5.3.1 Conditional Frequency Model 80
5.3.2 Temporal Patterns of Situations 81
5.3.3 Coverage Requirements 84
5.3.4 Guaranteed Safe Behavior 85
5.4 Discussion 87

6 An Empirical Evaluation of Framework and Techniques **88**
6.1 Improving Solutions by Accommodating Additional Flight Operations 89
6.1.1 A Restricted Static Scheduling Solution 90
6.1.2 Relating Functionality Improvement to Parameterized Flight Operation Sampling Demand 91
6.2 Schedule Server Resources and Conditional Schedule Resource Requirements 94

6.3 Generating Collections of Conditional Schedules and Environmental Coverage Requirements 95
6.3.1 Conditional Schedules 95
6.3.2 Environmental Coverage Requirements 96
6.3.3 Understanding Uniformly Generated Collections 97
6.4 Discussion 100

7 Results of Empirical Evaluation **101**
7.1 Test Plan 101
7.2 An Instantiation of the Executable Set Cover Solution Scheme . . 104
7.2.1 Algorithm Sketch 104
7.2.2 Simplified Executability Equations 105
7.2.3 Deriving an MILP That Combines Set Cover Constraints and Executability Equations 107
7.3 Results of the Empirical Study 108
7.3.1 Overall Results of Preliminary Round of Test Cases 109
7.3.2 Overall Results of a More Difficult Round of Test Cases . . 111
7.3.3 A Closer Look at the Computational Issues of Increasing Functionality through Conditional Scheduling 112
7.4 Discussion 116

8 Related Research **117**
8.1 Resource-Bounded Problem Solving 117
8.2 Reactive Control Policies 121
8.3 Deliberation/Reaction System Design 124
8.4 Real-Time Scheduling 127
8.5 Paging and Prefetching 129
8.6 Personal Prior Research Results 130

9 Limitations and Future Work **134**
9.1 Relaxing the Complete Coverage Requirement 135
9.2 Understanding the Limitations of Specific Time-Critical Control Systems 136

9.3 Developing New Time-Critical Control System Architectures . . . 138
9.3.1 Fixed paging strategy and long-term decision-making behavior . . . 138
9.3.2 Fixed retrieval intervals and short-term decision-making behavior . . . 141
9.4 Extending the Properties of Strategy and Decision-Making Circuits 143
9.5 Conclusion . . . 145

A Proofs and Omitted Details **147**
A.1 Decoupling Decision-Making Behavior Formulation and Time-Critical Control System Design . . . 147
A.2 The Existence of Arbitrary-Circuit Paging Solutions . . . 148
A.3 Arbitrary-Circuit Paging Complexity . . . 150
A.4 Reduction of Coverage Requirements . . . 152
A.5 Lower Bound Based On Greedy-Set-Cover . . . 152
A.6 Executability of Paging Strategies . . . 153
A.7 Upper Bound on Easy Circuit-Covering . . . 154
A.8 Existence of Easy Circuit-Covering . . . 154
A.9 Upper Bound on On-line Cache Resources for Easy Circuit-Covering 154
A.10 Circuit-Specific Lower Bound Based On Greedy-Set-Cover . . . 155
A.11 Relationship Between $|\mathcal{O}|$, $\bar{x}$, and cf . . . 156
A.12 Executability Under Empirical Study . . . 157
A.13 MILP Provides Correct and Complete Solution . . . 158

List of Tables

5.1 Situation-specific conditional sampling frequencies 81

5.2 Deriving coverage requirements given temporal patterns of situations based on single-tick cycles. and a corresponding frequency model 85

6.1 Restricted static scheduling solution 90

6.2 Test parameters 93

7.1 Test space – I 102

7.2 Test space – II 102

7.3 Calculated parameters – I 102

7.4 Calculated parameters – II 103

7.5 Overall results of preliminary round of test cases 109

7.6 Overall results of the more difficult round of test cases 111

7.7 Evidence of result skew as flight operations approach 20 114

List of Figures

1.1 Comparing on-line decision making to hierarchical storage 3
1.2 Abstracting on-line decision-making tradeoffs as a two-level paging architecture 5
1.3 Transition structure representing all possible interactions of system and environment 6
1.4 Predictable temporal patterns 6
1.5 Design-time tradeoffs may lead to minimal set covering problems . 7
1.6 Providing an on-line decision-making solution under a two-level paging architecture 9
1.7 Example conditional real-time avionics system components 10
1.8 Distributed multiprocessor avionics hardware augmented with a schedule server LRM 10

2.1 On-line decision-making interaction 16
2.2 A decision-making environment described by a discrete-time, discrete-state dynamical system 18
2.3 An execution hardware composed of a decision-making circuit π and a strategy circuit γ 20
2.4 An example execution trajectory 21
2.5 An example decision-making behavior 22

3.1 An example sequence of steps toward generating a solution to an on-line decision-making problem 31
3.2 An example sequencing of decision-making circuits 35

3.3 An example desired behavior that is interchangeable with respect to a collection of decision-making circuits 38
3.4 A circuit paging architecture 44
3.5 An example sequencing of decision-making circuits under a circuit paging architecture of fixed retrieval interval $n = 3$ 44
3.6 Fixed interval paging domain 48
3.7 Executable circuit coverage requirement 49
3.8 Predictable temporal patterns 50

4.1 Constructing the set of ground elements X from the flattened collection of sets of reachable situations 58
4.2 Constructing the family of sets Y by comparing a collection of decision-making circuits C to the flattened collection of sets of reachable situations R'. 58
4.3 Providing an on-line decision-making solution under a two-level paging architecture 60
4.4 Executable circuit cover procedure 68
4.5 Executable-Set-Cover(X, Y) 69

5.1 Distributed multiprocessor avionics hardware 73
5.2 Worst-case static schedules are fixed at design time *i.e.* they do not change from one cycle to the next 75
5.3 Example conditional real-time avionics system components 77
5.4 Distributed multiprocessor avionics hardware augmented with a schedule server LRM 78
5.5 Conditional schedules alter scheduling of flight operations over time 79
5.6 Example flight environment transition model and per situation conditional frequency model 83
5.7 Executable covering derived by comparing flattened coverage requirements to collection of conditional schedules 86

6.1 Worst-case static schedule under restricted model 91

6.2 Minimum sampling reduction factor as function of desired number of flight operations, given that $\bar{x} \in \{1, 3, 5, 7, 9\}$ operations must be sampled at worst-case frequency. 93
6.3 Growth in total possible schedules as function of number of desired flight operations 98
6.4 All possible sets of at most four flight operations 98
6.5 An example test case: i. uniformly generated schedule collection of size $6 = sf * 16$, $sf = .375$; ii. uniformly generated requirements collection of size $4 = mrf * 11$, $mrf = .36$, each requiring at most $2 = rf * 4$, $rf = .5$, worst-case frequency flight operations; iii. flattened collection of requirements; iv. schedules that cover at least one flattened requirement, and three possible coverings. . . . 99

7.1 Fraction of tests achieving each outcome as function of increased functionality (number of supported flight operations) 113
7.2 Fraction of tests shown executable by each algorithm component . 114
7.3 Decrease in distribution of test cases as the number of desired flight operations increase 115
7.4 Abrupt change in average run time at the high end of desired flight operations 115

9.1 Open-loop paging architecture 139
9.2 Architecture for simultaneously paging both decision-making circuits and strategy circuits 141
9.3 Bipartite cache, each half split to include decision-making circuit, strategy circuit and previously generated fringe data 145

Chapter 1

Thesis Summary and Road Map

A goal of this dissertation is to understand the tradeoffs involved in designing a system that must interact with a dynamic, uncertain environment. These tradeoffs arise when the system must make quick decisions about how to respond to situations in the environment and has limited on-line time and space resources to employ in decision making. Understanding the behavior of an on-line decision-making solution requires analyzing at design time the impact of decisions on future situations of the environment.

The class of problems in which these tradeoffs arise includes those of time-critical planning and scheduling under uncertainty. This thesis focuses on the example problem of designing a real-time avionics system to control a modern aircraft, such as the avionics system designed by Honeywell for the Boeing 777 [10]. However, similar tradeoffs arise in many varied problems, from mobile robotics, manufacturing automation, and intelligent agents to medical decision-making systems, real-time financial forecasting, and transportation system design.

In a real-time avionics system, multiple flight operations, including those critical to the safe operation of the aircraft, are scheduled on a distributed multiprocessor hardware system. Schedules must guarantee safe behavior, under all possible uncertain in-flight conditions. Time-critical response must be provided, despite on-line computational time and space resources that are tightly limited due to power and weight considerations. In order to understand the tradeoffs in designing a real-time avionics system, a precise design-time understanding of

execution-time dynamics is required.

Researchers have proposed a variety of methods for designing systems that interact with dynamic, uncertain worlds. A common property of such on-line decision-making solutions is the combination of fast computations that react to situations as they occur and slower computations that plan ahead for future situations. These gradations of decision making are designed to cope with uncertainty in the interaction of system and environment, and the necessity of making tradeoffs to carefully allocate bounded on-line resources. However, the combined behavior of slow deliberation and fast reaction is rarely well-understood beyond limited domain testing.

In this chapter, we briefly summarize the following contributions and components of this dissertation:

1. Develop a framework for expressing and evaluating tradeoffs, at design time, in designing systems that interact with dynamic, uncertain worlds. Including:

   - Formulating an on-line decision-making solution as a time-critical control system with specific time/space tradeoffs; explicitly modeling the relationships between all time and space resources;
   - Characterizing the predictable temporal patterns in the interaction of a dynamic, uncertain world and a given time-critical control system; explicitly capturing the interdependency of decision-making responses across time;
   - Formulating a decision problem to capture the necessary and sufficient conditions under which a system can be shown, at design time, to solve the corresponding on-line decision-making problem; and
   - Providing analytical techniques for evaluating and solving a particular class of decision problems; including complexity analyses, approximation algorithms, and combinatorial optimization enumeration schemes;

2. Demonstrate the framework by developing a new approach to designing and analyzing real-time avionics systems;

3. Empirically evaluate the framework and analytical techniques with respect to the real-time avionics scheduling approach; and

4. Explore limitations and extensions of the framework.

On-line decision making as a hierarchical storage problem

In order to better understand the behavior of an on-line decision-making system that consists of both slow deliberation and fast reaction, this thesis develops conceptual abstractions based on hierarchical storage systems. This comparison is illustrated in Figure 1.1. In hierarchical storage, fast low-capacity levels of memory, such as processor cache, combine with slow high-capacity levels, such as main memory. Since the capacity of fast cache is limited, data is distributed and transferred across levels of the hierarchy in an attempt to achieve overall behavior comparable to the speed of the fastest level with the capacity of the slowest level.

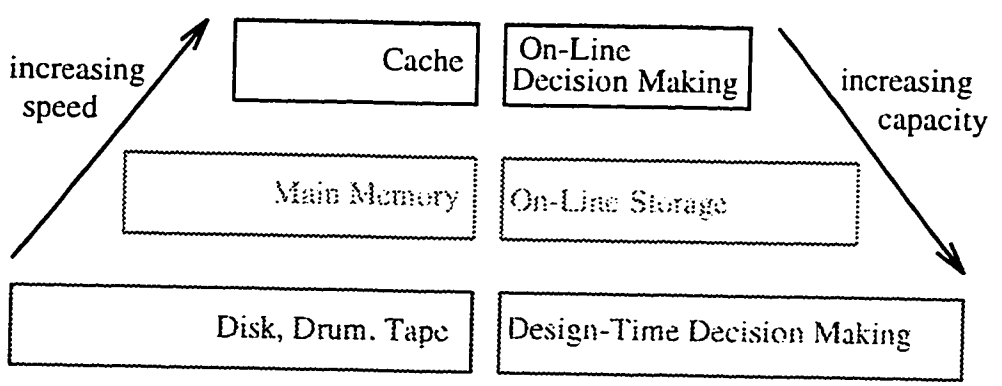

Figure 1.1: Comparing on-line decision making to hierarchical storage

Similarly, in on-line decision making, processing is distributed between fast reactive computations performed in response to situations of a given execution and slower deliberative computations performed at prior points, such as at design time. The results of processing are stored and transferred across the hierarchy in order to achieve the responsiveness of the fastest reactive computations with

bounded on-line resources. In on-line decision making, many gradations of deliberation speed can co-exist within a system. The analysis challenge is to characterize the combined behavior that results from the interactions of multiple levels of decision making.

In Chapter 2 we present a formalization that captures a wide range of on-line decision-making problems. In Chapter 3 we develop techniques for modeling an on-line decision-making solution as a time-critical control system based on a two-level paging hierarchy.

Time-critical control systems and predictable temporal patterns

An important contribution of this thesis is a framework for understanding the design-time tradeoffs involved in designing a system that must interact with a dynamic, uncertain environment. The foundation of the framework is to formulate an on-line decision-making solution as a time-critical control system and identify the time/space tradeoffs inherent in the formulation. The framework is supported by a characterization of predictable temporal patterns in the interaction of system and environment, under all possible on-line executions. The characterization and formalization of predictable temporal patterns are a unique contribution. The patterns capture design-time understanding of short-term and long-term interactions in which later situations of the environment are dependent upon earlier actions of the control system.

Specific technical results of this thesis center around a two-level time-critical control system developed in Chapter 3. This control system is intended to conceptually model the tradeoffs inherent in the interaction of slow deliberation and fast reaction. This particular model abstracts these tradeoffs as a problem of paging decision-making circuits from a bounded second-level store to a bounded target execution system (see Figure 1.2). Each paged decision-making circuit controls the execution hardware during a well-specified time interval. Paging decisions are made in response to situations of the environment at fixed intervals of time.

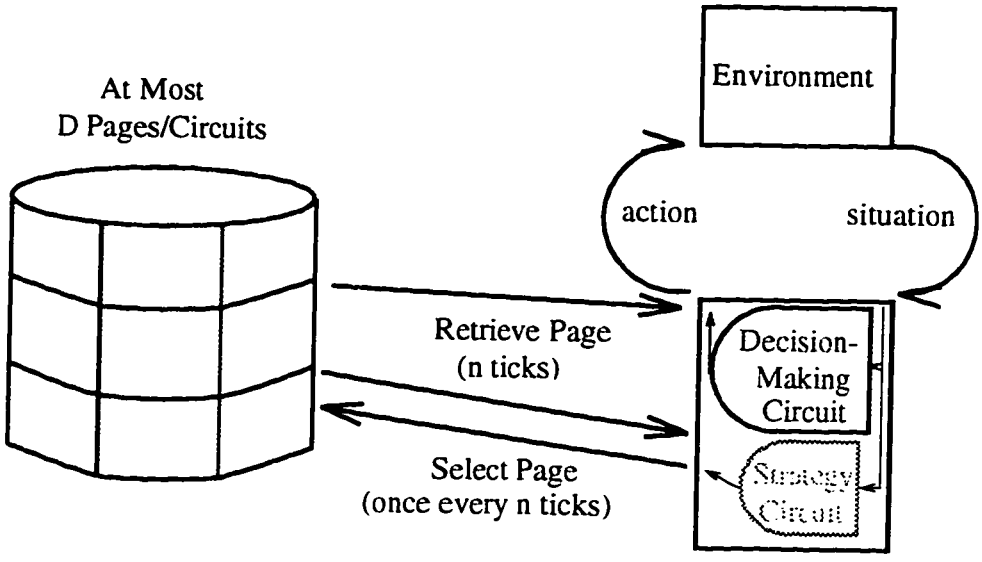

Figure 1.2: Abstracting on-line decision-making tradeoffs as a two-level paging architecture

The decision-making circuits model fast reactive computations, and the closed-loop paging of decision-making circuits into the execution hardware models slow deliberative computations.

All resources required to implement this interaction are explicitly modeled, including those of the circuit to perform paging. The interaction is then analyzed under specific environmental models and models of resource requirements and limitations. The predictable temporal patterns of this interaction consist of the long-term patterns of the environment at the pre-specified paging intervals, and the short-term patterns of the environment between each interval.

Figure 1.3 represents the possible interactions of system and environment as a state-transition graph. The upper view of the graph depicts the long-term patterns in terms of nodes that may be reached at fixed time intervals. The lower view depicts the short-term patterns in terms of nodes that may be reached between intervals. In Chapters 3 and 4 we show that the predictable temporal patterns of interest for the paging architecture of Figure 1.2 may be captured by the set of situations F^n and collection of reachable sets of situations R of Figure 1.4.

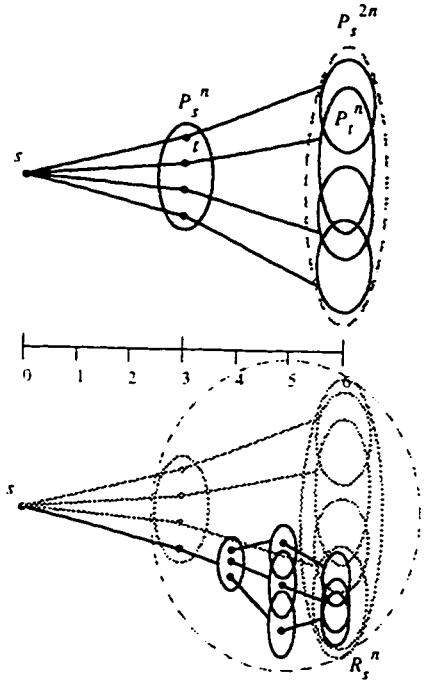

Figure 1.3: Transition structure representing all possible interactions of system and environment

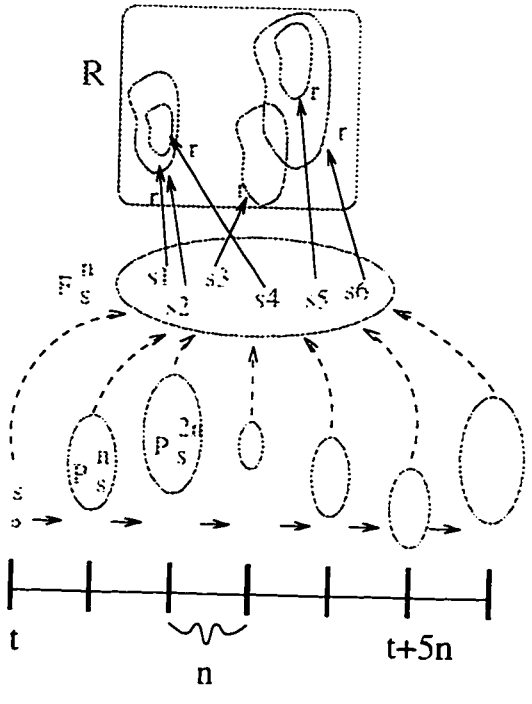

Figure 1.4: Predictable temporal patterns

Design-time tradeoffs lead to combinatorial optimization problems

Design-time tradeoffs may be evaluated by generating a decision problem that explicitly relates the resource requirements of a given on-line decision-making solution with resource limitations of the target execution system. Once we understand both the resource limitations of a time-critical control system and the predictable temporal patterns of situations that are possible in the interaction of control system and environment, we can compare the former resources to the latter requirements.

In Chapter 4 we show that these design-time comparisons lead to difficult combinatorial optimization problems. For example, in order to determine whether or not the paging architecture of Figure 1.2 can provide an on-line decision-making solution for a particular environment (as represented by the predictable temporal patterns of Figure 1.4), it is necessary to show that the architecture can provide adequate storage and time-critical retrieval of fast decision-making circuits that handle all possible sets of situations in R. Furthermore, the architecture must provide a paging strategy based on the set of situations F^n. The former *minimal set covering* condition is depicted in Figure 1.5. In this figure, a collection of decision-making circuits is compared to a collection of reachable sets of situations, based on the domains of the circuits.

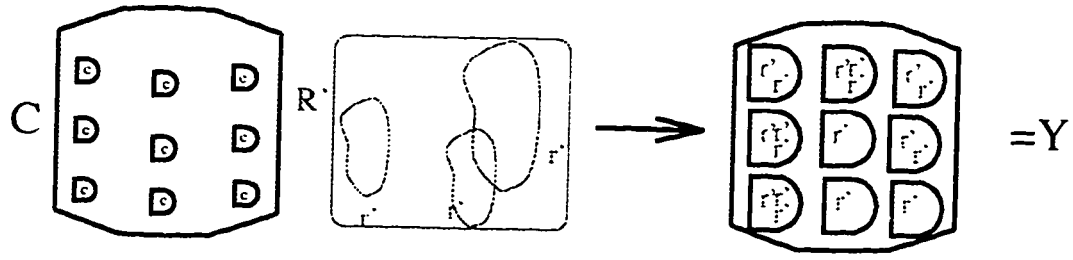

Figure 1.5: Design-time tradeoffs may lead to minimal set covering problems

Analytical techniques for evaluating and solving decision problems

In order to evaluate and solve the decision problems that capture the design-time tradeoffs of on-line decision-making solutions, in Chapters 4 and 6 we develop analytical techniques based on standard computer science techniques. We partition the techniques into those that look at the combinatorial issues of the decision problem and those used to provide a precise model of the relationships between all time and space resources and requirements. These partitions are then combined in order to develop necessary and sufficient conditions for on-line decision-making solutions that both provide guaranteed behavior and do so within limited time and space bounds.

These techniques include complexity analyses, approximation algorithms, and systems of equations. Algorithms to find feasible solutions are combined with proofs of infeasible conditions, equations to evaluate resource tradeoffs, and combinatorial optimization enumeration schemes in order to evaluate the decision problems. As a result, we are able to either show that an on-line decision-making solution cannot be constructed from a particular time-critical control system under a particular uncertain environment, or provide a solution that satisfies the necessary and sufficient conditions of the decision problem. For example, Figure 1.6 depicts an on-line decision-making solution in which a covering of reachable sets of situations by decision-making circuits can be provided within the resource limitations of the paging architecture of Figure 1.2.

Demonstrating the utility of the framework by developing a new approach to designing real-time avionics solutions

The utility of the framework developed in this thesis is demonstrated in Chapter 5 by its use in both the design and analysis of a new approach to designing real-time

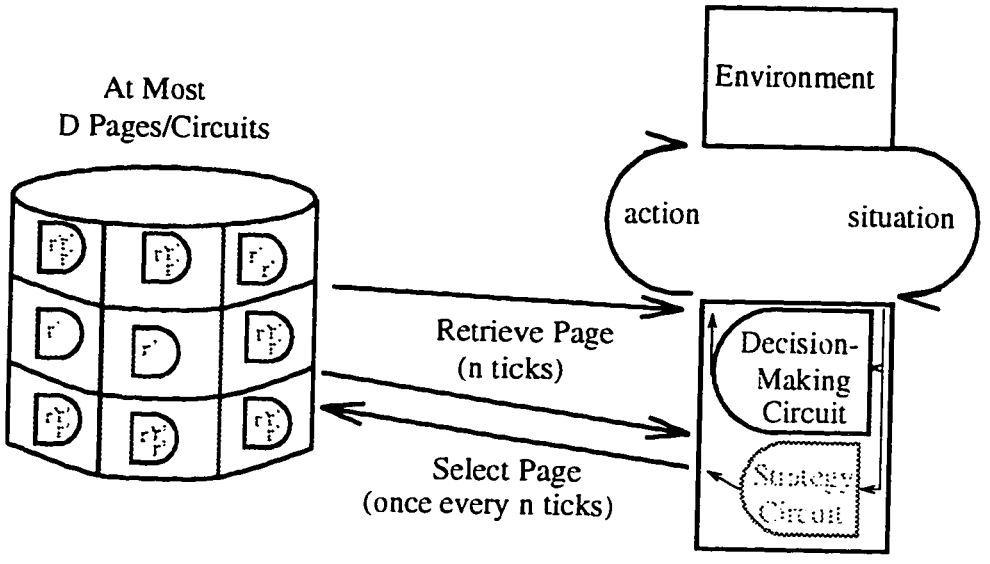

Figure 1.6: Providing an on-line decision-making solution under a two-level paging architecture

avionics solutions. This approach is based on the two-level paging architecture developed in Chapter 3 and analyzed in Chapter 4. Given that this architecture is initially developed as a conceptual abstraction, it is a somewhat surprising result that it may be adapted as a literal solution to real-time avionics problems. In fact, the solution approach provides a new way to pose and solve a variety of general scheduling problems in hard real-time environments under uncertainty. As such, it is a contribution of independent interest.

In this new approach, we use the framework and analytical techniques of this thesis to show how to extend the capabilities and flexibility of existing avionics systems by replacing traditional static scheduling solutions with conditional scheduling solutions, without sacrificing real-time behavior guarantees. Figure 5.3 illustrates example components of a real-time avionics system that accommodates conditional schedules. Components common to existing distributed multiprocessor hardware avionics systems are augmented with additional resources to sense in-flight conditions and conditionally modify execution schedules for flight operations depending upon the sensed situation. Figure 1.8 specifically adapts the two-level time-critical control system of Figure 1.2 to fit within existing distributed multiprocessor avionics hardware.

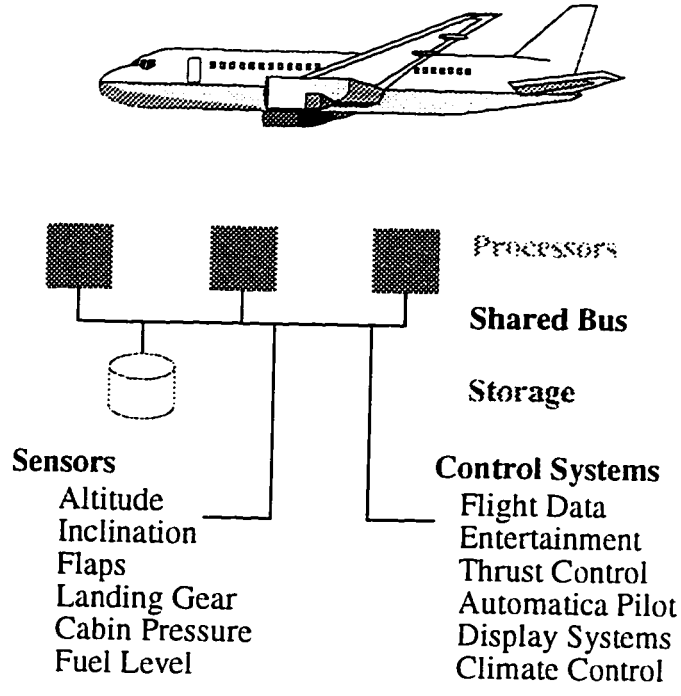

Figure 1.7: Example conditional real-time avionics system components

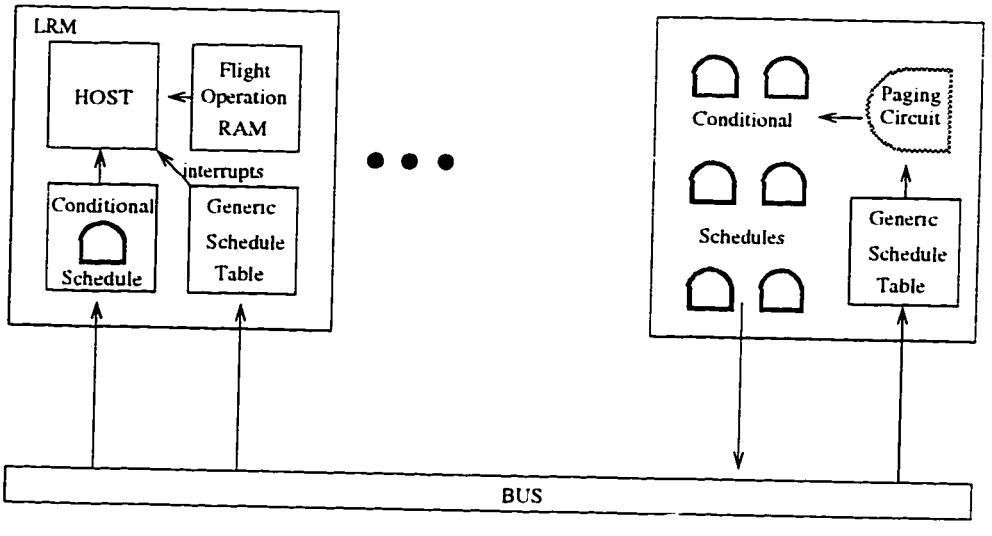

Figure 1.8: Distributed multiprocessor avionics hardware augmented with a schedule server LRM

The underlying premise of this approach is that flight operation sampling frequency requirements may be re-expressed relative to current flight conditions. Sensing of on-line situations can then be employed to determine the actual frequency requirements in each cycle of a given execution. Conditional schedules allow more effective use of limited resources by taking advantage of situation-specific conditional relationships between flight operations.

Empirical evaluation of analytical techniques with respect to real-time avionics

Given that the new approach to real-time avionics scheduling developed in this thesis looks at avionics problems differently than traditional solution approaches, there are no existing models of the environment and resources consistent with those necessary to adapt the analytical techniques of Chapters 3 and 4. Thus, in Chapter 6, we develop restrictions on the general approach of Chapter 5 in order to develop an empirical evaluation of the framework and techniques of this thesis. This evaluation provides a demonstration of the use of the framework of this thesis toward developing time-critical control systems using models specific to a given problem.

This empirical evaluation has two goals. Primarily, we want to understand the practicality of the techniques of this thesis: in terms of both the practical computational efficiency of design-time techniques, and the ability to attain positive results (executable solutions). Secondarily, we want to further validate our new approach to designing real-time avionics solutions by demonstrating scheduling solutions that provide a more flexible and effective use of limited resources.

The restrictions of this study lead to simplified resource equations that allow more direct combination of combinatorial techniques and resource tradeoffs, thus allowing the empirical study to make use of a standard enumeration scheme.

In Chapter 7 we present the results of this empirical study. The empirical results demonstrate that the framework and techniques of this thesis may be practically applied to the analysis and design of on-line decision-making solutions.

Furthermore, they demonstrate that positive results (executable solutions) are attainable over a non-trivial fraction of test cases.

Relation to existing literature

The techniques and applications of this thesis are related to work in many fields, including control theory, decision theory, operations research, systems engineering, static analysis, and real-time scheduling. In Chapter 8, we analyze the contributions of this dissertation in the context of existing literature. Most directly, this thesis is related to work in artificial intelligence that attempts to understand the tradeoffs involved in resource-bounded problem solving, and work on designing on-line decision-making systems to solve problems in specific domains.

Theoretical work on resource-bounded problem solving include development of models of rationality that take deliberation costs into account [30], development of flexible algorithms that adjust to varying resource limitations [15, 7, 42, 65], and development of techniques to provide design-time optimization of on-line solution quality [15, 7, 42, 61, 60, 59, 70]. Furthermore, there has been work on exploiting known structure at design time [8, 18, 53].

Systems have been designed that rely on fast on-demand reaction [1, 9], design-time pre-compilation of fast reaction circuits for all possible scenarios [62, 55], or design-time pre-compilation of circuits with behavior-driven timing guarantees [48]. Some systems provide on-line deliberation by relying on environment locality assumptions [17] or the ability to guarantee control over uncertainty [52]. Many systems have been designed that provide heuristic hybrid combinations of fast reaction and slow deliberation [26, 66].

This thesis provides a framework for evaluating the completeness and correctness of on-line decision-making solutions over a wide range of application domains. This framework provides explicit models of tradeoffs that are implicitly embedded in many decision-making solutions. We provide the unique contribution of being able to deal with sequences of actions that are interdependent, in the face of exogenous uncertainty, by making design-time use of dynamic models of the

environment and time-critical control formulations of the decision-making system. We capture predictable temporal patterns in the interaction of system and environment and use these patterns to evaluate behavior guarantees under all possible on-line executions. We provide an integration of slow deliberation and fast reaction that does not rely on open-loop control or low-complexity deliberations, yet is amenable to design-time analysis. Furthermore, we provide specific behavior guarantees, rather than expected quality measures, that are suitable for hard real-time applications. These guarantees do not rely on assumptions of complete control over uncertainty. Finally, the framework provides a clear path to the analysis and design of more complicated on-line decision-making solutions.

Future work

In Chapter 9 we evaluate the limitations of the techniques developed in this thesis and propose directions for future work to address these limitations. An important contribution of the framework developed in this dissertation is that, by explicitly modeling tradeoffs as time-critical control systems, we provide a method for evaluating the limitations of a particular on-line decision-making solution.

The two-level paging architecture explored in this thesis is a particular form of time-critical control system in which decision-making circuits are paged in a closed-loop manner while the paging circuit is fixed at design time. Furthermore, paging intervals are of fixed duration, and decision-making circuits are independent across intervals. Extensions to this architecture include paging multiple paging circuits in a closed- or open-loop manner, allowing paging intervals to have variable or indeterminate duration, and extending the model of decision-making circuits and paging to allow persistence and dependence across retrieval intervals. Each combination of these architectural extensions captures a different set of tradeoffs and may require new analytical techniques.

Furthermore, the analytical techniques of this thesis assume that decision-making circuits have been provided with pre-specified domains and resource requirements. We might step back and consider the simultaneous construction of

decision-making circuits and time-critical control architectures. and modeling of the dynamics of the interaction of decision-making circuits and the environment. Such construction could take tradeoffs into account in understanding achievable behaviors with respect to iteratively more accurate dynamical models.

Chapter 2

A Formalization of On-line Decision Making

In this chapter, we present a formalization of on-line decision making to be used throughout this dissertation. This formalization is consistent with the goal of expressing on-line decision-making problems using the conceptual abstraction of hierarchical storage.

In its most general form, on-line decision making consists of the interaction of an execution system and environment over time. This generic interaction is depicted in Figure 2.1. In this figure, the environment outputs situations and the execution system outputs actions. The intention of this chapter is to further specify these components and the resulting interaction.

In this thesis, an *on-line decision-making system* is formulated as a time-critical control system that implements the execution system of Figure 2.1. The time-critical control system makes specific time/space tradeoffs in order to execute within a given execution hardware. The behavior of the resulting on-line decision-making solution may then be evaluated with respect to the environment with which it interacts.

In Section 2.1 we provide a formalization of decision-making environments as discrete-time, discrete-space dynamical systems. In Section 2.2 we introduce the

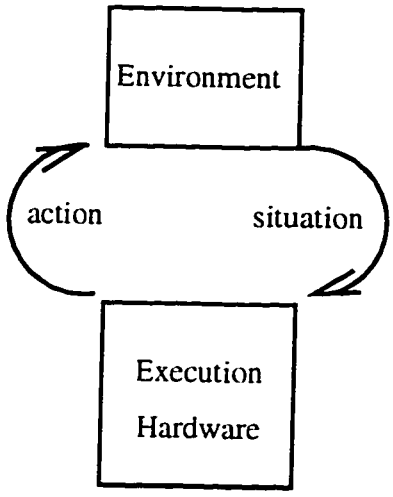

Figure 2.1: On-line decision-making interaction

generic notion of *circuits* to formally capture both fast reaction and slow deliberation. An on-line decision-making system, formulated as a time-critical control system, specifies the form and combinations of these decision-making and strategy circuits, respectively. This specification determines the time/space tradeoffs of the solution. We motivate how these circuits may be used to capture a wide range of on-line decision-making systems that combine reactive and deliberative computations.

Section 2.3 defines the notion of *decision-making behavior* that allows us to evaluate the interaction of a specific decision-making environment and time-critical control system over time. Finally, in Section 2.4, we discuss the explicit properties of the class of problems for which the techniques of this thesis are designed.

2.1 Specifying a Decision-Making Environment

A decision-making environment may be captured by a *discrete-time, discrete-state dynamical system.*

Definition 2.1 *A* discrete-time, discrete-state dynamical system *consists of a set of discrete-time* ticks, $T = \{0, 1, 2, \ldots\}$, *and an automaton,* $G = (\mathcal{A}, X, f, S, y, X_0)$.

where

$\mathcal{A}$ *is a finite set of inputs (actions, including the null action).*

X *is a finite set of states.*

f *is a state transition (partial) function* $f : \mathcal{A} \times X \times T \rightarrow X$ *describing the dynamics of the system.*

$\mathcal{S}$ *is a finite set of outputs.*

g *is an output function* $g : X \times T \rightarrow \mathcal{S}$. *and*

$X_0 \subseteq X$ *is a finite set of possible initial states.*

Note that f can be extended to nondeterministic transitions as well.

A dynamical system is *stationary* if the underlying transition and output functions are independent of the time tick, *i.e.* $f(a, x, t) = f(a, x, t')$ and $g(x, t) = g(x, t'), \forall t, t'$. A transition function is *Markov* (of order one) if the state at time tick t depends solely on the state at time tick $t - 1$ (and any input at time tick $t - 1$).

In general, the states of a dynamical system are not directly observable to the decision-making system. The outputs $\mathcal{S}$ compose the *observable* aspects of the underlying dynamical system. In designing a decision-making solution, we are interested in the properties of the process governing these observable *situations*. This process is referred to as a *decision-making environment*.

Figure 2.2 depicts the components of a decision-making environment. A decision-making system affects this environment through the production of actions in response to observed situations.

2.2 A Circuit Composition Formalization for Time-Critical Control System Design

A formalization of on-line decision making must be general enough to capture a wide range of problems and solutions, yet specific enough so that the time/space

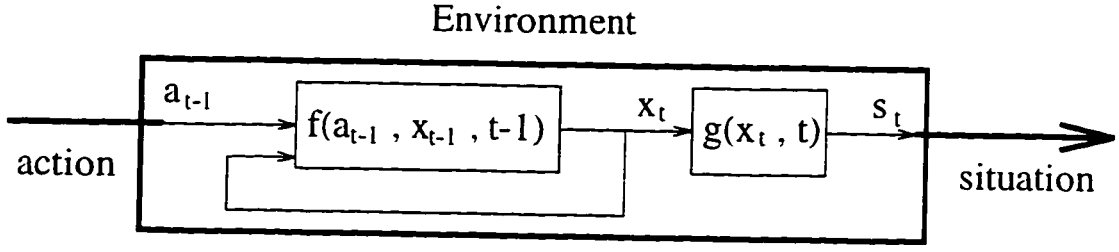

Figure 2.2: A decision-making environment described by a discrete-time, discrete-state dynamical system

tradeoffs in the corresponding time-critical control system can be easily identified and evaluated. Furthermore, to make use of conceptual abstractions based on hierarchical storage, a formalization must capture both slow deliberative computations and fast reactive computations, and the combined behavior resulting from the interactions of multiple levels of decision making.

To accomplish these representation and analysis goals, we formalize a time-critical control system as a composition of *circuits*. In this context, a circuit is a unit of computation with well-specified time and space requirements and input/output behavior. A circuit can capture a variety of traditional decision-making structures, *e.g.* lookup tables, decision-trees, or deliberative planning algorithms. Once these structures are modeled as circuits, they can be consistently compared without regard to their original design conceptions. Furthermore, these quantifications can be used at design time to reason about on-line decision-making behavior. Computational tradeoffs are embedded in the resource requirements of the circuits.

A circuit requires a fixed amount of space resources to implement (or simulate) in hardware and has a circuit delay describing the worst-case amount of time between the introduction of inputs and the generation of outputs. A time-critical control system is composed of multiple circuits whose resource requirements may be considered individually or over subsets of the system. An individual circuit is *executable* only if its space and time resource requirements are consistent with the limitations of the execution hardware.

In this dissertation we are concerned with developing time-critical control systems composed of two specific types of circuits. These circuits capture fast

reactive computations and slow deliberative computations, respectively:

1. a *decision-making circuit* interacts directly with the decision-making environment in order to produce actions in response to situations, and

2. a *strategy circuit* captures the on-line components of the meta-decision-making functionality of a decision-making solution.

Definition 2.2 *A* decision-making circuit *is a (partial) function from situations and times to actions*[1] $\pi : \mathcal{S} \times T \rightarrow \mathcal{A}$.

Definition 2.2 can be generalized to account for histories of situations or restricted to ignore time. The execution hardware of a decision-making solution connects the outputs of a decision-making circuit to the inputs of the decision-making environment and vice-versa. We typically assume that an active (executing) decision-making circuit is synchronized with the decision-making environment at discrete-time ticks. The granularity, h, of a time tick is a property of a given environment and execution hardware. Under this assumption, a decision-making circuit may take no longer than h time (one tick) to generate a response to any situation.

Definition 2.3 *A* strategy circuit *is a (partial) function* γ *from situations and times to* meta-decision-making actions.

A definition of meta-decision-making actions is specific to a particular time-critical control system. Strategy circuits may capture a wide-range of meta-decision-making actions, ranging from the individual open-loop retrieval of pre-compiled decision-making circuits to the closed-loop generation and sequencing of collections of custom-designed decision-making circuits. A strategy circuit implements the on-line execution of design-time tradeoffs with respect to a specific on-line decision-making solution. As with Definition 2.2, Definition 2.3 may be generalized to account for histories of situations or restricted to ignore time. The

[1] It is straightforward to extend this definition to nondeterministic decision procedures.

execution hardware of a decision-making solution connects outputs of a decision-making environment to inputs of a strategy circuit.

A time-critical control system may be composed of multiple decision-making circuits, each intended to execute at different points in time. The scheduling and sequencing of decision-making circuits may be determined partially at design time and partially at execution time, conditioned on situations of the decision-making environment. One or more strategy circuits may determine the scheduling and sequencing of decision-making circuits in a given time-critical control system. Additionally, the scheduling and sequencing of strategy circuits may be determined by the strategy circuits themselves.

In a given on-line decision-making solution, the currently active strategy circuit may execute concurrently with the currently active decision-making circuit. However, any concurrency among strategy circuits and/or decision-making circuits must be considered at design time, in terms of on-line computational requirements and limitations.

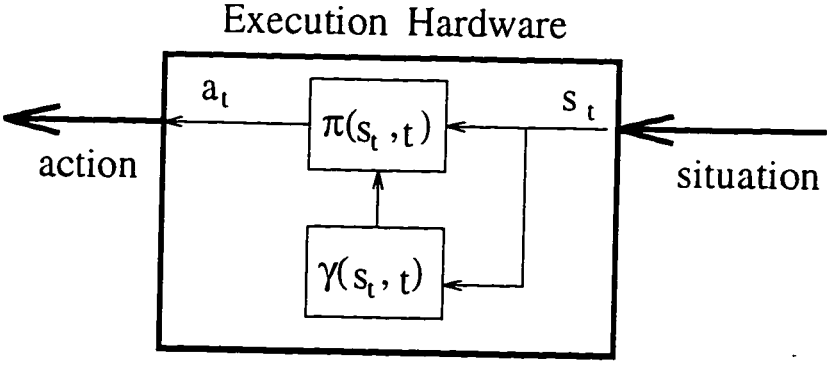

Figure 2.3: An execution hardware composed of a decision-making circuit π and a strategy circuit γ

Figure 2.3 depicts an example time-critical control system composed of a strategy circuit γ and decision-making circuit π. The two circuits are concurrently active, with γ modifying π in a closed-loop manner (*i.e.* in response to situations of the decision-making environment). Outputs of the combined control system are determined, at each time tick, by the decision-making circuit π. The form of meta-decision-making action by which γ modifies π is not specified in this

example. In Chapter 3 we develop a specific time-critical control system in which these interactions are fully specified.

2.3 Decision-Making Behavior

During any execution of a decision-making solution, the behavior of the combined decision-making environment and time-critical control system can be captured by a sequence of situations and actions. Let H describe the *horizon* of a decision-making execution, in terms of time ticks. Unless otherwise indicated, we assume $H = \infty$.

Definition 2.4 *$e = (p_0, p_1, p_2, \ldots, p_H)$ is an* execution trajectory, *such that each $p_i = (s_i, a_i), s_i \in \mathcal{S}, a_i \in \mathcal{A}$ corresponds to the situation s_i generated by the decision-making environment at time tick i and the corresponding decision-making action a_i (including the null action) generated by the decision-making system. Furthermore, $s_0 \in S^0$ for some initial set of situations $S^0 \subseteq \mathcal{S}$.*

Figure 2.4 depicts an example execution trajectory. At each time tick, an action is generated by the time-critical control system in response to the current situation (as indicated by the solid arrows). The decision-making environment then determines the situation for the next time tick (as indicated by the dashed arrows).

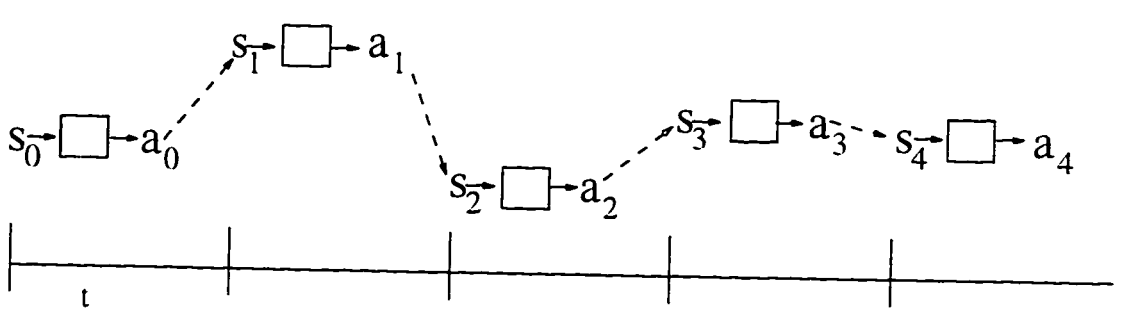

Figure 2.4: An example execution trajectory

Given nondeterminism in either the decision-making environment, time-critical control system or both, the interaction of environment and control system can be captured by a set of possible execution trajectories.

Definition 2.5 *A* decision-making behavior *is the set of all possible execution trajectories, E, that can result from executing a given decision-making solution under a specific decision-making environment.*[2]

Figure 2.5 depicts an example decision-making behavior. In this figure, at each time tick, the decision-making environment generates one of multiple possible situations. For each situation, or set of situations, at a given time tick, the time-critical control system may have multiple possible decision-making circuits that could be executing. Thus, in general, the response to each set of possible situations is a set of possible actions. In any given execution, a single execution trajectory is achieved, as depicted in the lower half of Figure 2.5.

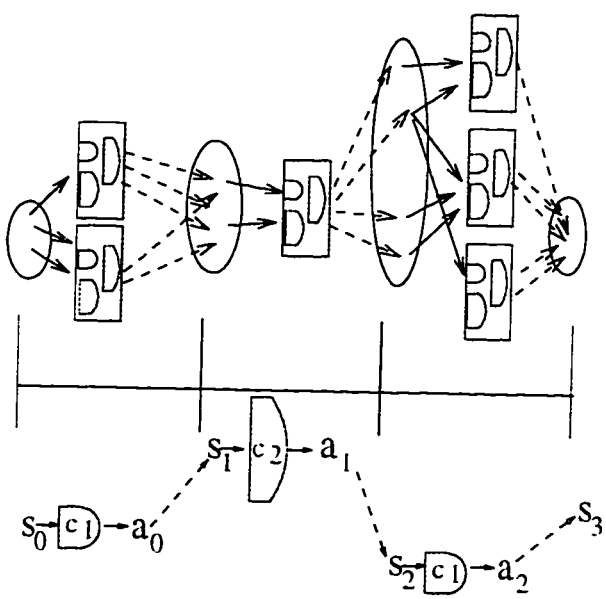

Figure 2.5: An example decision-making behavior

Depending upon problem-specific goals, an execution trajectory may be assigned a measure of quality and/or probability. Furthermore, we may assign a measure of quality to a set of trajectories E. Decision-making solutions can then be compared based on the corresponding decision-making behaviors.

[2]Note that, in the literature of Markov decision processes [57], an execution trajectory is equivalent to a *sample path* and a decision-making behavior is equivalent to a *sample space*.

A decision-making behavior E is *achievable* with respect to a decision-making environment if and only if there exists some decision-making solution to implement it. We define achievability more formally in Chapter 3. Achievability of decision-making behaviors is closely related to *controllability* in embedded control systems [19]. In particular, achievability abstracts out the details of control laws and control objectives (*e.g.* servo, set-point regulation) and allows us to work directly with the resulting set of possible trajectories. Note that the achievable decision-making behaviors of a decision-making solution are directly related to the choice of time-critical control system with respect to the form and interaction of decision-making circuits and strategy circuits.

The ability to model a given decision-making behavior, E, is an especially important component of design-time reasoning. Such modeling requires calculating or estimating, at design time, the possible trajectories that may result from on-line interaction of decision-making system and decision-making environment. Thus, the design-time models available to perform these evaluations are an important component of the analysis and design framework of this dissertation. The difficulty of modeling such a set, E, depends upon the combined properties of the given decision-making environment and decision-making system.

This characterization of decision-making behavior provides a comprehensive picture of the on-line interaction of decision-making solution and decision-making environment. The ability to evaluate such behavior at design time is a powerful tool and one of the primary contributions of this work.

2.4 Discussion of Properties of Difficult Classes of On-line Decision-Making Problems

The thesis is primarily concerned with understanding how design-time tradeoffs impact achievable decision-making behavior under difficult combinations of problem properties, such as those found in time-critical planning and scheduling under uncertainty. Namely, problems with environments that are large-scale, dynamic

and uncertain, and for which time-critical responses using high-complexity computations are required despite being restricted to limited on-line computational resources.

It may be possible to avoid formally quantifying the design choices of an on-line decision-making solution and, yet, attain acceptable decision-making behavior in the target domain. However, such problems must possess simplifying properties: whether or not these properties are explicitly noted. Once these properties are made explicit, however, the extendibility of a decision-making approach can be evaluated. In the following, we introduce classes of properties of decision-making problems and briefly discuss how they affect solution approaches.

Dynamic environments A dynamic environment is one in which exogenous events may occur that are not the direct result of actions taken by the decision-making system. The nature of the dynamics has direct impact on the ability of the decision-making solution to enforce behaviors on the environment. In a dynamic environment, change due solely to the passing of time may have an adverse impact on behavior.

This contrasts with a static environment in which change cannot occur due solely to the passing of time. All change is a direct result of decision-making actions. This does not mean that all actions have deterministic effects. One important simplification of assuming a static environment is that decision-making delay has no effect on future interactions.

Uncertainty A deterministic decision-making environment allows the effects of actions (including the passing of time) to be completely determined at design time. This does not imply that it is always computationally feasible to do so. Uncertainty, on the other hand, introduces the need to contend with various possible futures. Uncertainty may arise in the exogenous events of the environment, or in the effects of the decision-making actions themselves. Furthermore, uncertainty may exist in the computation of decision-making actions.

The number of possible situations that must be considered at design time is a function of both the amount of uncertainty and the dynamic rate of change

of an environment. Nevertheless, designing solutions to a deterministic decision-making problem can retain significant difficulty, *e.g.* scheduling of limited computational resources.

Time-criticality A problem is time-critical if there is a cost for delay in taking action. The phrase *real-time* is often used to describe time-critical problems. *Hard* real-time problems contain strict deadlines on taking action while *soft* real-time problems allow actions to take place after a deadline but, with a time cost.

The form of time-criticality in a given problem impacts the on-line decision-making issues. At one extreme, the rate of change may be slow enough, or on-line decision making fast enough, to always provide time-critical reaction with on-demand reasoning. At the other extreme, the environment may change too rapidly for any non-trivial on-line decision making to take place.

Large-scale problems Small-scale problems simplify on-line decision making by allowing complete pre-compilation of actions at design time and/or rapid selection of actions in response to on-line situations. Large-scale problems impact the computational complexity of both on-line decision making and design-time solution generation.

Limited computational resources The relationship between decision-making computational complexity and the availability of on-line computational resources fundamentally impacts achievable decision-making behavior. Limited on-line computational resources motivate techniques that force space- and time-consuming decision making to be performed at design time. Limited design-time computational resources may also be considered.

High-complexity decision procedures Low-complexity decision procedures can compute decision-making actions rapidly for any situation. Decision-making behavior that can be adequately computed with low-complexity decision procedures (such as those of greedy reactive approaches) is simple to implement in

all but the most extreme decision-making environments. High-complexity decision procedures require large time and/or space resources in order to compute some or all decision-making actions. However, high-complexity decision procedures executed at design time can lead to low-complexity on-line decision-making solutions.

Chapter 3

Solving On-line Decision-Making Problems by Sequencing Circuits

The process of constructing a time-critical control system to solve an on-line decision-making problem requires understanding the decision-making environment, the limitations of the execution hardware, the space of decision-making circuits and strategy circuits that may be employed to compose the control system, and the decision-making behavior that results from the interaction of time-critical control system and decision-making environment. This dissertation makes several simplifying assumptions in order to provide an initial understanding of this general approach to solving on-line decision-making problems.

In Section 3.1 we provide a hypothetical on-line decision-making problem and illustrate the steps that might be taken in generating a time-critical control system to solve the problem. In Section 3.2 we discuss these steps and introduce a *decoupling* assumption to better focus our exploration of the time/space tradeoffs of designing time-critical control systems. The decoupling allows us to assume that a collection of decision-making circuits is provided as input to our process of designing a time-critical control system.

Given a pre-compiled collection of decision-making circuits, the time-critical

control system design problem becomes one of figuring out how to *sequence* these circuits such that a desirable combined control behavior is achieved. We would like to provide an execution architecture and one or more strategy circuits such that the resulting time-critical control system achieves the desired behavior. In Section 3.3 we provide a more precise understanding of what is required to construct a time-critical control system that achieves a desired behavior by sequencing decision-making circuits. In particular, we develop a special class of decision-making behaviors and demonstrate sufficient conditions under which any time-critical control system can implement a behavior in this class.

In Section 3.4 we introduce a class of time-critical control systems based on a two-level hierarchical storage system. This class of *circuit paging* architectures solves on-line decision-making problems by paging decision-making circuits in response to situations of the decision-making environment.

Finally, in Section 3.5 we characterize the predictable temporal patterns of situations that are relevant for evaluating the behaviors achievable under circuit paging architectures. These patterns form the basis for developing analytical techniques to evaluate and solve circuit paging problems in Chapter 4.

3.1 An On-line Decision-Making Solution Design Scenario

As Director of Northeast Corridor Operations for Amtrak you've been put in a difficult situation. Due to an increase in tragic collisions and other mishaps, the National Transportation Safety Board is considering more stringent regulations on railway operations. Already operating at a deficit, these guidelines will surely remove any chance of returning to profitable, or at least break-even, operations.

You are convinced that these added regulations are unnecessary. Sophisticated sensing equipment exists throughout the system that could provide adequate information to guarantee safe and effective locomotive control under existing regulations. The problem, you conclude, is the lack of an effective control system to adjust the speed of the locomotives in response to changing traffic and weather

patterns.

You commission a team of mathematicians and control theorists to study the railway system and devise a controller to guarantee safe operation under existing regulations. With the cooperation of local commuter train services throughout the Northeast, the team develops a stochastic model of railway traffic. With further input from the National Weather Service and railway engineers, the team incorporates into the model various weather conditions and their effects on railway operations.

Armed with these models, the team develops a mathematical formulation for the transportation control system that provides a three-fold increase in railway system throughput, while simultaneously reducing travel delays. As an added benefit, the new system provides a reduced probability of tragic collisions and mishaps: lower even than the Safety Board had hoped for with their new regulations.

Triumphantly, you contract a team of software engineers to implement the control system. A fews days later the head software engineer informs you that the control system cannot be implemented. The theoretical control solution entails millions of variables and high-complexity algorithms yet requires computing a new set of responses once a minute, as traffic situations change. Even under the most optimistic hardware assumptions, each response would require hours to compute.

However, the engineer continues, it may be possible to trade time for space and pre-compute some components of the control system. A short time later the head engineer returns with mixed news. Through skillful tradeoffs they were able to pre-compute enough calculations to meet the responsiveness requirements. Unfortunately, the resulting solution requires several terabytes of expensive cache memory to implement.

Discouraged but still hopeful that the control system can be salvaged, you present the software engineer's analysis to the head mathematician. The mathematician points out that there has been no attempt to look closely at the problem

and decompose the formulation. In particular, although the control system is designed to handle all possible situations that may arise during railway operation, not all situations can occur at any given time of day or day of the year. Rush hour traffic situations never occur at night and ice storms are very unlikely in July. Furthermore, with a little lead time, the stochastic model can be used to anticipate patterns of situations over time. The mathematician provides you with an abstraction to capture patterns of situations over time, and then moves on to another problem.

You pose the decomposition suggestion to the software engineer. The software engineering team attempts to model the time and space requirements of the control system with respect to subsets of possible situations. They return with a disk filled with a thousand different control systems. Each one, they claim, corresponds to a different subset of situations. Each one is implemented as a circuit guaranteed to calculate the responses specified by the original mathematical formulation of the control system, and do so within a minute. Furthermore, each circuit is small enough to fit into the available cache of the traffic control system hardware with some room to spare. After exhaustive study, they have determined that these are the only subsets of situations for which such "executable" circuits could be derived. They, too, move on to other problems.

They leave it to you to figure out whether or not the given collection of circuits can be used as part of a traffic control system operating within the environment described by the stochastic model. You stare blankly at your desk and hope the rest of this chapter can help.

3.2 Decoupling Decision-Making Behavior and Time-Critical Control System Design

An on-line decision-making solution specifies the interactions between a decision-making system and a decision-making environment such that the system chooses actions in response to situations in the environment. These actions, in turn, may affect future situations. A decision-making behavior, E, provides a comprehensive

picture of this interaction.

In general, the design of a time-critical control system cannot be considered independently of the desired behavior of the interaction of system and environment. More specifically, choosing an execution architecture and constructing both strategy circuits and decision-making circuits impact the decision-making behaviors that may be achieved. However, circuits are intended to achieve a particular desired behavior. Thus, there is an inherent circularity in designing a time-critical control system to solve an on-line decision-making problem.

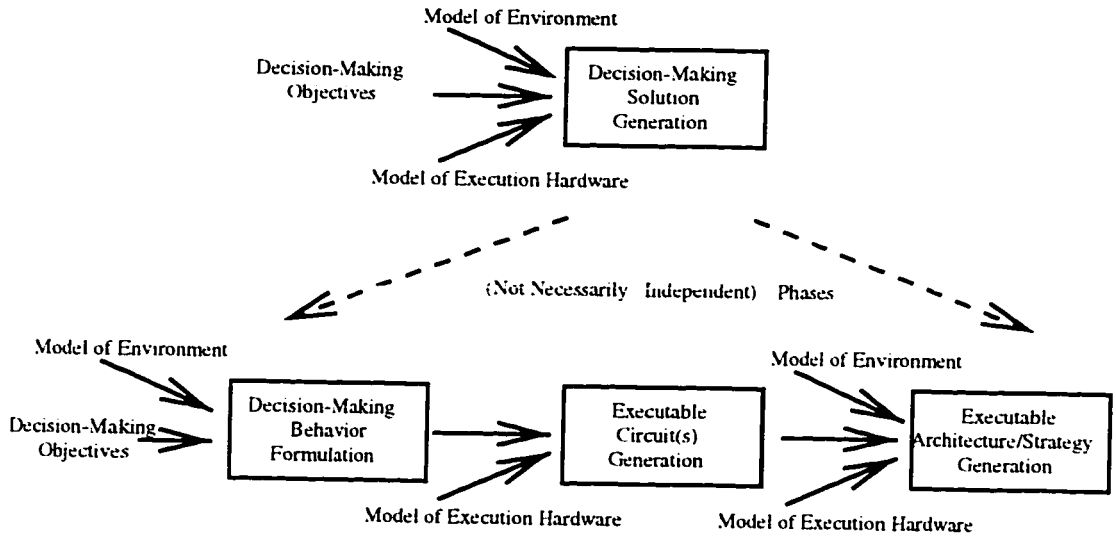

Figure 3.1: An example sequence of steps toward generating a solution to an on-line decision-making problem

To illustrate this point, Figure 3.1 depicts a sequence of steps in generating an on-line decision-making solution. With respect to the traffic control scenario, these three phases correspond to that of the mathematicians, the software engineers, and the Director, respectively from left to right.

In the hypothetical traffic control problem, phases of developing the traffic control solution are treated sequentially, without feedback. The team of mathematicians have no access to the parameters of the execution hardware and, thus, create a decision-making behavior formulation that describes desirable behavior without regard to whether or not it may be implemented. The software engineers have no information about the mathematical formulation that would allow

them to alter the behavior to fit the execution parameters. Their only option is to construct a piecewise decision-making solution, with pieces small enough to be executed, tweaking time and space tradeoffs as needed, and to specify under what assumptions the behavior of each piece is consistent with the mathematical formulation.

The initial step of Figure 3.1 consists of formulating the desired behavior of the solution based on a model of the decision-making environment and some set of decision-making objectives. The second step consists of generating a collection of decision-making circuits to satisfy this behavior formulation, such that the decision-making circuits may be executed within the limitations of the target execution hardware. The final step consists of choosing an execution architecture and constructing one or more strategy circuits to determine the scheduling and sequencing of decision-making circuits.

However, if these three steps are executed sequentially, neither the behavior formulation step nor the circuit generation step could take into consideration the time/space tradeoffs necessary in solving the final step. Thus, there is no reason to believe that the interaction of the resulting time-critical control system and decision-making environment will achieve the desired behavior.

This illustrates that, in general, the formulation of desired behavior and the generation of a time-critical control system to implement that behavior must be an interdependent process. However, the following informal *decoupling assumption* defines a time-critical control system design task in which we separate the final step of this process from the prior steps.

Definition 3.1 *A time-critical control system design problem is* decoupled *if we may assume that a collection of pre-compiled decision-making circuits is provided as input.*

In Section 3.3, we show that, for a particular class of behaviors, assuming a pre-compiled collection of decision-making circuits is equivalent to decoupling decision-making behavior itself. In other words, we show that to implement a decision-making behavior from this class, given a decoupled time-critical control system design problem, we need only design an execution architecture and one or

more strategy circuits for sequencing decision-making circuits that satisfy some well-defined properties.

Decoupling is one way to explicitly simplify the process of generating decision-making solutions. We consider carefully the impact of this simplification on both achievable decision-making behavior and complexity of generating decision-making solutions.

For the purposes of this chapter, it is not relevant that the decision-making behavior formulation phase and circuit generation phase are performed sequentially. In the traffic control problem, for example, the mathematicians and engineers could have gotten together and made tradeoffs in behavior quality to produce the resulting collection of circuits. The relevant point is that they failed to construct a single executable decision-making circuit that attains acceptable decision-making behavior. Therefore, an additional phase is necessary to determine if the resulting collection of circuits can be incrementally pieced back together without changing the decision-making behavior of the formulation.

3.3 Evaluating Decision-Making Behavior that is Achieved by Sequencing a Collection of Pre-compiled Decision-Making Circuits

In Section 3.2, we define a decoupled time-critical control system design problem as one in which a collection of pre-compiled decision-making circuits is provided as input to the problem. The problem then becomes one of determining how to sequence this collection of circuits such that a desired behavior is achieved. In this section, we precisely define how an on-line decision-making problem is solved by sequencing circuits, the class of desired behaviors for which a decoupled approach is applicable, and the combined properties of a time-critical control system and decision-making environment that are sufficient to provide a decoupled solution based on sequencing circuits. Additionally, we discuss some special cases in which these properties are not necessary to attain desired behavior under a decoupled

approach.

Sequencing A sequencing of decision-making circuits describes the process of multiple circuits sharing control of the same hardware, one at a time. The following definition captures the sequencing of decision-making circuits.

Definition 3.2 *Let C be a collection of decision-making circuits. A* sequencing *of decision-making circuits $(\pi_0, \pi_1, \pi_2, \ldots)$ is a mapping of decision-making circuits to time ticks, such that for each time ticks i, $\pi_i = c_S$ for some $c_S \in C$.*

As defined in Chapter 2, a decision-making circuit is a (partial) function from situations and time ticks to actions. In this chapter, we remove the dependence on time and consider the domain V_S of a decision-making circuit c_S to be a subset of the set of all possible situations $\mathcal{S}$. A decision-making circuit c_S is said to *cover* each $s \in V_S$. The worst-case execution time and space requirements of a circuit are well-defined with respect to a given execution hardware. In this chapter, we only consider decision-making circuits that take at most one tick to compute.

As defined, the action produced by a decision-making circuit depends only on the current situation of the decision-making environment. We can refer to a decision-making circuit that is a function of a single situation as a *Markov* decision-making circuit. Furthermore, since we have removed any dependence on time in determining actions, we can refer to the circuits as *stationary* decision-making circuits. Nevertheless, since decision-making circuits affect the environment, performing multiple-tick computations by sequencing stationary Markov decision-making circuits is still meaningful.

Figure 3.2 schematically depicts the sequencing of the collection of decision-making circuits $C = \{c_1, c_2, c_3\}$ to compose a decision-making solution. A given on-line execution of the sequencing $(\pi_1 = c_1, \pi_2 = c_2, \pi_3 = c_1, \pi_4 = c_3, \pi_5 = c_3)$ induces an execution trajectory (sequence of situation/action pairs).

Covering An execution trajectory captures an on-line execution of a sequencing of decision-making circuits. Since decision-making circuits can be partial

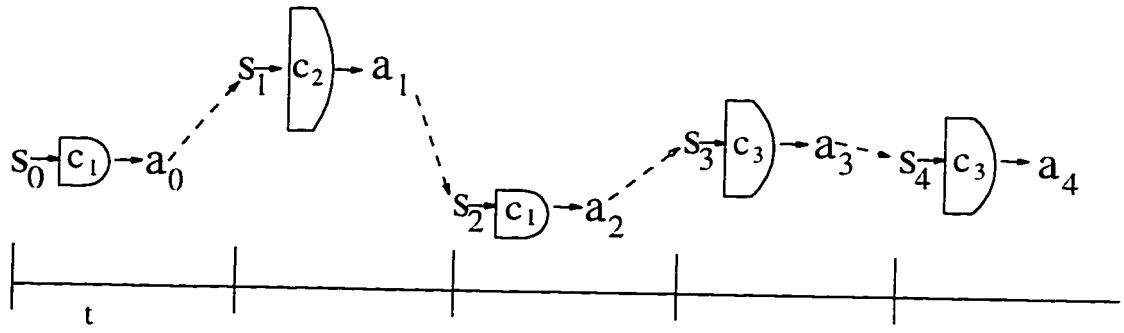

Figure 3.2: An example sequencing of decision-making circuits

functions, an arbitrary sequencing may result in executing a decision-making circuit at a time tick during which it may encounter a situation outside its domain. To establish a decoupling between decision-making behavior and time-critical control system design, it is useful to evaluate whether or not a specific decision-making solution can ever produce such ill-conceived sequencings.

However, evaluating the relationship between sequencings and situations of the environment requires more information about a decision-making solution than we have presented, thus far. In particular, we require information about *when* sequencings are determined, and *what* information about the environment is available in making those determinations. This information may be required at design time or during a particular execution of the decision-making solution.

Suppose we know what information a decision-making solution has available at the time it determines the decision-making circuit π_i to execute at time tick i. And that this information is reliable enough so that the decision-making solution never excludes from consideration the situation that will eventually be observed during time tick i, s_i. The following definition captures this supposition.

Definition 3.3 *Define* $PS^i \subseteq S$ *to be the* possible situations *that a decision-making environment might attain during tick* i *with respect to a particular execution of a decision-making solution. If* $s \notin PS^i$ *then it is guaranteed that* $s_i \neq s$.

Note that $\{PS^i\}_{i\geq 0}$ is defined with respect to a specific decision-making solution and execution and is not necessarily representative of the possible situations that may be generated by a decision-making environment over all possible decision-making solutions and executions.

The following definition establishes a relationship between sequencings and possible situations.

Definition 3.4 *A decision-making circuit $\pi_i = c_S \in C$ covers a set of possible situations PS^i if and only if $PS^i \subseteq V_S$.*

Definitions 3.3 and 3.4 are now used to develop a design-time guarantee about decision-making behavior, under a decoupled decision-making solution.

Desired Behavior Recall that a *decision-making behavior* corresponds to the set of all possible execution trajectories, E, that can result from executing a given decision-making solution under a specific decision-making environment. Further recall that a decision-making behavior E can be evaluated based on problem-specific goals. Define a *desired behavior* to be the set of trajectories E^* that attain an acceptable level of performance for a given problem. The following definition relates decision-making behavior and desired behavior.

Definition 3.5 *A decision-making solution with behavior E implements a desired behavior E^* if and only if $E \subseteq E^*$ and $E \neq \emptyset$.*

Note that Definition 3.5 implies that there is no preference for any given trajectory within a desired behavior E^*.

In general, decision-making actions affect future situations of the decision-making environment. In order to decouple decision-making behavior and solution design, we need to establish a relationship between situation-specific sequencing choices and desired behaviors, such that acceptable decision-making behavior is invariant with respect to equivalent sequencing decisions. We first introduce some additional terminology for describing components of execution trajectories.

Definition 3.6 *Let* $e = (p_0, p_1, p_2, \ldots)$ *be an execution trajectory. Define* e_i^- *to be the* prefix *trajectory of* e *up to and including* p_i. *Furthermore, execution trajectory* $(e; e')$ *is a* composition *of trajectories* e *and* e'.

Using these definitions, we can establish an invariant relationship between collections of decision-making circuits and desired behaviors.

Definition 3.7 *A desired behavior* E^* *is* interchangeable *with respect to a collection of stationary Markov decision-making circuits*[1] C *if and only if, for each* $c_S \in C$ *and* $s_i \in V_S$, *if there exists a trajectory* $e \in E^*$ *such that* $(s_i, a_i) \in e$ *at tick* i *then there exists a trajectory* $e' \in E^*$ *such that* $e_i'^- = (e_{i-1}^-; (s_i, c_S(s_i)))$.

In other words, Definition 3.7 states that, if it is acceptable to have an execution trajectory in which situation s_i occurs at time tick i, then it is acceptable to execute, at time tick i, any stationary Markov decision-making circuit that has situation s_i in its domain. With respect to the literature on Markov decision processes, Definition 3.7 is equivalent to stating that the collection of decision-making circuits collectively describe a (potentially nondeterministic) policy.

As an degenerate example, consider any desired behavior defined by the set of trajectories that might result from applying a deterministic stationary policy to a decision-making environment modeled as a Markov decision process. All trajectories of such a behavior specify a unique (deterministic) action in any situation. The behavior is, therefore, interchangeable with respect to any collection of decision-making circuits each of which (partially) implements the deterministic stationary policy. Note that we can generalize Definition 3.7 to include non-stationary and history-dependent decision-making circuits, as needed.

Figure 3.3 illustrates a desired behavior that is interchangeable with respect to a collection of decision-making circuits. In particular, s_i occurs at time tick i and $s_i \in V_S$ for $c_S \in \{c_1, c_4, c_5, c_6\}$. Therefore, there exists at least one trajectory in E^* that results from sequencing each of these decision-making circuits at time tick i. Note that c_4 leads to the same trajectory as c_1, while c_5 and c_6 do not, yet all must be in E^*.

[1] We can equivalently say that a collection of circuits is interchangeable with respect to a desired behavior.

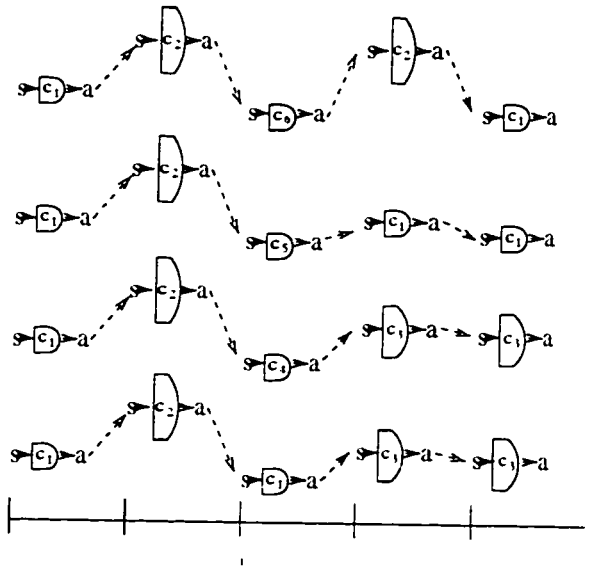

Figure 3.3: An example desired behavior that is interchangeable with respect to a collection of decision-making circuits

Finally, we relate desired behavior to behavior that actually "makes sense" for a given decision-making environment. A desired behavior E^* is *achievable* with respect to a decision-making environment if and only if there exists some decision-making solution to implement it. We can more formally define an achievable desired behavior as follows.

Definition 3.8 *A desired behavior E^* is* achievable *with respect to a decision-making environment if and only if for each $e \in E^*$, if $(s_i, a_i) \in e$ at tick $i \geq 0$, then for each situation $s_{i+1} \in \mathcal{S}$ that can be generated by the decision-making environment after taking action a_i in situation s_i, $(e_i^- : (s_{i+1}, ..))$ is prefix of some $e' \in E^*$; and for each $s_0 \in \mathcal{S}$ that corresponds to an initial situation of the decision-making environment, $(s_0, ..) \in e'$ for some $e' \in E^*$.*

Aside from being achievable with respect to a decision-making environment and interchangeable with respect to a collection of circuits, we don't rely on any other general restrictions on desired behavior. In particular, we don't restrict the measures of quality of long-term behavior. Desired behavior may include only the best possible "optimal" long-term behavior with respect to some objective,

it may include only some class of sub-optimal behavior that induces useful coverage properties, or it may include approximate behavior or heuristic behavior for which no provable relationship to optimal behavior exists. Nondeterministic specifications of desired behavior are explicitly allowed. However, as we see throughout this work, unrestricted desired behavior may not be achievable with respect to a particular execution architecture and hardware.

Decoupling Lemma We can now combine the definitions above to establish a design-time guarantee relating desired behavior and time-critical control systems. The following lemma states that we can decouple decision-making behavior formulation and time-critical control system design as long as we guarantee that our time-critical control system always produces coverings.

Lemma 3.1 *Let desired behavior E^* be interchangeable with respect to the collection of decision-making circuits C. For each time tick i, let $\{PS^i\}_{i\geq 0}$ be the possible situations attainable with respect to a given decision-making solution and execution.*

If desired behavior E^* is achievable under a given decision-making environment and a time-critical control system is guaranteed to only produce sequencings $(\pi_0, \pi_1, \pi_2, \ldots)$ such that each decision-making circuit $\pi_i = c_S \in C$ covers the set of possible situations PS^i, for each tick i, then any decision-making solution based on this control system implements desired behavior E^*.

Lemma 3.1 states that, in order to guarantee an achievable desired behavior, we need only provide a covering of possible situations from a suitable collection of decision-making circuits. Lemma 3.1 provides conditions under which globally dependent decision-making behavior may be accomplished with only local sequencing decisions.

Lemma 3.1 only provides conditions under which a decoupled decision-making solution is guaranteed to produce achievable desired behavior. It does not specify how to find such decision-making solutions, nor how to determine, at design time, the sets of possible situations that must be covered. We present a specific time-critical control system design approach that satisfies the conditions of this lemma

in Section 3.4, and the sets of possible situations that this approach requires in Section 3.5.

Note that Lemma 3.1 only allows us to state that our solution implements the desired behavior if the desired behavior is achievable. If the desired behavior is not achievable, we may generate a decision-making solution that covers the possible sets using a collection of circuits that is interchangeable with respect to the desired behavior, yet does not implement the desired behavior. The resulting decision-making behavior might include trajectories that are not in the desired behavior. To avoid this dilemma, we might require that desired behaviors be shown achievable prior to trying to find decision-making solutions that implement them.

Special Cases Note that the conditions of Lemma 3.1 are sufficient for attaining achievable desired behavior but, not always necessary. It may be possible to attain desired decision-making behavior without guaranteeing a covering of situations in each tick.

For example, consider a decision-making environment such that, in some subset of *action-neutral* situations, the action taken by the time-critical control system does not affect future situations. In these action-neutral situations, local sequencing choices may be made without any consideration of the impact on global behavior. In particular, the local sequencing choice might fail to provide a covering of those situations, without impacting global behavior.

For decision-making environments in which all situations are action-neutral, covering requirements are not directly relevant as measures of solution quality. In these cases we need only provide measures local action quality. The degree of covering only indirectly affects the behavior quality.

We may also consider solving problems for which a weakened form of Lemma 3.1 might apply. For example, consider a decision-making environment for which it is impossible find a decision-making circuit to cover all the possible situations in early time ticks, given only design-time information. We might attempt to sequence decision-making circuits during the initial ticks so that later ticks are easy to cover. And, we might attempt to provide this guarantee at design time.

One way to handle this case is to define a desired behavior for which the initial ticks of any trajectory are treated differently than later ticks. This is a restricted form of *non-stationary* decision-making behavior. In particular, suppose we allow two different behaviors. One corresponding to a desired long-term behavior and a second corresponding to a "driving" behavior. The "driving" behavior is intended to execute initially and drive the system to a point at which we can guarantee complete coverage and provide a sequencing solution to implement the desired behavior. The existence of such a "driving" behavior can be evaluated using classical notions of controllability in discrete event systems (*e.g.* see [19]).

3.4 Circuit Paging

In this section we introduce a class of time-critical control systems based on a two-level hierarchical storage system. These control systems provide an execution architecture and strategy circuit for sequencing a pre-compiled collection of decision-making circuits. We may then evaluate Lemma 3.1 with respect to an instantiation of this architecture and a given decision-making environment. Each *circuit paging* control system contains:

1. a *paging strategy circuit* for specifying the sequencing of decision-making circuits, and

2. a *circuit paging architecture* for implementing the paging solution within the execution hardware.

Paging Strategies The class of strategies we consider in this section are referred to as *paging strategies*. In particular, a paging strategy γ is a (partial) function from situations to decision-making circuits and times, $\gamma : \mathcal{S} \rightarrow C \times T$. A paging strategy implements a sequencing of decision-making circuits, taking advantage of on-line situations as needed. A paging strategy that takes advantage of on-line situations is referred to as *closed-loop*.

The execution component implementing a paging strategy is referred to as a *paging strategy circuit*. Note that, as with decision-making circuits, a paging

circuit can be a partial function. Determining the domain and the worst-case execution time and space requirements of a paging circuit is an important part of providing a decision-making solution. Furthermore, as we shall see, the domain of the paging circuit must itself satisfy covering conditions.

We immediately note a circuit paging control system is a restriction on the general class of time-critical control systems. In particular, paging strategies are defined to produce actions based only on the current situation, as opposed to sequences of situations. We can refer to such strategies as *Markov* strategies. Furthermore, since a paging strategy does not take into account the tick during which it is applied, it may be referred to as a *stationary* strategy. These are restrictions on the general formulation of strategies. In Chapter 9, we consider more general strategies.

Circuit Paging In order to implement a paging strategy we must define a computational architecture under which the strategy is executed. This computational architecture is referred to as a *circuit paging architecture*. A circuit paging architecture specifies the framework for turning design-time generation of decision-making circuits and paging strategy circuits into an on-line decision-making solution on a given execution hardware. A paging architecture is a special case of the execution hardware depicted in Figure 2.3.

The problem of designing a paging strategy circuit to implement a covering under a specific circuit paging architecture is referred to as a *circuit paging* problem. We assume that there exists some desired behavior E^* that is interchangeable with respect to a given collection of decision-making circuits. Given a specific paging architecture, a non-modifiable collection of decision-making circuits with pre-specified resource requirements and domains, a suitable transition model of the decision-making environment, and a model of the execution hardware, the circuit paging problem consists of generating a paging strategy to implement the achievable desired behavior, if such a strategy exists. Furthermore, the paging strategy must be itself implementable as a paging strategy circuit that is *executable* with respect to the time and space limitations of the execution hardware.

Finally, the domain of the paging strategy circuit must satisfy coverage requirements determined by the paging architecture.

A Restricted Class of Circuit Paging Architectures In order to evaluate whether or not a circuit paging architecture may be employed as an on-line decision-making solution, we must be more precise about the components of the architecture and the inherent time/space tradeoffs that they entail. A circuit paging architecture may include:

- a paging device (*e.g.* disk, tape, drum, inexpensive memory) on which to store a collection of decision-making circuits,
- a fast cache of M bits to store/simulate one or more circuits,
- a mechanism for moving circuits into and out of the fast cache of the execution hardware, and
- a way to activate an in-cache circuit so that it is "running" and its inputs and outputs are connected to the decision-making environment.

We further assume that a circuit paging architecture has a mechanism to execute the paging strategy circuit concurrently with the currently executing decision-making circuit.

The specific circuit paging architecture explored in this chapter and Chapter 4 is depicted in Figure 3.4. In this architecture, we assume a read-only storage device capable of storing at most D pages/circuits. We assume that a single closed-loop paging strategy circuit γ to implement the paging strategy is fixed at design time. The fixed paging strategy circuit γ selects a decision-making circuit at *fixed retrieval intervals* of n ticks in duration. Every n ticks, a new decision-making circuit/page is selected in response to the current situation and retrieved from disk to cache. The combined process of circuit selection and retrieval may take no longer than n ticks.

In this restricted circuit paging architecture, the fast cache is partitioned so that the paging strategy circuit, currently executing decision-making circuit and

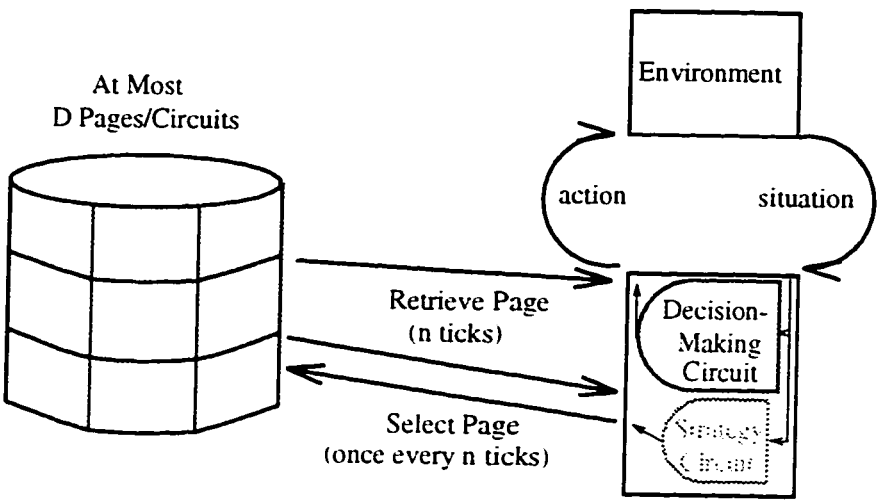

Figure 3.4: A circuit paging architecture

next decision-making circuit may all be stored simultaneously. Let B_γ describe the number of bits of storage required to maintain a strategy circuit in on-line cache. Since the paging strategy circuit is fixed, the remaining $M - B_\gamma$ bits of cache may be used for storing the currently executing decision-making circuit and retrieving the next decision-making circuit, using equal partitions of size $(M - B_\gamma)/2$ each.

An example execution trajectory in which decision-making circuits are retrieved every $n = 3$ ticks, and execute for $n = 3$ consecutive ticks, is depicted in Figure 3.5. Note that this is a restriction of the general sequencing of decision-making circuits of Figure 3.2.

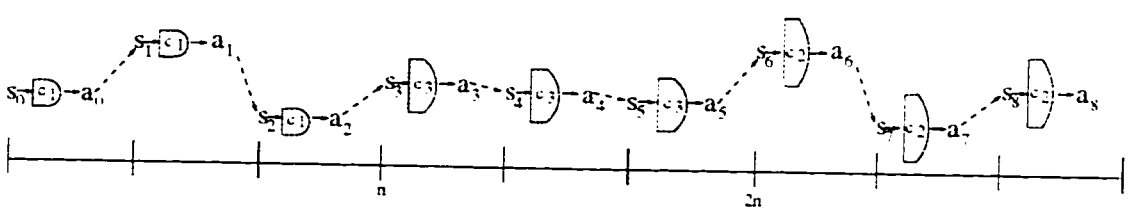

Figure 3.5: An example sequencing of decision-making circuits under a circuit paging architecture of fixed retrieval interval $n = 3$

We refer to the architecture of Figure 3.4 as a *bipartite cache, fixed retrieval interval, circuit paging architecture* (heretofore simply called *circuit paging*). Execution of an on-line decision-making solution under this architecture entails the following steps. An initial decision-making circuit π_0 and the strategy circuit γ are loaded into fast cache prior to decision making. At the start of decision making, the initial decision-making circuit is activated and begins converting situations to actions, once per tick. The initial observed situation s_0 is provided as input to the strategy circuit and decision-making circuit $\pi_n = \gamma(s_0)$ is selected to be paged from disk. π_n is subsequently activated after n ticks. Thereafter, a new decision-making circuit is selected, paged from disk and activated every n ticks.

3.5 Combining a Time-Critical Control System and Decision-Making Environment to Derive Predictable Temporal Patterns of Situations

Lemma 3.1 of Section 3.3 states that, under certain conditions, a decision-making solution consisting of an on-line sequencing of decision-making circuits implements a desired behavior. This lemma relies on the fact that PS^t accurately represents all possible situations of the decision-making environment that can be determined at the time at which the decision-making circuit π_t is selected. The circuit paging architecture of Section 3.4 explicitly specifies the fixed times at which decision-making circuits are selected in a circuit paging solution. In this section, we use this information to derive predictable temporal patterns of situations that must be guaranteed to be covered, at design time, in order to use Lemma 3.1 to show that a decoupled decision-making solution exists that solves an on-line decision-making problem.

The interaction of time-critical control system and decision-making environment may be described by a decision-making behavior E, as defined in Chapter 2. E captures all possible execution trajectories of this interaction. In this chapter,

we are interested in decision-making behaviors that may be represented by a state-transition graph. Such behaviors are both *Markov* and *stationary*.

Definition 3.9 *A decision-making behavior has the* Markov *property if the possible situations at time tick $i+1$ can be determined based solely on the situation at time tick i and any action taken at time tick i. A decision-making behavior is* stationary *if the possible situations subsequent to observing situation s and taking action a are independent of the time tick during which s is observed.*

Note that Definition 3.7 specifies stationary Markov decision-making circuits, but is NOT restricted to stationary Markov decision-making behaviors. In particular, if the decision-making environment is a non-stationary stochastic decision process, the resulting behavior under a given decision-making solution might remain non-stationary even if it is interchangeable with a collection of stationary Markov decision-making circuits.

Despite the applicability of non-stationary and history-dependent stochastic models, in studying circuit paging problems we explicitly restrict attention to stationary Markov decision-making behaviors, represented by state-transition graphs. These models are easier to represent and manipulate, and simplify the presentation of decision-making solutions. For circuit paging problems, the restriction to stationary Markov decision-making behaviors has the added benefit of allowing us to bound the size of sets of possible situations that need to be considered to satisfy Lemma 3.1. However, this benefit is not as important under other time-critical control systems. We re-introduce more general behaviors and transition models in Chapter 9.

Finally, we observe that Lemma 3.1 allows us to consider patterns of possible situations without regard to the actions taken by the on-line decision-making solution. We refer to such a model as *action-independent*. For example, if we assume that the decision-making environment is modeled as a *Markov decision process* (MDP) and the desired behavior E^* is determined by applying a stationary deterministic policy to the MDP, then we can consider the resulting *Markov chain* to capture the stochastic transitions between situations. Note that we are not,

in general, restricted to collections of decision-making circuits that describe a stationary deterministic policy.

The following transition model captures an action-independent stationary Markov decision-making behavior over the set of situations $\mathcal{S}$.

Definition 3.10 *For any situation $s \in \mathcal{S}$, let P_s^k be the set of all $s' \in \mathcal{S}$ such that there is a non-zero probability of being in situation s' exactly k ticks after having started in s. Note that $P_s^0 = \{s\}$ for all $s \in \mathcal{S}$. Also, for any initial set of situations, $P^0 \in 2^{\mathcal{S}}$, let $P^k = \cup_{\{s \in P^0\}} P_s^k$.*

(P_s^k, P^0) defines an action-independent stationary Markov transition model that describes the interaction of a decision-making environment and a decision-making solution. If the decision-making solution is intended to implement a particular desired behavior E^*, we can use this model to evaluate the sets of possible situations that require coverage by paged decision-making circuits. In particular, if the paging strategy requires that PS^i be determined based on the on-line situation observed at time tick $i-k$, then $PS^i = P_{s_{i-k}}^k$. However, we don't know s_{i-k} at design time. Before addressing this issue, we introduce the additional coverage requirement represented by the paging circuit domain.

As specified in Section 3.4, the circuit paging problem, under the architecture of Figure 3.4, requires that a fixed paging strategy circuit be generated at design time. Furthermore, this closed-loop paging circuit selects a new page every n ticks, conditioned on the on-line situation at each selection tick. Therefore, the domain of such a paging circuit must cover all situations that are possible during those selection times, using only design-time information. The following definition characterizes this long-term, unconditional pattern of situations by combining P_s^k over ticks that are a fixed distance apart. An example resulting set is illustrated in Figure 3.6.

Definition 3.11 *For any situation $s \in \mathcal{S}$, let $F_s^n = \cup_{\{j \geq 0\}} P_s^{nj}$ be the set of all possible situations that may occur in exactly nj ticks, for all integers $j \geq 0$, after having started in s. Also, for any initial set of situations $P^0 \in 2^{\mathcal{S}}$, let $F^n = \cup_{\{j \geq 0\}} P^{nj}$.*

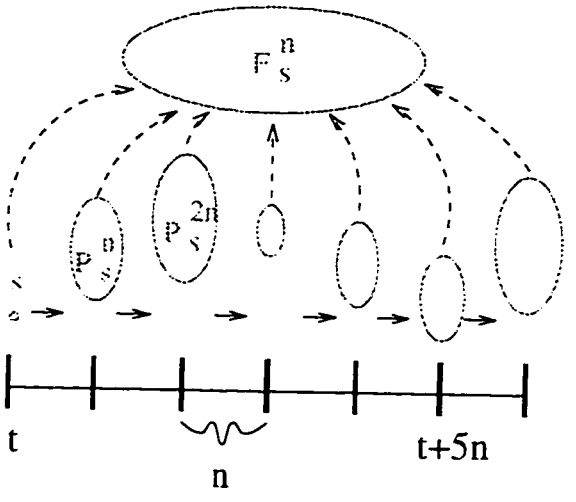

Figure 3.6: Fixed interval paging domain

Although it is constructed from a potentially infinite union, the set F_s^n is finite for any situation s drawn from a finite set of situations $\mathcal{S}$. A fixed paging strategy circuit designed to implement the desired behavior represented by (P_s^k, P^0) is not *well-defined* unless it covers each $s \in F^n$. Note that, a large F^n (*e.g.* $F^n = \mathcal{S}$) does not necessarily imply that the worst-case execution time and space requirements of a paging circuit be large.

As opposed to the paging strategy circuit, the specific coverage requirements that must be satisfied by decision-making circuits in an on-line sequencing are determined during a given execution, conditioned on on-line situations. However, in order to guarantee desired behavior, we must make coverage guarantees at design time.

The paging architecture of Figure 3.4 groups time ticks into intervals. Thus, we must guarantee that our paging strategy can select a decision-making circuit to cover any set of possible situations that might occur during an interval. And this guarantee must be made with respect to the on-line information available at the time tick, prior to the interval, at which a sequencing selection is made.

The following definition characterizes these short-term, conditional patterns of situations, given that a decision-making circuit is selected n ticks before it is executed, conditioned on the on-line situation; and must then execute for n

consecutive ticks. An example resulting set is illustrated in Figure 3.7.

Definition 3.12 *For any situation $s \in \mathcal{S}$, let $R_s^n = \cup_{\{n \leq k < 2n\}} P_s^k$ be the set of all possible situations that may occur between n and $2n$ ticks after having started in s.*

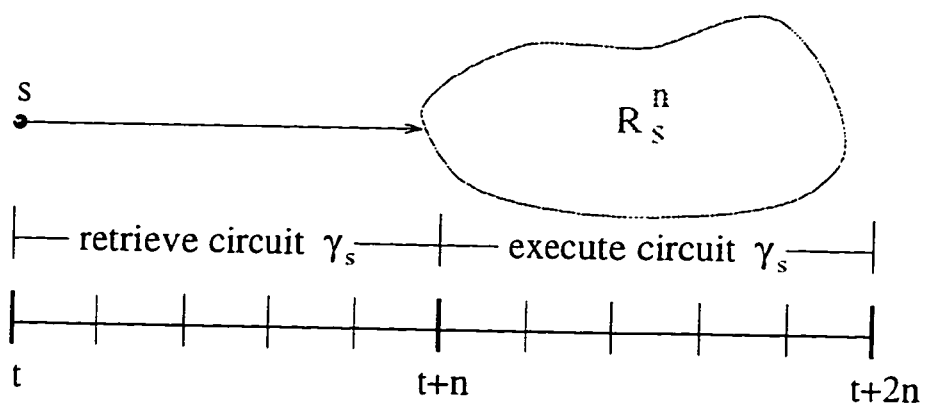

Figure 3.7: Executable circuit coverage requirement

The initial time interval requires separate consideration. In particular, since the initial decision-making circuit must be determined at design time, the coverage requirement of the initial interval must correspond to the set of possible situations that may occur during this interval, with respect to design-time information only. For any initial set of situations $P^0 \in 2^{\mathcal{S}}$, let $R_0^n = \cup_{\{0 \leq i < n\}} P^i$ be the set of all possible situations that may occur prior to time tick n.

For a given circuit paging problem with retrieval interval n, an action-independent stationary Markov decision-making behavior represented by the transition model P_s^k and initial set of situations P^0 can be used with the above definitions to derive the predictable temporal patterns of possible situations of interest. In particular, for a given initial set P^0, we are interested in the set F^n, the set R_0^n, and the sequences $\{P^{nj}\}_{j \geq 0}$ and $\{R_s^n | s \in P^{nj}\}_{j \geq 0}$. Figure 3.8 depicts the relationship between these patterns. In Chapter 4, we show that these are the patterns required to derive the sequences of sets of possible situations $\{PS^i\}_{i \geq 0}$ used to guarantee desired behavior in Lemma 3.1.

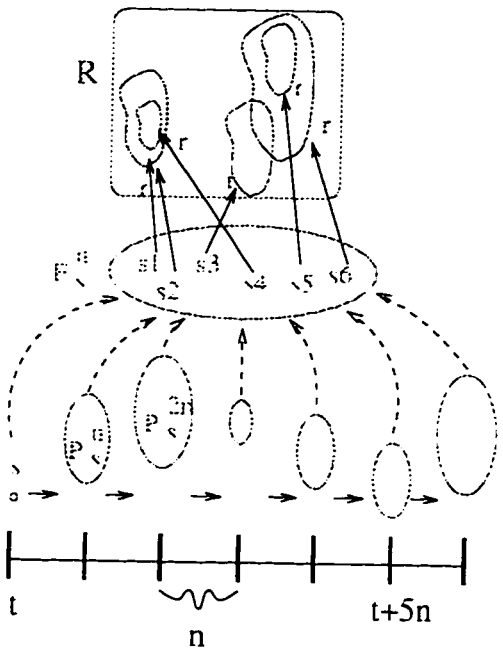

Figure 3.8: Predictable temporal patterns

3.6 Discussion

The restricted approach to designing time-critical control systems exemplified by the circuit paging problem has been motivated as a way to solve on-line decision-making problems for which decision-making behavior and time-critical control system design can be decoupled. We have shown under what conditions the sequencing of decision-making circuits can compose a decision-making solution, sketched a specific paging architecture for implementing this solution, and formalized the intuitive notion of predictable temporal patterns of situations with respect to this architecture.

In order to directly tackle a given on-line decision-making problem, it remains to be shown that this restricted approach can actually be used to solve problems. A circuit paging problem is solved by generating an executable strategy circuit to page decision-making circuits such that the strategy implements the required covering specified by a given sequence of sets of possible situations. Alternatively, we may show that such a strategy does not exist. These topics are explored in Chapter 4.

Chapter 4

Solving a Circuit Paging Problem

In this chapter we develop analytical techniques for evaluating and solving circuit paging problems. In Section 4.1, we first define the components of a specific problem and then, in Section 4.2, provide an existence lemma to capture the necessary and sufficient conditions under which a strategy exists to solve the specific problem. In Section 4.3, we show that, in general, it is a hard problem to determine whether or not a strategy exists that satisfies the conditions of this lemma; and explore in detail the implications of this result.

In Section 4.4, we reverse the hardness reduction in order to derive an approximation algorithm for generating circuit paging solutions that satisfy coverage requirements. In order to evaluate the approximation algorithm we introduce, in Section 4.5, detailed notions of the executability of paging strategy circuits. The approximation algorithm is then, in Section 4.6, shown to produce exact solutions for an easy special case. Finally, in Section 4.7, we employ the executability analysis and the approximation algorithm as components in developing enumeration (search) schemes. A specification of this enumeration scheme may be used to solve the specific circuit paging problem exactly, by determining whether or not an executable strategy exists and, if so, generating a corresponding circuit covering.

4.1 Paging Arbitrary Collections of Executable Circuits

Specifying a circuit paging problem requires choosing a paging architecture and describing the predictable temporal patterns of situations that can arise through interaction of decision-making environment and decision-making solution under the chosen architecture. In this chapter, we assume that there exists some achievable desired behavior E^* that is interchangeable with respect to an arbitrary collection of executable decision-making circuits. The goal of this chapter is to implement this behavior by solving a circuit paging problem, if possible.

Lemma 3.1 of Chapter 3 reduces a decision-making solution generation problem to a covering problem. We show that the predictable temporal patterns of situations developed in Section 3.5 can be used to derive the required coverings under the circuit paging architecture of Section 3.4. Given these patterns and paging architecture, we determine the conditions under which we can guarantee that an arbitrary collection of executable circuits can implement the achievable desired behavior E^*. By arbitrary, we mean that there is no global structure governing the domains across circuits. The problem is captured in the following definition.

Definition 4.1 *In an* arbitrary executable circuit collection, bipartite cache, fixed retrieval interval, circuit paging problem *(arbitrary-circuit paging, in short) we are given:*

- *a read-only storage device capable of storing at most D pages/circuits and fast cache of M bits capable of simultaneously storing two decision-making circuits and one fixed paging strategy circuit over D pages,*
- *a fixed retrieval interval n,*
- *a collection of stationary Markov decision-making circuits C that is interchangeable with respect to some achievable desired behavior E^*, each $c_S \in C$ covering domain $V_S \in 2^S$, and each executable with respect to execution and retrieval time and space limitations, and*

- *a stationary Markov transition model defined by k-tick transitions P_s^k and the set of possible situations at initial decision-making tick, P^0.*

The arbitrary-circuit paging decision problem is to determine whether or not there exists a subset $C' \subseteq C$ of size $|C'| \leq D$ and an executable paging strategy circuit, γ, that guarantees that any on-line sequencing of decision-making circuits attainable under the paging strategy implements the desired behavior E^*.

4.2 Existence of Solutions

In this section, we discuss the predictable temporal patterns of situations developed in Section 3.5 as they relate to the requirements of the arbitrary-circuit paging problem. We then show that these patterns can be used in conjunction with Lemma 3.1 to determine whether or not an arbitrary-circuit paging problem has a solution.

Under the circuit paging architecture of Section 3.4, an initial decision-making circuit π_0 is determined at design time. This circuit must cover the set of possible situations reachable prior to tick n, *i.e.* R_0^n. Given that the initial decision-making circuit is required to be loaded into fast cache prior to decision making, it is not technically paged-in the same way as subsequent circuits, and has slightly different executability requirements. However, to simplify the presentation, we include the initial possible situation set R_0^n as part of the coverage requirement of the paging solution, and consider the initial circuit π_0 to be drawn from the collection of decision-making circuits C.

A paging strategy circuit γ selects an executable decision-making circuit to retrieve from read-only storage, based on the current situation, every n ticks, starting with the initial tick. F^n defines all situations that could possibly occur at the beginning of a retrieval interval and, therefore, must be covered by any fixed paging circuit that implements desired behavior E^*.

Additionally, F^n can be used to derive the sets of possible situations specified in Lemma 3.1. In particular, since we must be prepared to page-in an executable decision-making circuit based on any situation that is observed at the beginning

of a retrieval interval, the subset of circuits forming an arbitrary-circuit paging solution must include at least one that covers R_s^n for each situation $s \in F^n$. We can relate this to the sequence $\{PS^i\}_{i \geq 0}$ by noting that, by definition, PS^i must never exclude the situation eventually observed at tick i, in the given execution. For the selection and execution patterns of the arbitrary-circuit paging problem, if s is observed at time tick nj $(j \geq 0)$ in a given execution, R_s^n captures all possible situations that may be observed in time ticks $n(j+1) \leq i < n(j+2)$. Thus, for each $n(j+1) \leq i < n(j+2)$, $PS^i \subseteq R_s^n$ in that execution[1]. The set $P^{nj} \subseteq F^n$ captures all such situations, s, observable at time tick nj, that can be determined at design time. Thus, for each situation $s \in F^n$, if R_s^n is covered by some decision-making circuit in the solution subset, then we know that we can select a decision-making circuit to cover each $\{PS^i\}_{i \geq 0}$ in any execution.

The above discussions are formalized in the following lemma, which provides necessary and sufficient conditions for determining whether or not a solution exists to an arbitrary-circuit paging problem.

Lemma 4.1 *Given a stationary Markov transition model P_s^k and initial set of situations $P^0 \in 2^S$, a decision-making solution exists that implements achievable desired behavior E^* for an* arbitrary-circuit paging problem *if and only if there exists a subset $C' \subseteq C$ of size $|C'| \leq D$ such that, for each $s \in F^n$, $R_s^n \subseteq V_S$ for some circuit $c_S \in C'$, $R_0^n \subseteq V_S$ for some circuit $c_S \in C'$, and there exists a fixed strategy γ with domain F^n and range C' that implements the mapping of situations to decision-making circuits and is executable under the given time and space limitations.*

Note that Figure 3.8 of Chapter 3 depicts the predictable temporal patterns of interest in Lemma 4.1.

Lemma 4.1 states that we can concentrate our efforts on determining whether or not a small enough covering of decision-making circuits over sets of possible situations exists; and whether or not an executable paging strategy to implement such a covering exits. Once we know that such a covering and strategy exist, we can use the conditions of the lemma to construct a circuit paging solution. The

[1] In fact, $PS^i = P_s^{i-nj}$ from the definition of R_s^n.

rest of this chapter is devoted to determining whether or not a suitable covering exists and, if so, finding the covering.

Unless otherwise indicated, we assume that D in Definition 4.1 is derived taking into account the physical size of read-only storage, the worst-case time and space of the decision-making circuits in C, and the worst-case time and space of any fixed strategy circuit. In other words, we may assume that the existence of $C' \subseteq C$ satisfying the conditions of Lemma 4.1 implies the existence of an executable strategy. In Section 4.5 we are more specific about what it means to be executable within time and space limitations.

4.3 Complexity

Lemma 4.1 provides the necessary and sufficient conditions for determining whether or not an arbitrary collection of executable circuits can satisfy the coverage requirements of a stationary Markov transition model, under the arbitrary-circuit paging architecture. Presumably, we need only figure out what it means for a strategy to be executable within a given instantiation of the architecture and then find a subset of circuits C' that satisfies the conditions of Lemma 4.1 and for which an executable strategy can be constructed. Unfortunately, the following theorem states that determining whether or not such a subset of circuits exists is a hard problem, even if we ignore the additional difficulty of ensuring that an executable strategy circuit exists. The result is obtained through reduction from *minimum set cover* [24].

Theorem 4.1 *The decision problem of Lemma 4.1 is NP-complete.*

The hardness result of Theorem 4.1 indicates that, despite our intention, the simplifications we have made by focusing on the circuit paging problem rather than the completely un-decoupled on-line decision-making solution design problem, have not removed all the complexity in the task. However, the reduction from minimum set cover provides clues to solving the specific arbitrary-circuit paging problem. Toward this goal, we briefly summarize the current knowledge on minimum set cover.

Definition 4.2 *The* set cover *problem consists of the following components [24, 14, 13].*

INSTANCE: *Collection Y of subsets of a finite set X.*

SOLUTION: *A set cover for X, i.e. a subset $Y' \subseteq Y$ such that every element of X belongs to at least one member of Y'.*

The *minimum set cover* optimization problem is to find a set cover of minimum cardinality. This problem has been well-studied. Not only is it NP-complete but, under commonly believed structural complexity assumptions, it is neither possible to find an efficient constant factor approximation nor an efficient approximation with any factor better than $\log(m)$, $m = |X|$. More specifically, approximating minimum set cover to any constant is NP-complete and approximating minimum set cover within $\Theta(\log m)$ implies NP $\subseteq$ DTIME($m^{\log(\log m)}$) [4]. Note that, by Theorem 4.1, these results are directly extended to the arbitrary-circuit paging problem.

Nevertheless, approximation algorithms for minimum set cover exist. The following classical greedy algorithm [13, 47] iteratively selects a set from the family Y that covers the most elements of X not yet covered, until all elements are covered.

```
Greedy-Set-Cover(X, Y)
U = X
Y' = ∅
while U ≠ ∅
   do select a y ∈ Y that maximizes |y ∩ U|
      U = U − y
      Y' = Y' ∪ {y}
return Y'
```

The classical analysis [12, 47, 50] provides an upper bound on the performance ratio of the greedy algorithm of $\mathcal{H}(\max\{|y| : y \in Y\})$, where $\mathcal{H}(d) = \sum_{1 \le i \le d} 1/i = \log(d) + O(1)$ is the d^{th} harmonic number. The variation in which $K = \max\{|y| : y \in Y\}$ may be bound in advance is sometimes referred to as the *minimum K-set cover* problem. Recently, Slavík [68] improved the analysis by providing a tight bound of $\log(m) - \log\log(m) + \Theta(1)$ for the greedy algorithm, including an upper bound of $\log(m) - \log\log(m) + 0.78$ and a lower bound of $\log(m) - \log\log(m) -$

0.31. Furthermore, Halldórsson [40] provides a local improvement extension of the greedy algorithm with an upper bound of $\mathcal{H}(K) - 13/30$ for the minimum K-set cover variation.

4.4 Approximation Algorithm

The hardness result of Theorem 4.1 entails a reduction from minimum set cover to the arbitrary-circuit paging existence problem of Lemma 4.1. In order to make use of the approximation algorithms for minimum set cover we must provide a reduction in the opposite direction. We must construct a finite set X and a collection Y of subsets of X such that a set cover $Y' \subseteq Y$ over X satisfies the covering requirements of Lemma 4.1.

In an arbitrary-circuit paging problem we are given a collection of circuits C with domains over $\mathcal{S}$, a retrieval interval n and a stationary Markov transition model (P^0, P_s^k). From the transition model and retrieval interval we can derive the collection of sets of situations $R = \{R_0^n\} \cup \{R_s^n | s \in F^n\}$ that might occur over the course of on-line decision making. The *circuit cover* requirement of Lemma 4.1 states that each possible set in this collection must be covered by some circuit in the solution $C' \subseteq C$. Instead of constructing the finite set X from R, we can reduce the size of collection of sets that need to be covered by the following observation.

Lemma 4.2 *A solution exists to an* arbitrary-circuit paging problem *with collection of sets of situations R if and only if a solution exists with collection of sets of situations $R' = \{r' \in R | \forall r \in R, r \neq r' \rightarrow r' \not\subset r\}$.*

In other words, in determining whether or not a family of circuits covers a collection of sets of possible situations, it is enough to look at only those sets that are not contained in any other set in the collection. This result is due to the fact that, since every set $r \in R$ must be covered by some circuit $c \in C'$, the circuit that covers r will also cover all subsets of r. Lemma 4.2 cannot be directly applied if the coverage guarantee of solutions given by Lemma 3.1 is not satisfied.

The finite set X is now constructed such that each element corresponds to a member set of the collection R'. *i.e.* $X = R'$. This construction is depicted in Figure 4.1.

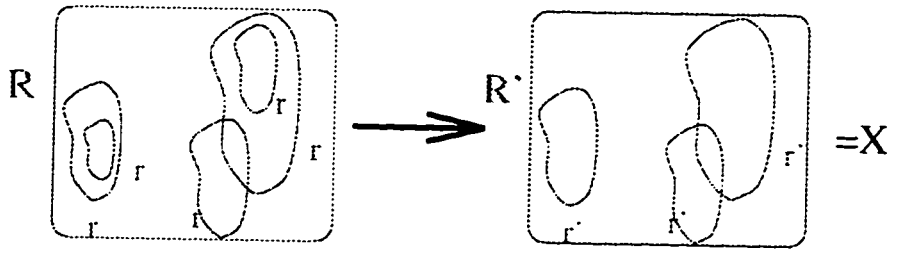

Figure 4.1: Constructing the set of ground elements X from the flattened collection of sets of reachable situations

Next, we construct the family of sets Y by comparing the flattened collection of sets R' to the collection of circuits C. The following definition captures this construction.

Definition 4.3 *Let* $y_S = \{r' \in X | r' \subseteq V_S\}$. *where* V_S *is the domain of circuit* $c_S \in C$. *Then* $Y = \{y_S | c_S \in C\}$.

This construction is depicted in Figure 4.2.

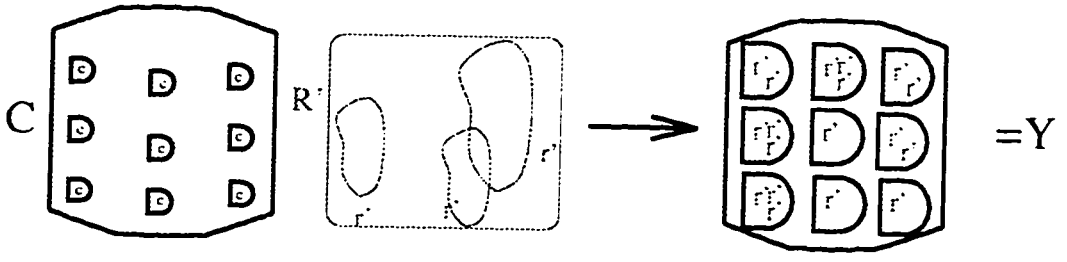

Figure 4.2: Constructing the family of sets Y by comparing a collection of decision-making circuits C to the flattened collection of sets of reachable situations R'.

X and Y can be constructed in time and space polynomial in the size of the set of situations $|S|$ and set of circuits $|C|$. We first observe that the size of R' (and R) is bounded by the size of the set of situations: plus one additional member for the initial set of possible situations, R_0^n. In particular $R' \subseteq R$. $|R| \leq |F^n| + 1 \leq |S| + 1$. And for each $r \in R$, $|r| \leq |S|$. Using a naive

approach. X can be constructed by pairwise comparison of members of R to determine containment, and Y can be constructed from pairwise comparisons of R' and C. Therefore, the naive approach provides worst-case time and space of $O(|S|^3 + |S|^2|C|)$, with further reductions possible using other approaches. Note that additional information about the source might be used to further reduce the upper bound on $|R'|$.

Theorem 4.2 *Let finite set X and collection Y describe the set cover problem constructed from a circuit cover problem with possible sets R and circuits C, as defined above. $Y' \subseteq Y$ is a set cover for X of size $|Y'|$ if and only if $C' = \{c_S \in C | y_S \in Y'\}$ is a circuit cover for R of size $|C'|$.*

By Theorem 4.2 and Lemma 4.2, we can produce an approximation algorithm for minimum circuit cover from an approximation algorithm for minimum set cover using the above construction. In particular, let $Y' \subseteq Y$ be a set covering produced by Greedy-Set-Cover applied to X and Y as constructed above. The corresponding circuit covering is $C' = \{c_S \in C | y_S \in Y'\}$. By Lemma 4.1, if $|C'| \leq D$ then C' is an executable solution to the arbitrary-circuit paging problem. Furthermore, we can use the classical analysis of Greedy-Set-Cover to achieve a stronger result.

From the classical analysis of Greedy-Set-Cover and Theorem 4.2, we can see that the performance ratio of our approximation algorithm for minimum circuit cover is at most $\mathcal{H}(|R'|) \leq \mathcal{H}(|F^n|) \leq \mathcal{H}(|S|)$. This directly leads to the following relationship between D, $|R'|$ and the existence of an executable circuit paging solution.

Corollary 4.1 *Let $C' \subseteq C$ be the circuit covering produced by Greedy-Set-Cover applied to X and Y in the above construction. If $|C'| > D\mathcal{H}(|R'|)$ then an executable circuit covering ($C^* \subseteq C$, $|C^*| \leq D$) does NOT exist.*

Thus, an executable solution exists if $|C'| \leq D$ and does not exist if $|C'| > D\mathcal{H}(|R'|)$. In fact, this result can be strengthened by applying the analysis of Slavík [68] and algorithm of Halldórsson [40]. For $D < |C'| \leq D\mathcal{H}(|R'|)$, we may

continue to search for an executable solution. This issue is taken up shortly. An example solution to the arbitrary-circuit paging problem is depicted in Figure 4.3.

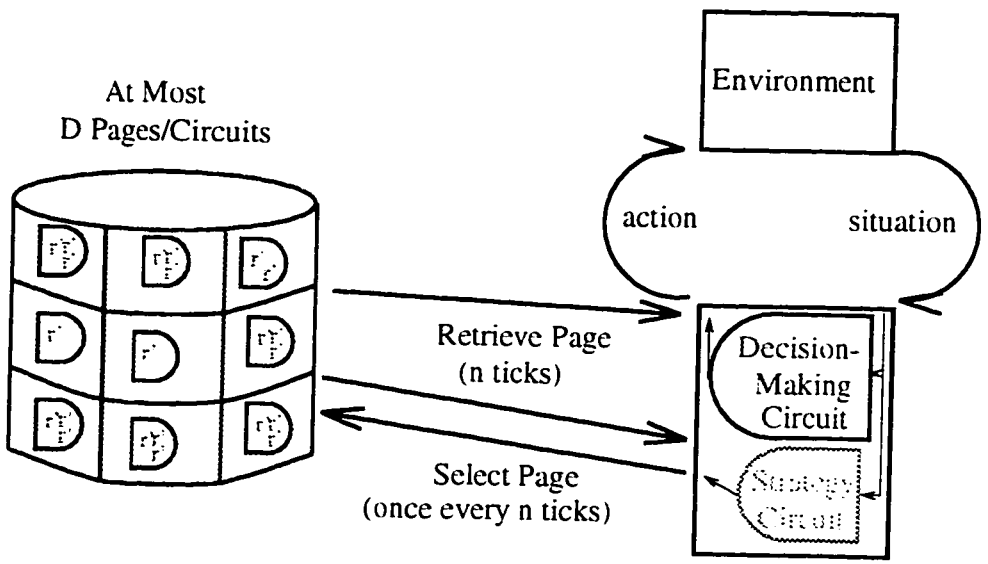

Figure 4.3: Providing an on-line decision-making solution under a two-level paging architecture

The results presented thus far rely on the fact that D is given as part of the problem statement. In particular, Corollary 4.1 relies on the assumption that D is constant over all possible circuit coverings. In other words, by definition, any circuit covering must have cardinality at most D in order to be executable.

However, D represents a potentially interesting tradeoff between the number of decision-making circuits contained in read-only storage and the worst-case time and space requirements of circuits in this set. It may be useful to explicitly consider this tradeoff in constructing solutions to an arbitrary-circuit paging problem. In order to evaluate this possibility we must look more closely at our notions of executability.

4.5 Executability

In previous sections we have been able to present the arbitrary-circuit paging problem without detail about what it means for a decision-making circuit or strategy circuit to be executable within time and space limitations. We have assumed that the derivation of D takes into account these details. We now develop this derivation, based on the architectural constraints sketched in Section 3.4.

For any arbitrary executable circuit collection C, M-bit bipartite cache, n-tick retrieval interval, circuit paging problem, we can define the following quantities:

$T_\delta^*(C)$ describes the worst-case number of ticks required to execute any decision-making circuit in C:

$B_\delta^*(C)$ describes the worst-case number of bits required to store any decision-making circuit in C:

$T_\gamma(d,k)$ describes the worst-case number of ticks required to execute a strategy circuit selecting one of d circuits, using $k \leq |\mathcal{S}|$ input situations:

$B_\gamma(d,k)$ describes the worst-case number of bits required to store a strategy circuit selecting one of d circuits, using $k \leq |\mathcal{S}|$ input situations; and,

T_ρ describes the worst-case number of ticks required to retrieve one bit of data from a read-only paging source.

The paging architecture of Section 3.4 specifies that 1) any decision-making circuit must execute in at most one tick of time; 2) two decision-making circuits plus the fixed strategy circuit must be stored simultaneously in M bits of on-line storage; and 3) the combined process of executing the strategy circuit, locating the page on disk and retrieving the page, must execute in at most n ticks of time. Given a transition model for which F^n defines all input situations that could possibly occur at the beginning of a retrieval interval, these constraints may be captured, respectively, in the following equations.

$$T_\delta^*(C) \leq 1 \tag{4.1}$$

$$2 * B_\delta^*(C) + B_\gamma(d, |F^n|) \leq M \quad (4.2)$$

$$T_\gamma(d, |F^n|) + T_\rho B_\delta^*(C) \leq n \quad (4.3)$$

The number of circuits that may be stored on the read-only storage device may be additionally limited by physical storage constraints. However, under the reasonable assumption that the physical limitations of read-only storage are not binding, D may be derived by taking the maximum d that satisfies the above equations.

Lemma 4.3 *Let $D = \max d$, such that Equations 4.1, 4.2, 4.3 are satisfied. An executable strategy circuit exists for* any *collection $C' \subseteq C$ such that $|C'| \leq D$.*

Recall that, given that the initial decision-making circuit is loaded into fast cache prior to decision making, it is not technically paged-in the same way as subsequent circuits. This potentially impacts only Equation 4.3, which pertains to retrieval time bounds. If Equation 4.3 is binding, this opens the possibility that the initial decision-making circuit may be larger than the other circuits without violating the executability conditions. We do not consider this possibility any further in this chapter.

Equations 4.1, 4.2, 4.3 can also be used to provide a bound on the maximum cardinality of a circuit covering for which a strategy circuit exists, for a specific covering $C' \subseteq C$. Note that the bound D' for a specific covering C' may be weaker than the bound D that must apply to *any* covering. This relationship is presented in the following lemma.

Lemma 4.4 *An executable strategy circuit exists for a specific circuit covering $C' \subseteq C$ if and only if there exists a D' such that*

$$\begin{aligned} |C'| &\leq D' \\ T_\delta^*(C') &\leq 1 \\ 2 * B_\delta^*(C') + B_\gamma(D', |F^n|) &\leq M \\ T_\gamma(D', |F^n|) + T_\rho B_\delta^*(C') &\leq n \end{aligned}$$

Lemma 4.4 provides a potentially weaker bound (more slack) for evaluating the executability of circuit coverings generated by Greedy-Set-Cover. In particular, a covering C' generated by Greedy-Set-Cover is executable if $|C'| \leq D'$, $D' \geq D$. In Section 4.7 we use Lemma 4.4 to develop an exact algorithm for finding executable circuit coverings. Prior to that, we present an easy special case.

4.6 Easy Special Case

It is interesting to consider the relationship between $|R'|$ of Lemma 4.2 and D of Lemma 4.3. Consider the case $|R'| \leq D$. When $|R'| \leq D$ the combinatorics of finding circuits to cover multiple possible sets disappears. In particular, we can construct a circuit cover by individually covering each $r' \in R'$. The following lemma helps us show that Greedy-Set-Cover already takes advantage of this easy special case.

Lemma 4.5 *If $C' \subseteq C$ is a circuit-covering produced by Greedy-Set-Cover applied to X and Y in the construction of Section 4.4, then $|C'| \leq |R'|$.*

The insight of Lemma 4.5 provides us with a condition under which the approximation algorithm for arbitrary-circuit paging based on Greedy-Set-Cover always produces a feasible solution.

Theorem 4.3 *If $|R'| \leq D$ then Greedy-Set-Cover always produces an executable circuit covering C', $|C'| \leq D$.*

Theorem 4.3 indicates that we can apply the construction of R' from R as given in Lemma 4.2 and compare to the value of D given as part of the problem statement. If $|R'| \leq D$ then we know the problem is easy to solve. If $|R'| > D$ we might still find an executable solution by applying our Greedy-Set-Cover-based approximation algorithm: either directly, if possible, or as part of an enumeration method we shortly specify in Section 4.7.

However, if $|R'| > D$ we may instead consider the cost of adding additional on-line resources such that we can guarantee, if any circuit covering exists, then

an executable circuit covering exists that is easy to find. To simplify the analysis assume that the worst-case time to execute a strategy circuit is independent of the number of pages in read-only storage. In other words, $T_\gamma(d,k) = T_\gamma(1,k), \forall d$. For example, we might assume that a strategy circuit to sequence d pages is constructed as a parallel combination of d 1-page circuits, with negligible additional delay. Furthermore, assume that such a strategy circuit is always executable for all $k \leq |\mathcal{S}|$ as defined by Equation 4.3. These assumptions are not strictly necessary but, allow us to focus solely on the effects of limited on-line cache.

Under these assumptions, we need only consider a modified version of Equation 4.2 to provide the desired resource bound. The following lemma provides an upper bound on the amount of on-line cache M needed to guarantee that a particular arbitrary-circuit paging problem has an easy solution, if it has any solution.

Lemma 4.6 *For any arbitrary-circuit paging problem described by possible situations sets R, circuits C, cache size M, and maximum pages D, if a circuit covering exists then an executable circuit covering can always be found in polynomial time if*

$$M \geq 2 * B^*_\delta(C) + B_\gamma(|R'|, |F''|)$$

The result of Lemma 4.6 provides an upper bound on the amount of on-line cache needed to solve a particular problem, under the assumption that the amount of on-line cache is the binding resource limitation. Such a measure can be used to guide the selection of paging architectures.

We can combine this approach with Lemma 4.4 to determine an upper bound on the amount of on-line cache needed to execute a particular covering. In particular, observe that we can substitute any covering C' produced by Greedy-Set-Cover into Lemma 4.6 and directly calculate an upper bound on the amount of on-line cache needed to execute that covering: namely, $M \geq 2 * B^*_\delta(C') + B_\gamma(|C'|, |F''|)$.

4.7 Developing an Exact Algorithm

In this section we show how to use of the results of the previous sections as components in developing an exact algorithm that determines whether or not an executable strategy circuit exists for any arbitrary-circuit paging problem: and, if so, finds the corresponding covering. In Chapter 7 we demonstrate an exact algorithm using these components that solves a restricted formulation of the arbitrary-circuit paging problem.

As shown, if D is fixed as part of the problem statement, we can initiate our search for executable solutions by first applying Greedy-Set-Cover to generate $C' \subseteq C$ and then checking the conditions of Corollary 4.1 and Theorem 4.3. These results allow us to limit our search to those problems for which $|R'| > D$ and $D < |C'| \leq D\mathcal{H}(|R'|)$. If $|R'| \leq D$ and/or $|C'| \leq D$ then Greedy-Set-Cover provides an executable solution. If $|C'| > D\mathcal{H}(|R'|)$ then an executable solution does not exist. However, these results only apply to problems for which D is fixed as part of the problem statement. If we allow D to vary as D', as specified in Lemma 4.4, then these results do not directly apply.

The result of Corollary 4.1 does not hold if D is not fixed as part of the problem statement. Instead, Lemma 4.4 leads to the following modified corollary.

Corollary 4.2 *Let $Y' \subseteq Y$ be a set covering produced by Greedy-Set-Cover(X, Y) and let $y^*(Y') = \max_{\{y_S \in Y'\}} |y_S|$ be the size of the largest set in the covering. Furthermore, let $C' = \{c_S \in C | y_S \in Y'\}$ be the corresponding circuit covering and let D' be the maximum d satisfying Equations 4.1, 4.2, 4.3 with $C = C'$.*

If $|C'| > D'\mathcal{H}(y^*(Y'))$ then there does NOT exist an executable circuit covering for any collection $C^* \subseteq C$ such that $B^*_\delta(C') \leq B^*_\delta(C^*)$.

Corollary 4.2 provides a bound to eliminate from consideration any covering containing decision-making circuits taking at least as much space as the worst-case circuit in C'. Note that Corollary 4.2 does not eliminate from consideration any covering $C^* \subseteq C$ such that $B^*_\delta(C') > B^*_\delta(C^*)$.

Thus far, we have presented several techniques to use in developing an exact algorithm for solving the arbitrary-circuit paging problem. The Greedy-Set-Cover-based approximation algorithm of Section 4.4 can be used to find feasible circuit coverings, the executability equations 4.1, 4.2, 4.3 may be used to evaluate the executability of feasible coverings, and Corollary 4.2 can be used to eliminate certain decision-making circuits from our solution search. We now present an integer programming formulation of the minimum set cover problem to use as an additional component in developing exact algorithms.

Definition 4.4 *For any set covering problem described by elements X and collection Y, let variable $Z_S = 1$, if set $y_S \in Y$ is in the covering Y', and 0, if not. Furthermore, let constant $W_{x',S} = 1$, if $x' \in y_S$, and 0, if not. $\forall x' \in X, y_S \in Y$. The* minimum set covering integer program *is then:*

$$\begin{array}{rrl} Z^I = & \min \sum_{\{y_S \in Y\}} Z_S & \textit{such that} \\ \textit{for each } x' \in X\textit{:} & \sum_{\{y_S \in Y\}} W_{x',S} Z_S & \geq 1 \\ \textit{for each } y_S \in Y\textit{:} & Z_S & \in \{0,1\} \end{array}$$

Minimum Set Covering Integer Program (MSCIP)

The following theorem states that an integer solution to MSCIP is a set covering of minimum cardinality and infeasibility implies that no set covering exists.

Theorem 4.4 *Let finite set X and collection Y describe a set cover problem. If $Y' = \{y_S \in Y | Z_S = 1\}$ is a covering derived from an optimal solution to MSCIP then Z^I is the minimum cardinality of any covering $Y^* \subseteq Y$ of X. Furthermore, if MSCIP is infeasible then no feasible covering exists.*

Combining Theorem 4.2 and Theorem 4.4 reveals that we can evaluate the executability of an arbitrary-circuit paging problem by solving MSCIP and comparing to our upper bound D. This is true if D is fixed as part of the problem statement but, requires additional mechanisms to take advantage of the flexibility introduced in Section 4.5. Furthermore, solving a general integer program is at least as difficult as solving a minimum set covering problem.

One way to make use of MSCIP without immediately solving the integer program is to solve the relaxed linear programming version. The relaxed linear programming version of MSCIP removes the 0-1 integer constraint on each Z_S such that the relaxed solution value Z^F is a lower bound on the integer solution value, $Z^F \leq Z^I$. Additionally, if the relaxed problem is infeasible then so is the integer problem. Therefore, the relaxed linear program may be used to obtain a lower bound on the number of circuits in any executable covering. As with the lower bound developed in Corollary 4.2, this lower bound may be used to eliminate certain decision-making circuits from our solution search. Namely, we may eliminate any decision-making circuit whose inclusion in a solution leads to an upper bound on pages, via the executability equations, that is below the relaxed lower bound.

Thus, both the integer and linear programming versions of MSCIP may be used as components in developing an exact algorithm for solving the arbitrary-circuit paging problem. We may combine them with our techniques for finding feasible circuit coverings, evaluating the executability of feasible coverings, and eliminating circuits through lower bounds.

In the following, we demonstrate one way to use these techniques to develop an exact algorithm. We begin by using the techniques of Section 4.4 to convert an arbitrary-circuit paging problem to a set cover problem. We then call an exact algorithm based on solving the set cover problem augmented with executability equations. Finally, we re-construct a circuit cover solution from the executable set cover solution. Figure 4.4 provides the procedure *Executable-Circuit-Cover* to convert to and from a circuit cover problem.

The *Executable-Set-Cover* exact algorithm first applies a loop consisting of solving the linear programming relaxation, testing against the executability equations, and removing "too large" decision-making circuits. The loop is repeated until a fractional solution is found that requires fewer pages than the maximum number of pages allowed by solving the executability equations. We then apply the greedy set cover algorithm to try to construct an executable integer set cover

```
Executable-Circuit-Cover(n, C, P_s^k, P^0)

Construct patterns of possible situations from stochastic source
   R_0^n = ∪_{0 ≤ k < n} P^k
   R_s^n = ∪_{n ≤ k < 2n} P_s^k
   F^n = ∪_{j ≥ 0} P^{n j}
   R = {R_0^n} ∪ {R_s^n | s ∈ F^n}
   R' = {r' ∈ R | ∀ r ∈ R, r ≠ r' → r' ⊄ r}

Construct components of equivalent set cover problem
   X = R'
   y_S = {r' ∈ X | r' ⊆ V_S} where V_S is the domain of c_S ∈ C
   Y = {y_S | c_S ∈ C}

Solve set cover problem and interpret results
   Y' = Executable-Set-Cover(X, Y)
   If (Y' == ∅) return ∅
   C' = {c_S ∈ C | y_S ∈ Y'}
   return C'
```

Figure 4.4: Executable circuit cover procedure

solution. If this fails, we apply Corollary 4.2 toward further eliminating decision-making circuits. If these steps neither provide an executable solution nor remove all decision-making circuits from consideration, we initiate a structured enumeration (search) of the potentially reduced solution space. This algorithm sketch is depicted in Figure 4.5.

The algorithms of Figures 4.4 and 4.5 provide a solution sketch for incorporating the analytical techniques of this chapter into an exact algorithm. We have been intentionally vague about the correct and complete structured enumeration required to complete the sketch. In Chapter 7 we provide a way to complete this algorithm sketch by developing a mixed integer linear program (MILP) that augments MSCIP with a simplified version of the Equations 4.1, 4.2, 4.3. We then use a standard branch and bound algorithm to provide the required correct and complete enumeration.

One approach to providing a structured enumeration that takes advantage of the components developed in this chapter is to develop an iterative scheme that alternates linear programming and greedy set covering steps and uses the

upper bound: Calculate the maximum executable number of sets (circuits).
D = max d satisfying Equations 4.1,4.2,4.3

largest circuits: Construct the collection of all sets representing circuits of the largest size.
$Y_{\max} = \{y_S \in Y \mid B^*_\delta(\{c_S\}) \geq B^*_\delta(C)\}$

relaxed lower bound: Get lower bound on executable solution by solving linear programming relaxation of minimum set cover problem.
Z^F = value of relaxed solution to MSCIP
If relaxed MSCIP is infeasible return $\emptyset$

executable: The following loops eliminate each layer of sets corresponding to the largest circuits, until it is possible that the remaining sets form an executable set cover (similar to Corollary 4.2).
while ($Z^F > D$)
while ($Z^F > D$)
$Y = Y - Y_{\max}$
if ($Y == \emptyset$) return $\emptyset$
else recalculate D and $Y_{\max}$ for new Y
Z^F = value of relaxed solution to MSCIP with new Y
If relaxed MSCIP is infeasible return $\emptyset$

If lower bound is executable and an integer solution, then we are done.
If ($Z^F \leq D$) and all $Z_S \in \{0,1\}$
return $Y' = \{y_S \in Y \mid Z_S = 1\}$

greedy covering: Find a greedy set covering and calculate the maximum number of sets (circuits) for this solution to be executable.
Y' = Greedy-Set-Cover(X, Y)
D' = max d satisfying Equations 4.1,4.2,4.3
with $C = \{c_S \in C \mid y_S \in Y'\}$

If greedy cover is executable, then we are done.
if ($|Y'| \leq D'$) return Y'

greedy lower bound: Use greedy cover to determine new lower bound on optimal covering. If this lower bound is larger than linear programming lower bound and not executable then we can eliminate another layer of sets (via Corollary 4.2).
else if ($|Y'| \,/\, \mathcal{H}(y^*(Y')) > D'$)
$Y'_{\max} = \{y_S \in Y \mid B^*_\delta(\{c_S\}) \geq B^*_\delta(C')\}$
$Y = Y - Y'_{\max}$
if ($Y == \emptyset$) return $\emptyset$

enumerate: Feasible relaxed covering exists, provably too-large circuits removed. Search for an executable solution.

Figure 4.5: Executable-Set-Cover(X, Y)

associated bounds to cut off branches that can not lead to an executable solution. While such an approach is attractive, it is easy to develop a scheme that attempts to cut off branches by repeatedly solving very similar problems, making very slow progress toward a solution.

Another approach is to develop a best first search routine that starts from the greedy solution and tries to improve upon it in a systematic fashion, such as through a local exchange, or through random perturbation. We do not explore alternative enumeration schemes any further in this work.

Given a correct and complete structured enumeration, the combined procedure of Executable-Circuit-Cover and Executable-Set-Cover provides a correct and complete algorithm for constructing a solution to an arbitrary circuit paging problem.

The primary intention of this section is to demonstrate the existence of an exact algorithm for constructing circuit paging solutions. Secondarily, we demonstrate the use of the approximation algorithms and executability conditions developed in previous sections. The resulting procedure is designed to quickly converge on an executable covering, if one exists; or to quickly prove that an executable covering does not exist. However, in the worst-case, a correct and complete enumeration scheme may enumerate all possible set coverings. The running time would then be worst-case exponential in the number of circuits in the problem statement.

There may be additional efficiencies and analyses that could be used to design an enumeration scheme that is not worst-case exponential. However, due to the hardness results of Section 4.3, a polynomial exact algorithm for this problem is unlikely, in general.

The practical efficiency of the algorithm depends upon the frequency with which processing terminates prior to reaching the enumeration stage, and the number of decision-making circuits that can be removed prior to enumeration. In Chapter 7 we develop an enumeration scheme and explore the efficiency of this algorithm with respect to a specific application developed in Chapters 5 and 6.

4.8 Discussion

In this chapter, we have demonstrated the use of a decoupled on-line decision-making solution design approach for implementing desired behavior by sequencing the on-line execution of a collection of decision-making circuits. We have focused on the arbitrary-circuit paging instantiation of the circuit paging problem. We have taken the process from problem definition to approximation solution and analysis through exact solution sketch. To do so, we have fixed components of the problem that are actually parameters of the larger on-line decision-making solution design task. In Chapter 9 we take another look at the broader task and begin to map out the space of problems.

The balance of this chapter is an attempt to cope with the complexity results of Section 4.3. While we have made progress in coping with this hurdle, we have not completely overcome the complexity, in general.

Ironically, this complexity is evidence that the study of design-time tradeoffs in on-line decision-making solution design is a worthwhile exercise. The complexity arises from an explicit formulation of the type of tradeoffs that must be made in order to solve on-line decision-making problems. Our method of formulating circuit paging problems allows these tradeoffs to be performed at design time, ensuring that time-critical on-line decision making is tractable within resource limitations. This compares favorably to approaches in which this complexity appears (implicitly or explicitly) at execution time. Design-time versus on-line complexity tradeoffs is a recurring theme in this work.

Chapter 5

Arbitrary-Circuit Paging: Real-time Avionics Application

This thesis develops a framework for the analysis and design of on-line decision-making systems. The arbitrary-circuit paging architecture is a two-level time-critical control system intended to conceptually model the tradeoffs inherent in the interaction of slow deliberation and fast reaction. In Chapters 3 and 4, we develop analytical techniques to understand this interaction under specific environmental models and models of resource requirements and limitations.

In addition to being viewed as a conceptual model, we might consider the use of the arbitrary-circuit paging architecture as the basis for designing on-line decision-making solutions. In this chapter, we explore this possibility by presenting a real-time avionics system (Section 5.1) and then extending this system through the use of the arbitrary-circuit paging architecture (Section 5.2). We may then apply the analytical techniques of Chapters 3 and 4 toward understanding the behavior of the new solution (Section 5.3).

5.1 Real-time Avionics

In modern *fly-by-wire* aircraft all flight operations are computer controlled. There are no mechanical connections between pilot and airplane. Correspondingly, it is

critical that the electronics designed to control these aircraft adhere to strict FAA validation. In this section we describe a real-time avionics (*aviation electronics*) system similar to the Airplane Information Management System (AIMS) designed by Honeywell for the Boeing 777 [10, 45].

AIMS consists of two redundant rack-mounted hardware cabinets. Each cabinet contains a 2-bit parallel *SAFEbus* and a number of Line Replaceable Modules (LRMs). Processor LRMs provide processing power for flight operations, while I/O LRMs provide processing of flight data I/O. Multiple LRMs compose a distributed multiprocessor hardware system. Each processor LRM contains bus interface circuitry, processor, intermodule memory and a schedule table implemented in eprom. Figure 5.1 illustrates a distributed multiprocessor avionics hardware system similar to AIMS.

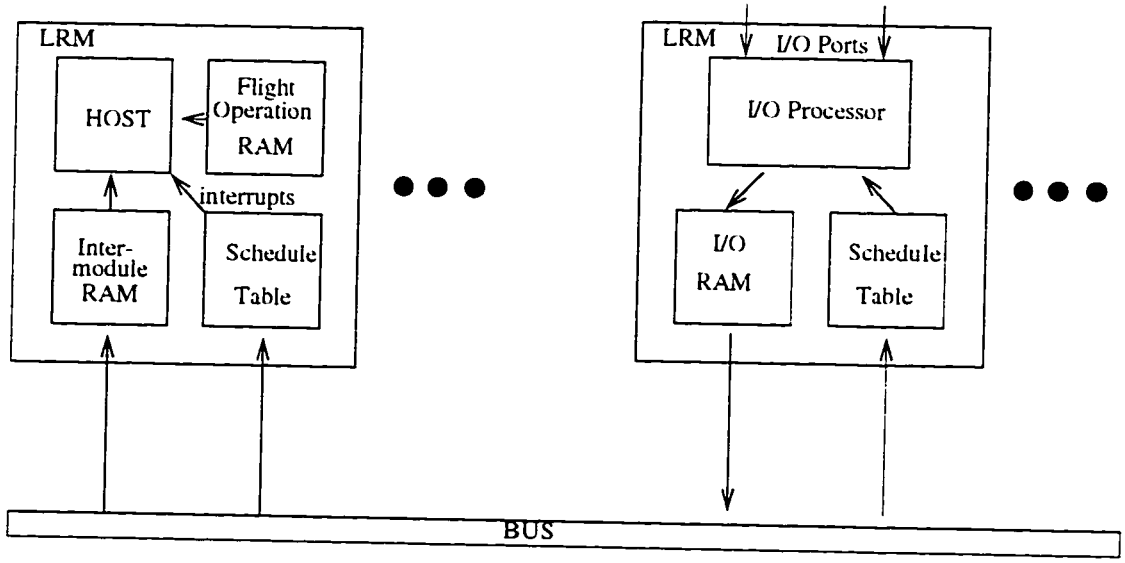

Figure 5.1: Distributed multiprocessor avionics hardware

Schedule tables provide synchronization for processing and access to the shared bus. A schedule table has a fixed number of entries and can implement a cyclic schedule that repeats after a fixed time period. Each entry consists of either a data transmit, data receive, or processor interrupt command. Data transfer commands hard-code storage locations in intermodule memory for transmission

or reception. Interrupt commands are used to signal process swaps to the real-time OS in the host processor of the given LRM. Parallel processing LRMs are synchronized to execute commands in their schedule tables at the same pace. In this way, the flow of data across the SAFEbus and the processing activity of each LRM are deterministically synchronized.

Real-time avionics scheduling, in this context, refers to the generation of schedule tables, for all LRMs, such that the resulting sequences of transmit, receive and interrupt commands satisfy the I/O and flight operation processing requirements of the avionics system. Each LRM has limited processing time and memory space for implementing flight operations. Such flight operations might include thrust management, flight-data acquisition, flight and navigation display systems, flight management, climate control, data communications management, central maintenance, etc.. Schedule tables are generated at design time and must be guaranteed to provide safe, time-critical behavior, consistent with FAA regulations, under uncertain in-flight conditions.

Real-time avionics scheduling is complicated by subtle dependencies between flight operations, and competition among flight operations for limited data bus resources and limited LRM processing time and space resources. The design task is to figure out how to use the limited hardware resources to satisfy the requirements of the flight operations. The traditional approach to real-time avionics scheduling is to design a static schedule that covers all possible in-flight conditions.

An important requirement of real-time avionics scheduling is that each flight operation must be scheduled according to its predetermined *sampling frequency*. The sampling frequency specifies the number of times per second a flight operation must be sampled on the distributed hardware in order to ensure time-critical response. For example, in AIMS these frequencies range from 5-80 hertz. Other requirements include the execution time per sampling, data size, and dependencies between operations and across samplings.

Required sampling frequencies are determined according to traditional real-time repetition robustness criteria. Traditional criteria, in effect, specify *worst-case* frequency requirements. The minimum sampling frequency of each flight operation is dictated by the worst-case environmental situations the aircraft might encounter. Schedules are statically constructed to adhere to this worst-case requirement.

Together, the frequencies over all flight operations determine a *cyclic* scheduling task. Within each cycle, each operation must be scheduled its required frequency of instantiations on the distributed multiprocessor hardware; with dependencies and other requirements satisfied. In the AIMS system, the static cyclic schedule consists of thousands of jobs that must interact with the asynchronous external environment through shared I/O ports. Figure 5.2 depicts a worst-case static schedule implemented in schedule table eprom. This schedule remains constant over time. Each flight operation is always sampled at its worst-case frequency, regardless of in-flight conditions.

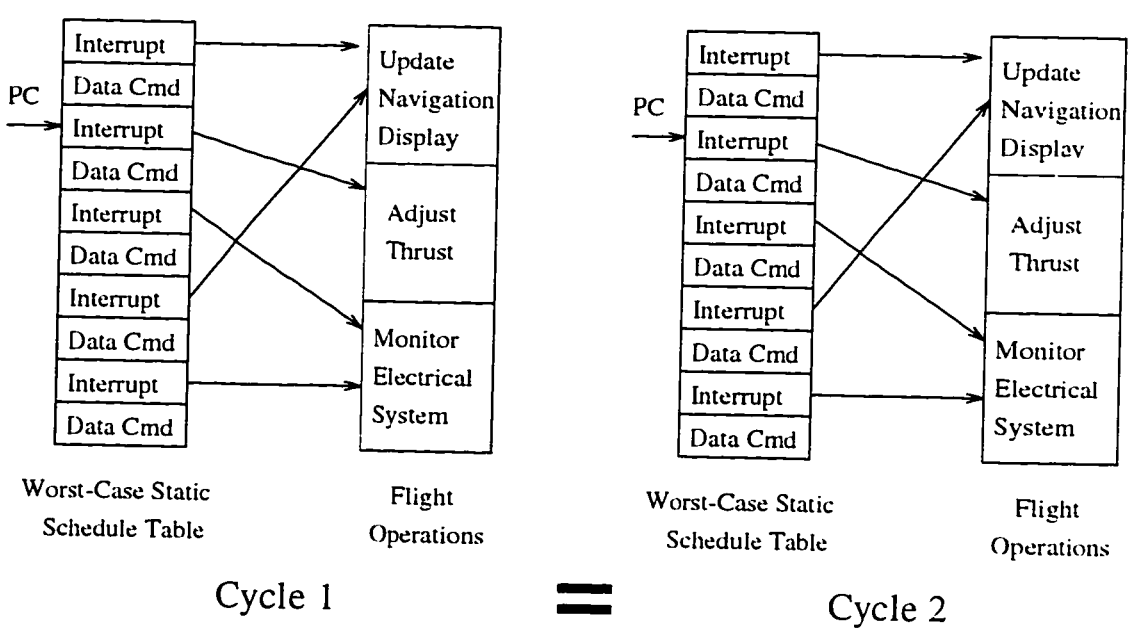

Figure 5.2: Worst-case static schedules are fixed at design time *i.e.* they do not change from one cycle to the next

5.2 Conditional Real-time Avionics Scheduling

In this section, we develop an extension to the traditional approach to real-time avionics scheduling. The extension may be accommodated with few changes to existing avionics hardware platforms, such as the distributed multiprocessor architecture described in Section 5.1.

Static scheduling has traditionally been necessary to guarantee safe behavior. However, in the following extension we show that, for problems with certain properties, static schedules may be replaced with with conditional schedules without sacrificing real-time behavior guarantees. The underlying premise is that flight operation sampling frequency requirements may be re-expressed relative to current flight conditions. Sensing of on-line situations can then be employed to determine the actual frequency requirements of each cycle of a given execution.

Conditional schedules allow more effective use of limited resources by taking advantage of situation-specific conditional relationships between flight operations. These relationships are obscured by worst-case frequency requirements. For example, the thrust-management operation may require frequent execution during take-off and landing, while stabilization management may require more frequent execution during cruising. Additionally, climate control may need more monitoring during warm clear weather conditions, while the navigation control may require more frequent monitoring during foggy conditions. Executing at worst-case frequency for all processes, all the time, may never be necessary in practice.

Figure 5.3 illustrates example components of a real-time avionics system that accommodates conditional schedules. These components include existing processors and the shared bus, and are augmented with additional resources to sense in-flight conditions and conditionally modify execution schedules for flight operations depending upon the sensed situation.

In order to accommodate conditional scheduling, we must provide mechanisms to alter the execution schedule over time. These mechanisms must obey well-specified time and space limitations in order to avoid sacrificing real-time behavior guarantees. The arbitrary-circuit paging architecture may be used as the basis for

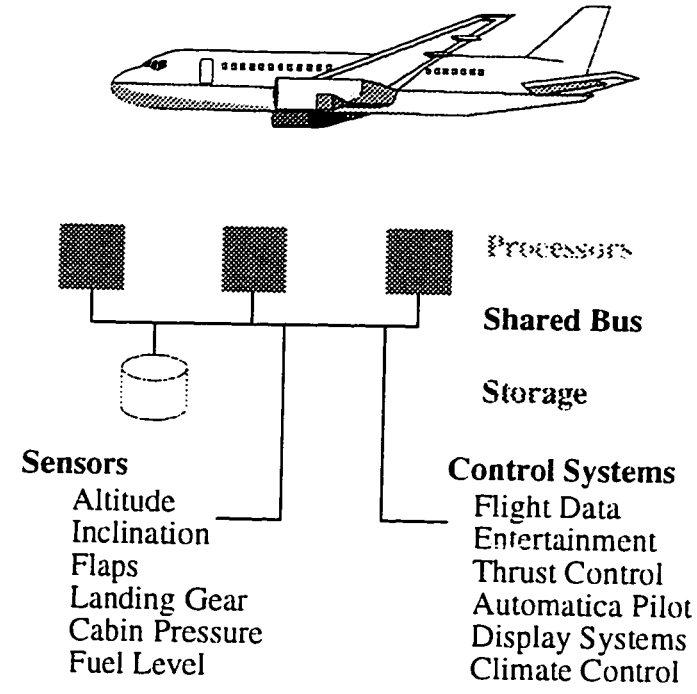

Figure 5.3: Example conditional real-time avionics system components

extending existing avionics hardware platforms to handle conditional scheduling, without sacrificing behavior guarantees.

We implement the arbitrary-circuit paging architecture, within existing distributed multiprocessor avionics hardware, by introducing an additional LRM to serve as a *schedule server*. The schedule server LRM is packed with eproms that contain pre-compiled conditional schedules, each targeted toward different in-flight conditions. The schedule server LRM also contains a *paging circuit* that is designed to "eavesdrop" on all flight data that is transmitted by the I/O LRMs, and use this data to select the most appropriate conditional schedule. Figure 5.4 depicts the augmented hardware.

At fixed intervals of time (synchronized by the schedule table), the paging circuitry determines the current situation from the flight data and uses this situation to select a conditional schedule. The conditional schedule is then transmitted to the processor LRMs (again, synchronized by the schedule table). The conditional schedule is loaded into the intermodule memory of each LRM and specifies a new

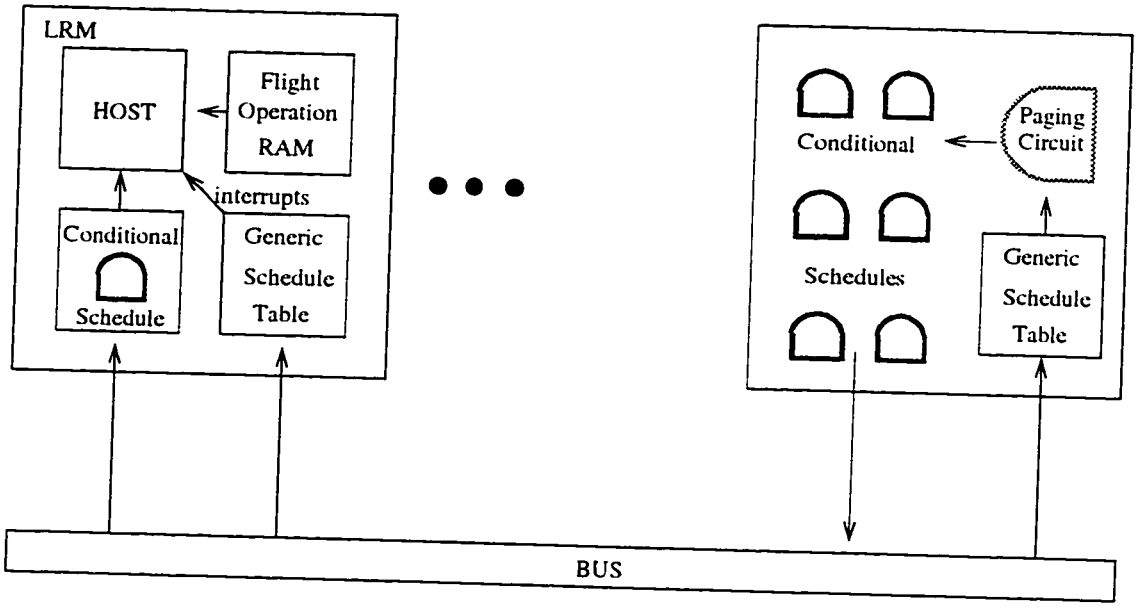

Figure 5.4: Distributed multiprocessor avionics hardware augmented with a schedule server LRM

pattern for flight operation execution over the next fixed interval of time. Conceptually, the schedule server LRM can be viewed as a deliberative black box that "delivers", to the processor LRMs, execution schedules that are guaranteed appropriate to the current situation, according to well-specified timing patterns.

All conditional schedules must be consistent with the fixed synchronization pattern of the schedule tables. To do so, we design a *generic schedule table* with pre-determined interrupt and transmit/receive commands. The flight operation that is launched during a given interrupt depends upon the conditional schedule currently loaded into intermodule memory. Data transfers are treated similarly. In other words, the host processor of the LRM interprets the commands of the fixed generic schedule table differently depending upon the varying conditional schedule in intermodule memory. In this way, synchronization may still be guaranteed deterministically at design time. Note that bus traffic due to paging must be taken into account in the generic schedule table.

Figure 5.5 depicts the change of execution from worst-case static schedules to conditional schedules. Under static schedules, the schedule table commands point directly to hard-coded flight operations, and remain constant over time. Under

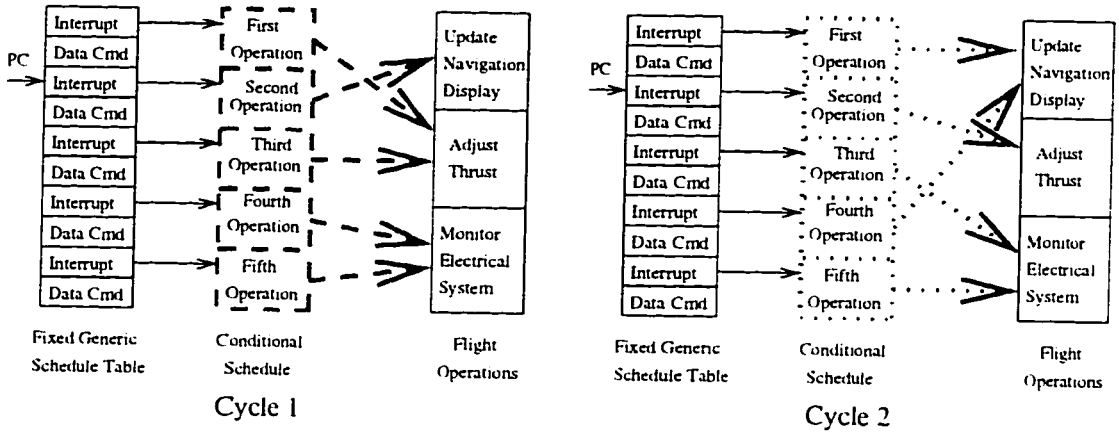

Figure 5.5: Conditional schedules alter scheduling of flight operations over time

conditional schedules, the generic schedule table points to hard-coded addresses in intermodule memory that store the commands of the current conditional schedule. As shown, the commands in one cycle of processing may differ from the previous cycle as new conditional schedules are transmitted from the schedule server LRM. Each conditional schedule contains the same flight operations as the static schedule, but the frequency with which any given flight operation is sampled changes over time.

5.3 Guaranteeing Behavior Under Conditional Real-time Avionics Scheduling Solutions

Section 5.2 provides the computational mechanisms for a conditional real-time avionics scheduling solution. It does not, however, describe how to determine the requirements that conditional schedules must satisfy, given a particular flight environment. In this section we develop the details for relating a model of the in-flight environment to the conditional frequency requirements of flight operations. We do so in a way that mirrors the development of Chapter 3 and allows us to use the analytical techniques of Chapter 4 to provide behavior guarantees.

5.3.1 Conditional Frequency Model

Static schedules must satisfy requirements determined by worst-case sampling frequencies. Similarly, conditional schedules must satisfy requirements determined by situation-specific sampling frequencies. In order to evaluate the requirements of a given flight environment we must more precisely characterize these situation-specific sampling frequencies.

The following conditional frequency model expresses situation-specific frequency requirements, both in terms of individual situations and sets of situations. The model is intended to capture the frequency at which a flight operation must be sampled to achieve safe behavior, given a specific known situation or set of possible situations. Situation-specific frequencies are extended to sets of situations by taking the maximum required frequency over the set.

A *situation* s, in this context, is an assignment of values to a set of variables corresponding to in-flight conditions such as altitude, inclination, cabin pressure, and fuel level. Let $\mathcal{S}$ be the space of situations, suitably defined over sensor readings, and $\mathcal{O}$ be the space of flight operations.

Definition 5.1 *Let $\nu_o(s) \in \mathbf{Z}$ be defined as the* instantaneous frequency requirement *of flight operation $o \in \mathcal{O}$ in situation $s \in \mathcal{S}$. For any set of possible situations $S \subseteq \mathcal{S}$, the frequency requirement of a given flight operation $o \in \mathcal{O}$ is defined as the maximum frequency over all situations in that set, namely $\nu_o(S) = \max_{\{s \in S\}} \nu_o(s)$.*

Note that worst-case sampling frequencies can be derived as a special case of Definition 5.1, in which we take the maximum frequency over all possible situations, $\nu_o = \nu_o(\mathcal{S}) = \max_{\{s \in \mathcal{S}\}} \nu_o(s)$. This is consistent with the traditional static approach to real-time avionics scheduling, which does not incorporate on-line sensing of situations and, thus, requirements must be defined over any possible situation in the space of situations.

The development of specific conditional frequencies for a given flight environment must be approached similarly to the traditional task of designing worst-case frequencies. Such procedures are problem-specific. An illustrative example of conditional sampling frequencies for real-time avionics systems is given in Table 5.1.

This example consists of six different weather-related situations, and sampling frequencies for three different flight operations. Each flight operation is sampled either once or twice per cycle as a function of the in-flight situation.

Situation	Update Navigation Display	Adjust Thrust	Monitor Electrical System
Clear/calm	1	1	1
Clear/low winds	1	1	1
Clear/high winds	1	2	1
Cloudy	1	1	1
Fog	2	1	1
Lightning storm	1	1	2

Table 5.1: Situation-specific conditional sampling frequencies

5.3.2 Temporal Patterns of Situations

The conditional frequency model of a given real-time avionics problem provides the situation-specific sampling requirements of each flight operation. In order to completely evaluate the requirements with respect to the design of a conditional scheduling solution, we require a model of the patterns of situations that may occur over time. This model is then combined with the frequency model to determine, at design time, whether or not a given conditional scheduling solution may be employed as part of a real-time avionics solution.

However, deriving a model of temporal patterns of situations is complicated by the fact that scheduling decisions made in response to a current situation can affect future in-flight situations. Thus, a model of patterns of situations cannot be derived without understanding how the avionics system will interact with situations in the environment, *i.e.* without knowing the conditional scheduling solution.

Another way to look at the traditional approach to real-time avionics scheduling is to note that, as long as the specified worst-case frequency requirements are

achieved by the static solution, safe behavior is guaranteed. The worst-case frequency model provides a way to achieve safe behavior without explicitly modeling the interaction of avionics system and external environment. In other words, a scheduling solution need only guarantee that the given frequencies are achieved in order to guarantee that the behavior is achieved. This, effectively, decouples design-time tradeoffs in schedule generation from behavior specification.

To avoid the circularity of needing to know the conditional scheduling solution in order to construct the conditional scheduling solution, we make the following *decoupling* assumption. We assume, in constructing a model of temporal patterns of situations, that the conditional scheduling solution will guarantee safe behavior by ensuring that the conditional schedule executing at any given time achieves the sampling frequency required by any situation that may occur during that time. We may then use the analytical techniques of Chapter 4 to prove whether or not a conditional scheduling solution provides this property.

Deriving a temporal pattern of situations involves capturing the transitions that can occur between situations, assuming safe behavior. We may then combine this with a model of conditional frequencies to help generate and evaluate conditional scheduling solutions. Figure 5.6 depicts an example transition model augmented with the example conditional frequency model presented in Section 5.3.1. For example, this figure shows that, if the current situation is Clear/low winds, the situations that are possible in the next cycle are Clear/low winds, Clear/calm, Clear/high winds, and Lightning storm. Additionally, depending upon the actual situation, the sampling frequency of either the adjust thrust operation or the monitor electrical system operation may need to be doubled to satisfy safe behavior requirements.

In general, a transition model can be represented by a set of situations that are possible at the start of execution, P^0; and a function capturing the transitions between situations, P_s^k. P_s^k describes the set of situations possible at tick $t + k$ given situation s at tick t, under safe behavior. The arcs in Figure 5.6 represent a single-tick transition model P_s^1.

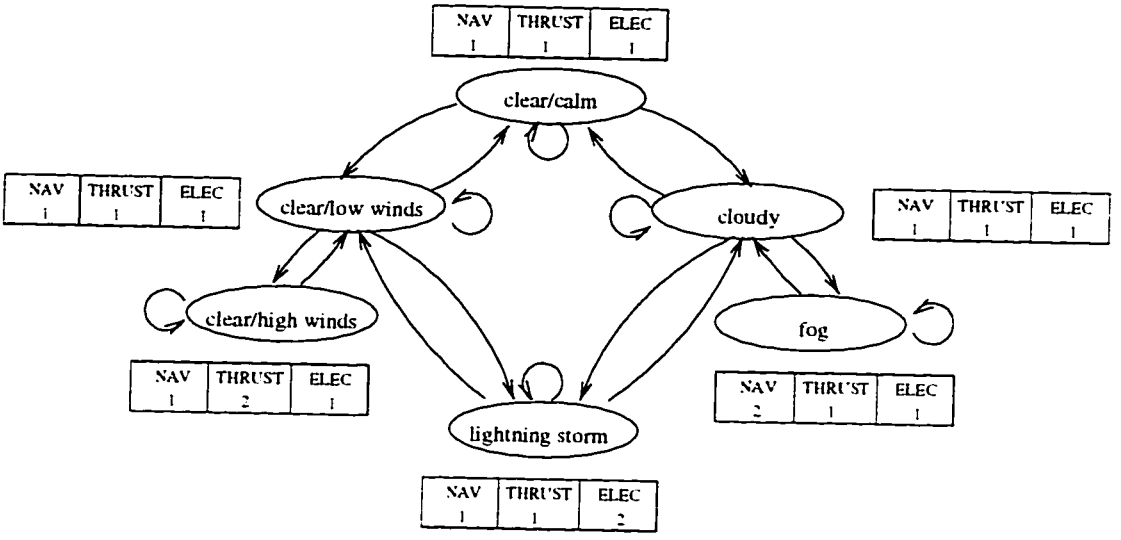

Figure 5.6: Example flight environment transition model and per situation conditional frequency model

In order to evaluate a conditional scheduling solution based on the arbitrary-circuit paging architecture, we use the transition model to derive sets of situations that represent specific temporal patterns relevant to the execution of the solution. In particular, the conditional scheduling mechanisms described in Section 5.2 specify a fixed pattern of paging and execution. This pattern consists of fixed-interval paging decisions and a fixed-interval execution cycle, as determined by the generic schedule table.

Define an execution cycle to span n ticks. Once every n ticks, the current situation is determined from on-line flight data and is used as input to the paging circuitry. The paging circuitry must select a conditional schedule to transmit to the processor LRMs. This conditional schedule will apply during the next cycle. The next cycle occurs n ticks into the future and spans the next n consecutive ticks.

The temporal patterns of situations relevant to the conditional scheduling mechanisms are based on those developed in Chapter 3. We must model the set of situations that may occur when paging decisions are made, F^n, and the collection of sets of situations that may occur during each execution cycle, $R = \{R^n_s | s \in F^n\}$. Where $R^n_s = \cup_{\{n \leq k < 2n\}} P^k_s$.

It is reasonable to assume that the flight environments of real-time avionics problems are *ergodic*. In other words, in the long run any situation can occur at any time. Thus, we typically define F^n to be the set of all possible situations $\mathcal{S}$. Therefore, the primary temporal pattern of situations we require is the collection of sets $R = \{R_s^n | s \in \mathcal{S}\}$.

5.3.3 Coverage Requirements

In Section 5.3.2, we derive the collection $R = \{R_s^n | s \in \mathcal{S}\}$ as the primary temporal pattern of situations of interest. Definition 5.1 provides a direct way to combine the collection R with a situation-specific sampling frequency model in order to re-express R in terms of frequency requirements. The resulting collection of frequency requirements form the basis of evaluating a conditional scheduling solution. This collection is captured in the following definition.

Definition 5.2 *For any set of situations r, let $\nu(r) = \{\nu_o(r) | o \in \mathcal{O}\}$ correspond to the set of frequency requirements for each flight operation over all situations in r. Given a collection of sets of situations $R = \{R_s^n | s \in \mathcal{S}\}$ and a frequency model ν, define the* coverage requirement *to be the collection $R_\nu = \{\nu(R_s^n) | s \in \mathcal{S}\}$. Denote the member of R_ν corresponding to situation $s \in \mathcal{S}$, by $R_\nu(s)$ i.e. $R_\nu(s) = \nu(R_s^n)$.*

Table 5.2 provides the coverage requirement for the example conditional frequency and transition models of Figure 5.6. In the example problem, $n = 1$. Thus, $R_s^n = P_s^1$, for each $s \in \mathcal{S}$. The table depicts the single-transition neighbors of each situation, the instantaneous frequency requirement of each flight operation for each neighbor, and the coverage requirement due to the situation. The coverage requirement over all situations is $R_\nu = \{111, 122, 121, 212, 211, 112\}$. Note that the coverage requirement can be further summarized by the flattened collection $R'_\nu = \{122, 212\}$.

If we suppose that only five total samplings are allowed per cycle and that the lightning storm situation requires two navigation samplings per cycle (instead of one), the problem becomes infeasible. In particular, after sensing a clear/low

Situation (s)	neighbors (P_s^1)	$\nu(s')$ for each $s' \in P_s^1$	$R_\nu(s)$
Clear/calm (C/C)	$\{C/C.C/L.CY\}$	$\{111.111.111\}$	111
Clear/low winds (C/L)	$\{C/C.C/L.C/H.LS\}$	$\{111.111.121.112\}$	122
Clear/high winds (C/H)	$\{C/L.C/H\}$	$\{111.121\}$	121
Cloudy (CY)	$\{C/C.CY.F.LS\}$	$\{111.111.211.112\}$	212
Fog (F)	$\{CY.F\}$	$\{111.211\}$	211
Lightning storm (LS)	$\{C/L.CY.LS\}$	$\{111.111.112\}$	112

Table 5.2: Deriving coverage requirements given temporal patterns of situations based on single-tick cycles, and a corresponding frequency model

winds situation, we would require a worst-case schedule (222) requiring six samplings in order to guarantee safety.

5.3.4 Guaranteed Safe Behavior

Chapter 4 develops a general framework for comparing temporal patterns of situations to the *domains* of a known collection of decision-making circuits. In a conditional real-time avionics scheduling solution, the conditional schedules are analogous to the decision-making circuits. While we may express the domains of conditional schedules in terms of situations, it is more natural to express them in terms of sets of sampling frequencies. The following definition captures this intuition.

Definition 5.3 *The* domain V_S *of conditional schedule* c_S *is the minimum sampling frequency* $\bar{\nu}_o$ *that it guarantees for each flight operation,* $o \in \mathcal{O}$. $V_S = \{\bar{\nu}_o | o \in \mathcal{O}\}$. $\bar{\nu}_o \in \mathbf{Z}$.

Guaranteeing safe behavior of a real-time avionics system depends upon showing that flight operation frequency requirements are satisfied at all times. These time-critical response requirements must adequately cover over all possible in-flight conditions. The coverage requirement R_ν of Definition 5.2 captures the collections of frequencies that must be covered by conditional schedules in order to claim that a conditional scheduling solution can satisfy the requirements of a given flight environment.

Since Definitions 5.2 and 5.3 express covering requirements and conditional schedule domains, respectively, in terms of sampling frequencies, it is straightforward to compare these collections using the analytical techniques of Chapter 4. The following lemma compares a given collection of conditional schedules and the coverage requirement of a flight environment.

Lemma 5.1 *Given a collection of conditional schedules C, a flight environment (P^0, P_s^k), and a frequency model ν, a conditional scheduling solution guarantees safe behavior only if for each $r \in R$, there exists some conditional schedule $c_S \in C$ such that, for each $o \in \mathcal{O}$, $\bar{\nu}_o \in V_S$ and $\nu_o(r) \leq \bar{\nu}_o$*

Figure 5.7 shows an example collection of conditional schedules that satisfies the conditions of Lemma 5.1.

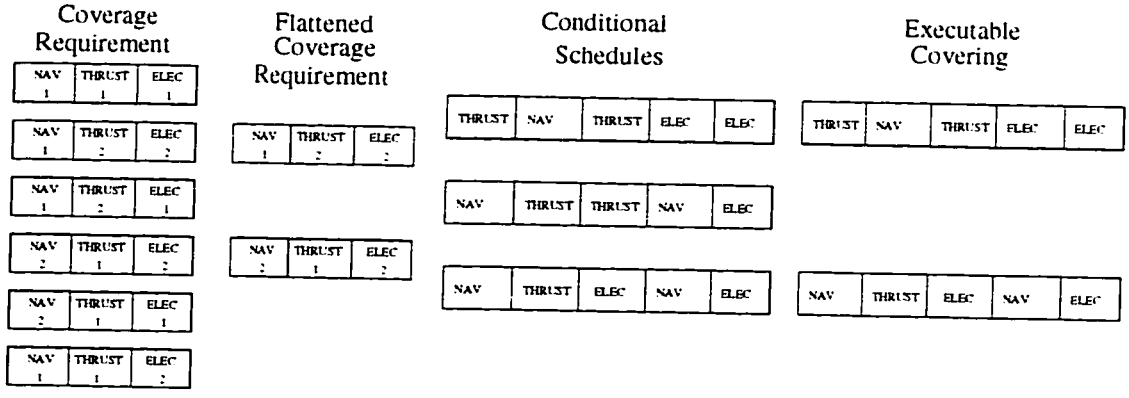

Figure 5.7: Executable covering derived by comparing flattened coverage requirements to collection of conditional schedules

In order to guarantee safe behavior, we must show that Lemma 5.1 is satisfied *and* that the computational mechanisms to perform paging can be accommodated in the limited time and space resources. To evaluate the latter constraint, in Chapters 6 and 7 we develop models of time and space resources in the context of parameterized conditional scheduling solutions.

5.4 Discussion

This chapter develops a conditional scheduling approach to real-time avionics system design. This new approach demonstrates the the utility of the on-line decision-making solution design and analysis framework developed in this thesis by using the framework both to design the approach and to analyze the behavior of solutions under the approach. This solution approach may be extended to provide a new way to pose and solve a variety of general scheduling problems in hard real-time environments under uncertainty. In particular, this technique may be applied to any scheduling problem for which worst-case load requirements can be weakened by conditioning on on-line situations.

This conditional scheduling approach not only provides a more flexible and effective use of limited resources, but it can additionally provide new capabilities to real-time avionics systems. For example, by allowing sensor-based switching of schedules, we can achieve emergency-specific responses that differ from normal flight responses in cases for which these responses are dissimilar.

Chapter 6

An Empirical Evaluation of Framework and Techniques

In Chapters 2, 3 and 4, we develop a framework and set of techniques for the analysis and design of on-line decision-making solutions. In Chapter 5, we demonstrate the use of the framework and techniques in developing a new approach to an existing problem. To complete our understanding of these results, in this chapter we develop and motivate an empirical study of arbitrary-circuit paging with respect to conditional real-time avionics scheduling. In Chapter 7, we present the results of this study.

This empirical evaluation has two goals. Primarily, we want to understand the practicality of the techniques of this thesis: in terms of both the practical computational efficiency of design-time techniques, and the ability to attain positive results (executable solutions). Secondarily, we want to further validate our new approach to designing real-time avionics solutions by demonstrating scheduling solutions that provide a more flexible and effective use of limited resources.

A fundamental difficulty with developing an empirical study of a new problem-solving approach is that data for evaluating the approach may not exist. With respect to the conditional real-time avionics scheduling approach, there exist neither conditional frequency models, conditional schedules nor schedule servers. Thus, we must generate these models from available knowledge, balancing the

desire to obtain general results with the requirements of a specific study. The goal of this chapter is to generate an informative study based on what we know or can reasonably extrapolate.

In Section 6.1, we formulate a specific static scheduling solution that is a restricted version of general real-time avionics schedules. In Chapter 5, we propose that conditional scheduling provides a way to potentially accommodate more flight operations in a schedule by reducing the sampling demands of some subset of flight operations. Rather than test this hypothesis empirically, we provide two parameters to capture various reductions in sampling demand, and show that the relationship between functionality and sampling demand can be understood analytically.

We then focus our attention on the relationship between collections of conditional schedules, collections of environmental requirements and bounds on hardware resources. In Sections 6.2 we introduce a single test parameter to capture the bounds on hardware resources and requirements of conditional schedules. In Section 6.3 we introduce three final parameters to quantify the size of the collections of conditional schedules and environmental requirements in a given test case.

6.1 Improving Solutions by Accommodating Additional Flight Operations

In this section, we analyze the claim that the conditional scheduling approach can add functionality to a scheduling solution without increasing hardware resources. We do so by establishing a specific static scheduling solution and two parameters capturing various reductions in flight operation sampling demand. We then provide an analytical relationship between these parameters. The form of schedule studied in this section is a restriction on the general approach of Chapter 5 that removes complexities common to both static scheduling and conditional scheduling.

6.1.1 A Restricted Static Scheduling Solution

In this restricted static scheduling solution, we make assumptions about the number of flight operations, the execution time requirement and sampling frequency requirement of each operation, the data and ordering dependencies between operations, and the schedule cycle duration. In particular, we assume that our avionics scheduling problem is solved by a static schedule that cycles every 1000 milliseconds. Each 1000 millisecond cycle consists of 1000 flight operation instantiations of one millisecond each. These 1000 instantiations are distributed among 10 flight operations such that each receives a common worst-case sampling frequency of 100 samplings per cycle. This schedule is summarized in Table 6.1.

Cycle Duration (milliseconds)	Flight Operations (quantity)	Common Worst-Case Sampling Frequency (Hertz)	Common Execution Time (milliseconds per sampling)
1000	10	100	1

Table 6.1: Restricted static scheduling solution

Note that this restricted scheduling solution does not model multiple processors, I/O requirements, or ordering of flight operation samplings.

Figure 6.1 illustrates the restrictions of this section with a worst-case cyclic static schedule similar to that of Figure 5.2 of Chapter 5. There are three flight operations, each with a common worst-case sampling frequency of two hertz and common execution time of one time slice. The static schedule consists of six evenly divided time slices. Given that the ordering of instantiations is not relevant, Figure 6.1 depicts the instantiations in fixed consecutive order, per flight operation. Data commands are evenly distributed to account for any required I/O.

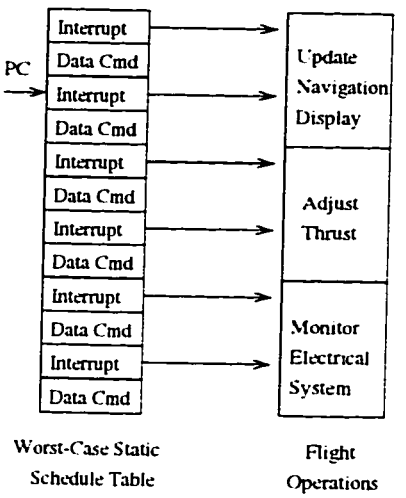

Figure 6.1: Worst-case static schedule under restricted model

6.1.2 Relating Functionality Improvement to Parameterized Flight Operation Sampling Demand

The conditional scheduling approach allows us to reduce the sampling frequency requirement of each flight operation, depending upon the in-flight conditions. Reducing sampling requirements adds more slack to a schedule and potentially allows us to either reduce the hardware resources or increase the number of flight operations that may be accommodated within the same resources. In this section, we assume that hardware resources for processing flight operations are fixed and concentrate on the additional functionality that conditional frequencies might provide. The additional functionality of interest is the number of flight operations, above 10, that can be accommodated in a 1000 instantiation schedule.

The first parameter concerns the number of samplings per operation per cycle. We provide a *common reduction factor* $cf \in [0, 1]$ such that all flight operations that do not require the worst-case sampling of 100 instantiation per cycle, can be sampled at the rate of $100cf$ instantiations per cycle.

The second parameter concerns the mix between operations sampled at worst-case frequency and those sampled at the reduced frequency. Define the maximum number of flight operations that must be sampled at the worst-case frequency to be $\bar{x} \in [0, 10]$. Note that $cf = 1$ and $\bar{x} = 10$ correspond to the static worst-case

sampling requirements, allowing at most 10 flight operations to be scheduled.

The third parameter captures the number of flight operations that we desire to accommodate in a scheduling solution, $|\mathcal{O}| \geq 10$. The relationship between the maximum number of flight operations, the common reduction factor and the maximum number of flight operations that must be sampled at worst-case frequency can be evaluated analytically. In particular, given a 1000 instantiation cycle, the relationship is expressed in the following lemma.

Lemma 6.1 *Given a common reduction factor of* $cf \in [0.1]$, *such that all but* $\bar{x} \in [0.10]$ *flight operations may be sampled at the rate of* $100cf$ *per cycle, the maximum number of flight operations,* $|\mathcal{O}| \geq 10$, *that may be accommodated in a given schedule are*[1]

$$|\mathcal{O}| \leq \frac{10 - \bar{x}}{cf} + \bar{x}.$$

For any two parameters, $|\mathcal{O}|$, cf, or $\bar{x}$, Lemma 6.1 may be used to determine bounds on the third. In particular, we may re-express the relationship as $0 \leq cf \leq \frac{10-\bar{x}}{|\mathcal{O}|-\bar{x}} \leq 1$ or $0 \leq \bar{x} = \lfloor \frac{10-|\mathcal{O}|cf}{1-cf} \rfloor \leq 10$. Figure 6.2 graphically depicts the upper bound on cf as a function of $|\mathcal{O}|$, for values of $\bar{x} \in \{1.3.5.7.9\}$. For example, in order to accommodate 30 flight operations, 5 of which are sampled at 100 cycles per second, the other 25 must be sampled at most 20 cycles per second. For each value of $\bar{x}$, the minimum sampling reduction factor at first drops rapidly as $|\mathcal{O}|$ increases, and then levels off. As the number of operations that must be sampled at worst case frequency approaches 10, the initial drop-off becomes steeper.

Note that, in order to analyze the more general problem, we would need to model the additional resources required (*i.e.* RAM) on each processor LRM for storing additional flight operations. If necessary, this complication can be addressed by providing paging of flight operations themselves and, thus, does not change the basic tradeoffs.

[1]Note that, when $cf = 0$ the correct answer is ∞; except for $\bar{x} = 10$, when the first term should be interpreted $0/0 = 0$ such that the bound is 10.

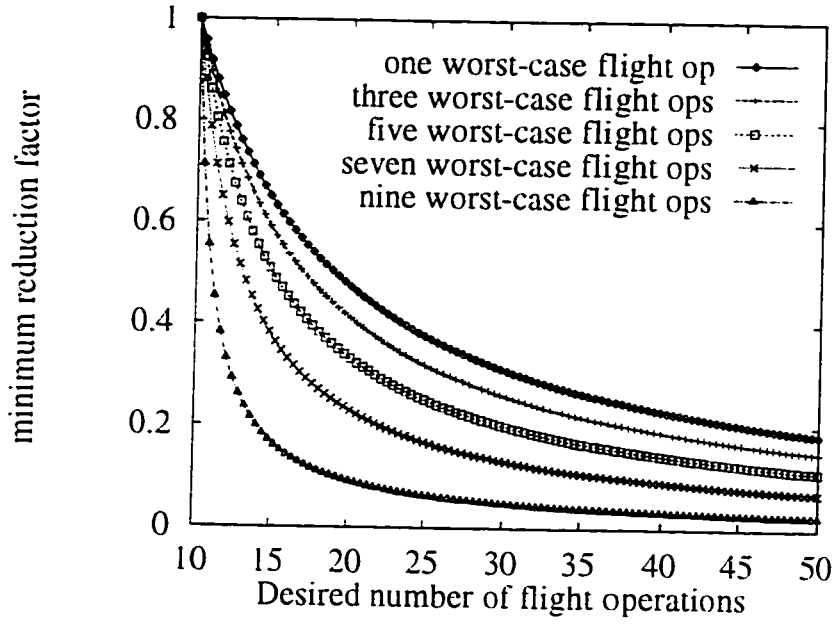

Figure 6.2: Minimum sampling reduction factor as function of desired number of flight operations, given that $\bar{x} \in \{1, 3, 5, 7, 9\}$ operations must be sampled at worst-case frequency.

To summarize, if we can show that a given in-flight environment does not have any situations for which more than $\bar{x}$ flight operations simultaneously require a sampling rate of 100, then we can potentially provide a conditional schedule solution that allows up to $|\mathcal{O}| = (10 - \bar{x})/cf + \bar{x}$ flight operations. Realizing this potential depends upon the ability to actually provide such a conditional scheduling solution within limited resources. The relationship between these parameters is summarized in Table 6.2.

Maximum Simultaneous Worst-Case Operations	Reduction Factor	Maximum Allowable Flight Operations
$0 \leq \bar{x} \leq 10$	$0 \leq cf \leq 1$	$10 \leq \lvert\mathcal{O}\rvert \leq (10 - \bar{x})/cf + \bar{x}$

Table 6.2: Test parameters

6.2 Schedule Server Resources and Conditional Schedule Resource Requirements

As mentioned in Section 6.1.2, in order to realize the potential increase in flight operations, we must show that a conditional scheduling solution is possible with limited resources. In the general conditional scheduling approach of Chapter 5, there are many degrees of freedom in specifying the details of a schedule server and associated paging solution.

We define a single test parameter, D^*, to capture the *overriding maximum number of schedules* that can be stored on a schedule server. Additionally, we model the execution space requirements of each schedule, $c_S \in C$, as a function of the number of flight operations sampled at worst-case frequency in the schedule. As given in the following definition.

Definition 6.1 *Let c_S be a conditional schedule drawn from the collection C specifying the minimum sampling frequencies for $|\mathcal{O}|$ flight operations. Let $\bar{r}_S$ be the number of flight operations sampled at worst-case frequency in schedule c_S. The* execution space *B_δ of c_S is $B_\delta(c_S) = 100 * \bar{r}_S + 100cf * (|\mathcal{O}| - \bar{r}_S)$.*

In order to employ a generic schedule table, we assume that all conditional schedules on the schedule server require as much space as the largest schedule on the server. We do not model any other resources required for paging. In particular, we assume that there is enough space on the schedule server to implement a paging circuit mapping any possible situation $s \in \mathcal{S}$ to an overriding maximum of D^* conditional schedules. For example, we might implement the paging circuit as a table with $|S|$ entries of $\log D^*$ bits each, requiring the current situation to be coded into $\log |S|$ address bits.

We further assume that the combined process of selecting a conditional schedule and transmitting the schedule over the bus can always be accomplished within the 1000 millisecond cycle duration. This requires that, in addition to the 1000 millisecond slices reserved for flight operation instantiations, there are additional resources in the generic schedule table to synchronize paging of conditional schedules over the bus. Note that, given that the generic schedule is linearly ordered,

we can synchronize the paging of slices of the conditional schedule for the next cycle with the execution of the conditional schedule for the current cycle. This eliminates the need for the additional buffering resources modeled in Chapter 3.

Figure 5.5 of Chapter 5 depicts computational mechanisms similar to those of our restricted model. In particular, the generic schedule table is divided into a fixed pattern of flight operation interrupts and data transfer commands. There are a fixed number of flight operation instantiations in any cycle and each interrupt points to a flight operation instantiation of equal-duration.

In order to study the tradeoff between the number of schedules stored on a schedule server and the size of the largest schedule, we must relate our models for B_δ and D^*. In Section 7.2 we provide a way to do so without introducing additional test parameters.

6.3 Generating Collections of Conditional Schedules and Environmental Coverage Requirements

We now introduce three test parameters to capture the number of conditional schedules available in a given collection, the number of requirements defining the in-flight environment and the relationship between the worst-case sampling guarantees of conditional schedules and the worst-case sampling requirements of in-flight environments.

6.3.1 Conditional Schedules

Define $|C|^*$ to be the total number of different conditional schedules that we could possibly pre-compile for a given test case. In any test case, we select $|\mathcal{O}|$ and cf, and bound $\bar{r}$ so that each schedule is executable in 1000 milliseconds, as shown

in Lemma 6.1. For a fixed $\bar{x}$ and $|\mathcal{O}|$, $|C|^*$ is calculated by the following formula:[2]

$$|C|^* = \sum_{0 \leq i \leq \bar{x}} \binom{|\mathcal{O}|}{i}$$

Define a *schedule generation factor* test parameter $sf \in [0.1]$ to be the fraction of the total possible schedules that are generated for a given test. We precompile a collection of conditional schedules by randomly generating $|C| = sf|C|^*$ schedules, each containing $|\mathcal{O}|$ flight operations with at most $\bar{x}$ operations being sampled at 100 hertz. These schedules are chosen uniformly from the space of all possible schedules.

6.3.2 Environmental Coverage Requirements

Given the problem-specific nature of developing a transition model and situation-specific conditional frequency model for a given flight environment, an empirical study can easily fall into the trap of providing too many degrees of freedom and not enough structure, or providing structure that does not adequately capture any particular problem. Rather than address these issues directly, we base our empirical study on varying the form and size of the coverage requirement, as given in Definition 5.2 of Chapter 5.

We capture a flight environment as a collection of coverage requirements $R_\nu = \{\nu(R_s^n)|s \in \mathcal{S}\}$. Given the simplifications of this empirical study, each element of this collection specifies the minimum required sampling frequency for each of $|\mathcal{O}|$ flight operations. This frequency can either be 100 or $100cf$ hertz.

We define two test parameters to model a collection of coverage requirements. The first parameter summarizes the difficulty of covering the requirements in the collection by bounding the number of flight operations that require worst case sampling frequency in any requirement. For a given test case, this parameter is expressed relative to the worst case bound $\bar{x}$ provided by Lemma 6.1. The second

[2]Note that this is the partial sum of a row of Pascal's triangle; for which there is no closed form solution. However, it may be calculated easily by summing the elements from 0 to $\bar{x}$ of the recurrence $|C|_i^* = (|C|_{i-1}^* * (|\mathcal{O}| - i + 1))/i$, $|C|_0^* = 1$.

parameter specifies the fraction of all possible requirements that are included in a collection.

Define the *worst-case requirement factor* $rf \in [0.1]$ such that the worst-case sampling requirement $\bar{x}_r = rf\bar{x}$. Define the *coverage requirement generation factor* $mrf \in [0.1]$ to be the fraction of total possible coverage requirements such that

$$|R_\nu| = mrf|R_\nu|^* = mrf \sum_{0 \le i \le \bar{x}_r} \left(\begin{array}{c} |\mathcal{O}| \\ i \end{array} \right).$$

We generate a flight environment by randomly generating a collection of $|R_\nu|$ requirements uniformly chosen from $|R_\nu|^*$ possible requirements.

6.3.3 Understanding Uniformly Generated Collections

Figure 6.3 depicts the growth in the total number of possible conditional schedules as the number of desired flight operations rises above 10. for various values of $\bar{x}$. As can be seen, this function increases very rapidly. The rapid growth indicates that the potential design-time problem difficulty increases rapidly with the number of flight operations. The collection of possible coverage requirements grows similarly.

We may better understand the relationship between sf, rf, mrf and the other test parameters by looking at the space of all possible subsets of up to $|\mathcal{O}|$ flight operations. In Figure 6.4 we depict a simplified example of only four flight operations. There are 16 possible sets of these four flight operations. As can be seen, the sets are related by a lattice in which lower levels contain fewer flight operations per set than upper levels, thus lower levels are subsets of upper levels. Set number 16 contains all four operations and set number 1 is the empty set. Arrows indicate level-by-level containment, such that a path from node j to node k indicates that node j is a subset of node k.

Figure 6.5 depicts an example test case of four flight operations in which the test parameters are $|\mathcal{O}| = 4$, $sf = .375$, $rf = .5$, and $mrf = .36$. A collection of $6 = sf * 16$ schedules of at most four flight operations is uniformly generated. The maximum number of coverage requirements with at most $2 =$

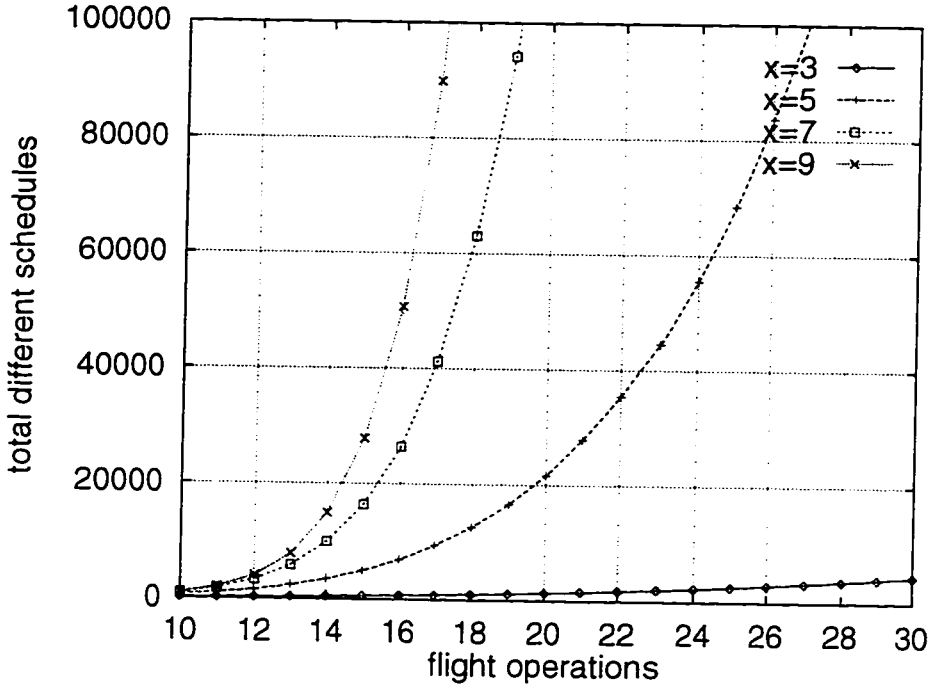

Figure 6.3: Growth in total possible schedules as function of number of desired flight operations

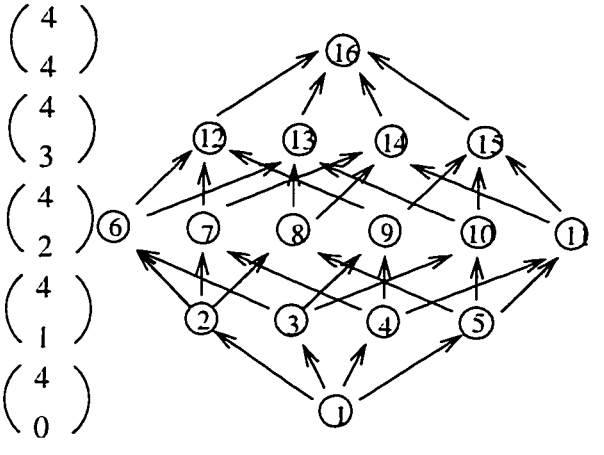

Figure 6.4: All possible sets of at most four flight operations

$rf * 4$ operations is 11. A collection of $4 = mrf * 11$ coverage requirements of at most two flight operations is uniformly generated. The flattened collection of requirements contains two sets. Finally, there are four schedules that cover at least one of the flattened coverage requirements. Depending on the values of test parameters cf and D^*, one or more of the coverings A, B and C may provide an executable solution.

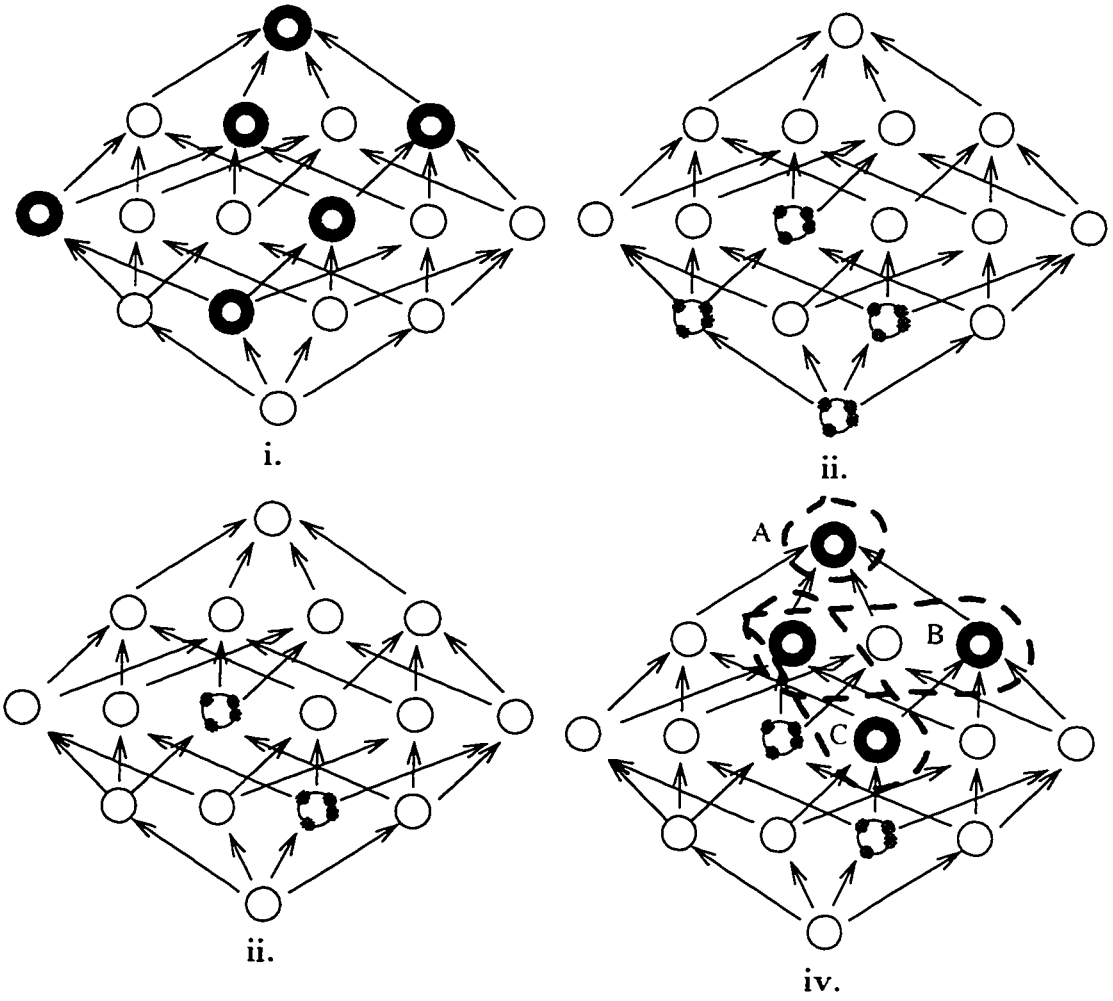

Figure 6.5: An example test case: i. uniformly generated schedule collection of size $6 = sf * 16$, $sf = .375$; ii. uniformly generated requirements collection of size $4 = mrf * 11$, $mrf = .36$, each requiring at most $2 = rf * 4$, $rf = .5$, worst-case frequency flight operations; iii. flattened collection of requirements; iv. schedules that cover at least one flattened requirement, and three possible coverings.

The minimum d over each schedule in a given covering must be at least as

large as the number of schedules in the covering in order for the solution to be executable. The formula for calculating d given in Definition 6.1 specifies that d is the same for each row of the lattice of Figure 6.5. Including schedules drawn from upper rows allow fewer pages (smaller d) in an executable solution than including schedules from lower rows.

These figures provide an intuitive understanding that the ability to provide a covering depends upon the relationship between the number of schedules in a collection $sf|C|^*$, the number of requirements in a collection $mrf|R_\nu|^*$, and the highest row from which coverage requirements are drawn $\bar{r}_r = rf\bar{r}$. The ability to provide an executable covering additionally depends upon the test parameters cf and D^*.

6.4 Discussion

Much of the development in this chapter is an attempt to provide a coherent study despite the lack of existing models of the environment and resources suitable for conditional scheduling. Development of such models would allow an empirical study of a specific application environment and architecture.

Chapter 7

Results of Empirical Evaluation

The empirical study developed in Chapter 6 specifies six test parameters to capture relationships between collections of conditional schedules, collections of environmental requirements and bounds on hardware resources. By testing various combinations of these parameters we can explore the space of conditional avionics scheduling problems.

In this chapter we specify a design-time algorithm for solving test cases. This algorithm, presented in Section 7.2, is a specific instantiation of the analytical techniques developed in prior chapters. By applying this algorithm to a subset of the test space of conditional avionics scheduling problems, we may attain an empirical evaluation of the practicality of the techniques of this thesis. In Section 7.1 we summarize the test cases of this study, and in Section 7.3 we provide the results of this empirical evaluation.

7.1 Test Plan

In this section, we summarize the test cases and testing procedure for this empirical study. There are six primary test parameters developed in Chapter 6. In this study, we choose a specific set of values for these test parameters. These values are chosen manually in an attempt to locate regions of the test space containing difficult design-time problems. These six parameters are summarized in

Tables 7.1 and 7.2. Each test case consists of evaluating one point within this six dimensional space.

Flight Operations	Sampling Frequency Reduction Factor	Maximum Addressable Conditional Schedules
$\lvert\mathcal{O}\rvert \in$ $\{10, 12, \ldots, 20\}$	$cf \in \{.05, .1, .25, .5, .75, .9\}$ $\leq (10/\lvert\mathcal{O}\rvert)$	$D^* \in$ $\{10, 12, \ldots, 20, 50, 100, 200, 300\}$

Table 7.1: Test space - I

Schedule Generation Factor	Worst-Case Requirement Factor	Coverage Requirements Generation Factor
$sf \in \{.01, .1, .2, .4\}$	$rf \in \{.25, .5, .6, .75, .9\}$	$mrf \in \{.01, .1, .2, .4\}$

Table 7.2: Test space - II

Given an assignment of values to the test parameters, we may derive calculated parameters of interest. These calculations are summarized in Tables 7.3 and 7.4.

Maximum Operations Sampled at Worst-Case Frequency	Total Number of Possible Executable Conditional Schedules
$0 \leq \bar{x} = \lfloor \frac{10 - \lvert\mathcal{O}\rvert cf}{1 - cf} \rfloor \leq 10$	$\lvert C\rvert^* = \sum_{0 \leq i \leq \bar{x}} \binom{\lvert\mathcal{O}\rvert}{i}$

Table 7.3: Calculated parameters - I

The various combinations of test parameters lead to test cases that range from a handful of conditional schedules and requirements (columns and row) to tens of thousands of each. Various relationships between the number of schedules, number of requirements, number of worst-case flight operations provided or required, resource bounds, and desired maximum flight operations are represented.

A test case consists of the following steps:

Maximum Operations Simultaneously Sampling at Worst-Case Frequency	Total Number of Possible Coverage Requirements
$\bar{x}_r = rf\bar{x}$	$\lvert R_\nu \rvert^* = \sum_{0 \le i \le \bar{x}_r} \binom{\lvert \mathcal{O} \rvert}{i}$

Table 7.4: Calculated parameters - II

1. Select values for parameters $|\mathcal{O}|, cf, D^*, sf, rf$, and mrf.
2. Calculate values for parameters $\bar{x}, |C|^*, \bar{x}_r$, and $|R_\nu|^*$, and
3. For some number of iterations (usually 10):
   (a) uniformly generate a random collection of $sf|C|^*$ conditional schedules.
   (b) for each schedule c_S, calculate the page bound d_S (as described in Section 7.2).
   (c) uniformly generate a random collection of $mrf|R_\nu|^*$ coverage requirements.
   (d) flatten the collection of coverage requirements.
   (e) reduce the collection of schedules to only those that cover some requirement.
   (f) generate an executable set cover problem.
   (g) solve the executable set cover problem (using the algorithm of Section 7.2).

In the process of solving the executable set cover problem, we record information about whether or not an executable covering was found, the total running time, running time per algorithm component, and other measures. From these measures we summarize over all test problems the fraction of tests that were

feasibly solved, fraction that were infeasible, fraction that were not solved, the average running time of solved tests, etc.

Implementation Note: When fractional, $\bar{x}$ and $\bar{x}_r$ are interpreted as ceilings and thus, rounded down: $|C|$ and $|R_\nu|$ are rounded to nearest integer. $\bar{x}_r$ is computed from the rounded version of $\bar{x}$.

7.2 An Instantiation of the Executable Set Cover Solution Scheme

In Chapter 4 we develop design-time analytical techniques for evaluating on-line decision-making solutions. The techniques are combined into a solution sketch that requires an enumeration scheme or search algorithm in order to be completely specified. In this section, we provide an enumeration scheme by using the simplifications of Chapter 6 to combine executability equations and covering constraints into a single mixed integer linear program (MILP). Additionally, we augment the sketch of Chapter 4 with a few additional implementation details.

7.2.1 Algorithm Sketch

The algorithm used to solve executable set cover problems in this study may be sketched by the following steps:

1. Evaluate whether or not any set cover exists.
2. Look for trivial executable solutions.
3. Iterate LP until a feasible solution of size less than or equal to D^* is found, or we prove that no such solution exists, stop if solution is integer.
4. Execute Greedy-Set-Cover, remove infeasible schedules using analytical bound, stop if executable solution is found.
5. Execute MILP to search for an exact solution.

Executable set cover problems are generated by comparing the collection of flattened requirements to the reduced collection of schedules. This process provides a simple method for determining whether or not a set cover exists, and whether or not executable set coverings are trivial to find. This method is polynomial in the size of both collections. Since the executable set cover algorithm components are not invoked, any problem solved during this step is designated as solved on "Input."

Implementation Note: Although polynomial in complexity, much work goes into generating executable set covering problems. However, since this work is constant for all problem instances of the same size, it is not recorded as part of the computational load of the problem.

7.2.2 Simplified Executability Equations

The simplifications of this empirical study lead to simplified resource equations that allow the incorporation of resource tradeoffs directly into an MILP. By doing so, we may then use standard branch and bound enumeration schemes to instantiate the executable set cover solution sketch.

In Chapter 6 we specify a schedule server model that does not model any resources for paging itself: choosing instead to assume a fixed amount of resources set aside for paging. By fixing the paging resources of the schedule server, we need only consider the overriding bound on conditional schedules on the server D^*, and a space tradeoff relating the number of conditional schedules on the server, the size of the largest schedule, and the available space on the schedule server M. Furthermore, we only generate random conditional schedules that are individually executable as specified in Lemma 6.1.

An executable conditional scheduling solution $C' \subseteq C$ must satisfy the following simplified executability equations.

$$|C'| \leq D^* \tag{7.1}$$

$$|C'| * B_\delta^*(C') \leq M \tag{7.2}$$

Where $B_\delta^*(C')$ is the maximum number of bits of any conditional schedule $cs \in C'$.

To avoid introducing a new test parameter, we define M in terms of other test parameters. Observe that M is closely related to D^*. In particular, the maximum allowable number of conditional schedules occurs when we have $D^* * B^*_\delta(C') \leq M$. We set $M = |\mathcal{O}| * 100 * cf * D^*$. Under this definition, when we construct a solution with the maximum allowable number of conditional schedules, $|C'| = D^*$, the only conditional schedules we can include are those with all operations sampled at the reduced sampling frequency *i.e.* the smallest possible conditional schedule. This normalization provides that increasing the overriding maximum D^* always affects the solution tradeoff.

We conclude Chapter 5 noting that, in order to guarantee safe behavior, we must both show that the conditions of Lemma 5.1 are satisfied and that the computational mechanisms to perform paging can be accommodated in the limited time and space resources. We may now provide conditions under which the latter holds.

In particular, we have reduced the executability conditions to a single space tradeoff. This tradeoff relates the number of conditional schedules in a solution to the size of the largest schedule in a solution. Under this simplification, we can pre-compute this constraint for each given conditional schedule. The pre-computed value gives an upper bound $d_S \leq D^*$ on any conditional scheduling solution $C' \subseteq C$ that includes c_S. This modification is captured in the following executability lemma.

Lemma 7.1 *Let* $0 \leq B^*_\delta(C) \leq 1000$ *be the maximum number of bits of any conditional schedule* $c_S \in C$.

Given a collection of conditional schedules C, *overriding maximum pages/server space* D^*, *number of flight operations* $|\mathcal{O}| \geq 10$, *worst-case sampling frequency* $\bar{x} = 100$, *and conditional sampling frequency* $0 \leq cf \leq 1$, *such that*

$$|\mathcal{O}| \leq \frac{10 - \bar{x}}{cf} + \bar{x}$$

a collection of conditional schedules $C' \subseteq C$ *is* executable *if and only if*

$$|C'| \leq \min_{\{c_S \in C'\}} d_S.$$

where $d_S = (|\mathcal{O}| * 100 * cf * D^*)/B_\delta(c_S)$, *for each* $c_S \in C$.

7.2.3 Deriving an MILP That Combines Set Cover Constraints and Executability Equations

Due to the simplifications of this study, we can incorporate the tradeoffs of the executability conditions directly into an augmented set cover mixed integer linear program (MILP). Standard techniques for solving this MILP instantiate the structured enumeration of the algorithm of Chapter 4, Figure 4.5.

We use the pre-computed values d_S, for each $c_S \in C$, provided by Lemma 7.1 as coefficients in the desired MILP. This replaces the need to jointly evaluate the executability equations for collections of schedules and find the minimum allowable number of schedules over each independent schedule in the set.

We incorporate this minimization process by using a standard linear program formulation technique based on **if-then** constraints [69]. Roughly, an if-then constraint captures a relationship between constraints such that, if the first constraint is satisfied then the second constraint must be satisfied. However, the implication does not necessarily apply in the opposite direction.

For our purposes the if-then constraint is: if the set corresponding to conditional schedule c_S is included in the solution, then the total number of schedules in the solution must not exceed d_S. Briefly, if Z_S is the 0-1 variable corresponding to whether or not conditional schedule (set) y_S is included in the solution, and $\sum_{\{y_S \in Y\}} Z_S$ is the number of schedules included in the solution, then the desired if-then constraint is: if $Z_S > 0$ then $\sum_{\{y_S \in Y\}} Z_S \leq d_S$.

We first add a variable $d \leq D^*$ to the MILP to capture the total number of conditional schedules in a solution. We then replace the objective function with a new objective to minimize d and add the constraint $\sum_{\{y_S \in Y\}} Z_S \leq d$. Given that constant D^* is the largest possible value that d can achieve, we derive if-then constraints of the form

$$d - d_S - D^* + D^* Z_S \leq 0$$

for each $y_S \in Y$. If $Z_S = 0$ this constraint is always satisfied. However, if $Z_S = 1$, this constraint may only be satisfied if $d \leq d_S$ *i.e.* if the number of schedules in the solution does not exceed d_S.

As a further simplification, we need only add if-then constraints for conditional

schedules for which $d_S < D^*$. Thus, our augmented MILP adds one additional column and at most $|Y^*| + 1$ additional rows to MSCIP. The resulting MILP is given in Definition 7.1.

Definition 7.1 *For any set covering problem described by elements X and collection Y^*, let variable $Z_S = 1$, if set $y_S \in Y^*$ is in the covering Y', and 0, if not. Furthermore, let constant $W_{r',S} = 1$, if $r' \in y_S$, and 0, if not. $\forall r' \in X, y_S \in Y^*$. Finally, for each $y_S \in Y^*$, $d_S = (|\mathcal{O}| * 100 * cf * D^*)/B_\delta(c_S)$.*

$$
\begin{array}{rrcl}
 & Z^I = \min d & \textit{such that} & \\
 & \sum_{\{y_S \in Y\}} Z_S & \leq & d \\
\textit{for each } r' \in X\textit{:} & \sum_{\{y_S \in Y\}} W_{r',S} Z_S & \geq & 1 \\
\textit{for each } y_S \in Y^* \textit{ and } d_S < D^*\textit{:} & & & \\
 & d - d_S - D^* + D^* Z_S & \leq & 0 \\
 & 0 \leq d & \leq & D^* \\
\textit{for each } y_S \in Y^*\textit{:} & Z_S & \in & \{0,1\}
\end{array}
$$

MSCIP augmented to incorporate simplified executability equations (MSCMILP)

Standard branch and bound techniques may be used to employ MSCMILP as the structured enumeration scheme of the algorithm of Figure 4.5. This provides an exact algorithm to solve the arbitrary-circuit paging problem as applied to the restricted version of real-time conditional avionics scheduling. Theorem 7.1 provides this result and may be used to determine the existence of a real-time conditional scheduling solution that guarantees safe behavior.

Theorem 7.1 *MSCMILP is a correct and complete enumeration scheme that finds an executable conditional scheduling solution or shows that no executable conditional scheduling solution exists.*

7.3 Results of the Empirical Study

The following empirical results demonstrate that the framework and techniques of this thesis may be practically applied to the analysis and design of on-line

decision-making solutions. Furthermore, they demonstrate that positive results (executable solutions) are attainable over a non-trivial fraction of test cases. Finally, practical application and positive results are maintained as the number of flight operations are increased. Though, there is some concern that results are skewed at the far end of this increase.

The empirical results are divided into a preliminary round to explore the test space, and a subsequent round that focuses on more difficult test cases.

7.3.1 Overall Results of Preliminary Round of Test Cases

To better understand the space of test points, we performed an initial round of testing in which we manually chose test points in what we expected to be difficult regions of the test space. The results of this round are provided in Table 7.5. A test case is either shown not to have a covering ("No Set Cover"), shown to have a covering but not one that can be executed on the parameterized hardware ("Not Executable"), shown to have an executable solution ("Executable"), or shown to break the algorithm for various reasons ("Undetermined"). The categorization of test outcome is listed by row and the final four columns indicate in which component of the algorithm the result was achieved.

Status	Cases	Fraction	Avg Time (cpu secs)	Frac Input	Frac LP	Frac Greedy	Frac MILP
All	5299	1	55	.73	.15	.11	.02
No Set Cover	1391	.26	0	1	0	0	0
Not Executable	598	.11	64	0	.99	0	.01
Executable	3164	.60	5	.782	.041	.177	.001
Undetermined	145	.03	1635*	0	.35	0	.65

*Forced termination at 1800

Table 7.5: Overall results of preliminary round of test cases

A surprising result of this round of testing is that 73 percent of the 5300 cases tested were solved on "Input" (during point generation). Of the remaining 27

percent. 15 percent were solved by the linear programming relaxation of the minimal set cover integer program and 11 percent were solved the Greedy-Set-Cover algorithm. Only 2 percent of these test cases even made it to the enumeration stage.

These results indicate that this round of testing did not find many hard problems. The design-time analytical techniques were very effective here, conclusively showing that a test case either has or does not have an executable solution in 97 percent of the cases. This result is further strengthened by the fact that many of the test cases consisted of large problems of thousands of schedules and/or requirements.

The mix of outcomes in this preliminary round provided that 60 percent of the cases resulted in executable solutions. In other words, a majority of the tested conditional real-time avionics scheduling problems could be solved by the arbitrary-circuit paging architecture. This demonstrates that the techniques of this thesis are of practical design interest in these test cases. An additional 11 percent were proven not to have executable solutions while 26 percent of the cases had no covering at all. In other words, in a quarter of the cases there existed requirements of the environment that could not be handled by any conditional schedule in the collection, regardless of schedule server resources.

While the average time to find a solution was 55 seconds, this figure has little meaning given the large variance of execution times and the forced termination at 1800 seconds. Furthermore, public domain code for LP and MILP solution was employed. This code is orders of magnitude slower than the best commercial codes, especially in the larger test cases.

Given the abundance of "easy" test cases and positive solutions, we were concerned that this round of testing did not adequately represent the test space. We devised a second, more difficult round of testing.

7.3.2 Overall Results of a More Difficult Round of Test Cases

In this round of testing, we attempted to find a more difficult region of the test space by filtering out the sets of parameters corresponding to the easiest problems. In particular, we removed cases with very few requirements or conditional schedules, as well as cases for which the ratio of schedules to requirements or requirements to schedules was large. Finally, we removed cases in which the collection of schedules rose above 100000 or collection of requirements rose above 10000. From our experiences in the initial round, these cases exhaust the capabilities of our hardware/software.

The overall results of this round of testing are provided in Table 7.6.

Status	Cases	Fraction	Avg Time (cpu secs)	Frac Input	Frac LP	Frac Greedy	Frac MILP
All	83973	1	218	.60	.29	.09	.02
No Set Cover	33741	.40	0	1	0	0	0
Not Executable	17664	.21	176	0	.99	0	.01
Executable	28022	.33	16	.589	.127	.280	.004
Undetermined	4522	.05	3262*	0	.77	.00	.22

*Forced termination at 3600

Table 7.6: Overall results of the more difficult round of test cases

These results indicate that, while we did manage to find a more difficult region of the test space, the majority of cases were still very easy to solve. In particular, 60 percent of the cases were solved at the point generation stage. This is down from the 73 percent of the initial round. However, the cases solved by the LP increased from 15 to 29 percent. Adding the 9 percent of cases solved by the Greedy algorithm yields 98 percent of test cases solved prior to enumeration. This is the same proportion as that of the initial round of testing. This provides additional evidence that the analytical techniques of this thesis can be effectively applied.

The biggest change from the initial round is that the number of test cases for

which an executable solution was found decreased from a majority of 60 percent to 33 percent. The number of cases for which no set cover existed increased from 26 to 40 percent. This indicates that we were successful in finding a more difficult region of the test space. However, we still attained a non-trivial 33 percent of cases for which an executable solution could be constructed. This provides further evidence that the arbitrary-circuit paging architecture is of practical design interest.

The results of both the initial round and the more difficult round of testing indicate that hard test cases are not easy to find. We may have completely missed large pockets of difficult problems with our limited testing. However, we conjecture that this particular empirical study does not have such pockets. We do not, however, draw any conclusions about other empirical studies or real-world scheduling problems from these limited cases.

7.3.3 A Closer Look at the Computational Issues of Increasing Functionality through Conditional Scheduling

In order to obtain an empirical understanding of the ability of the conditional real-time scheduling approach to improve upon traditional static solutions, we further categorize the results of the second round of testing in terms of the number of desired flight operations. In Chapter 6 we provide an analytical understanding of this potential improvement in terms of relationships between test parameters. This closer look provides an understanding of the computational issues of this approach to real-time avionics solution design.

In Figure 7.1, we show how the fraction of each outcome changes as the number of desired flight operations increases from 10 to 20. Recall that the worst-case static schedule supports 10 flight operations. A general trend in this graph is that the fraction of test cases for which executable scheduling solutions are found increases with increasing desired flight operations. Conversely the fraction of unsolvable test cases decreases.

Looking more carefully, we see that this relationship is actually reversed as

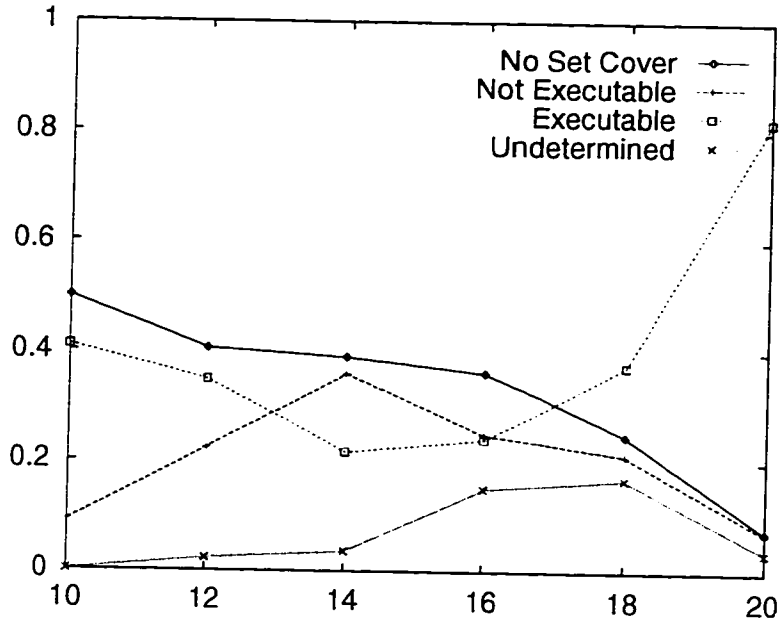

Figure 7.1: Fraction of tests achieving each outcome as function of increased functionality (number of supported flight operations)

the flight operations are increased from 10 to 14. This indicates that the mid-range between 10 and 20 provides the most difficult test cases. This trend is further substantiated in Figure 7.2, in which we note that the fraction of executable solutions that are "easy" to determine (found during point generation) and those which require additional computation has a minima (maxima) at 14 flight operations.

One explanation for this preliminary result can be given in terms of the combinatorial structure of uniformly distributed collections given in Chapter 6. In this empirical study, over all test cases, there is less structure in the relationship between collections of schedules and requirements in this mid-range of values. In other words, when the majority of schedules and requirements are drawn from the center of the lattice (as is the case for desired flight operations between 14 and 16), the collections appear as if they were drawn from a flat distribution rather than a structured lattice. Further study, beyond the scope of this dissertation, would be required to substantiate this intuition.

Finally, we note that the trends at the far end of the graphs of Figures 7.1

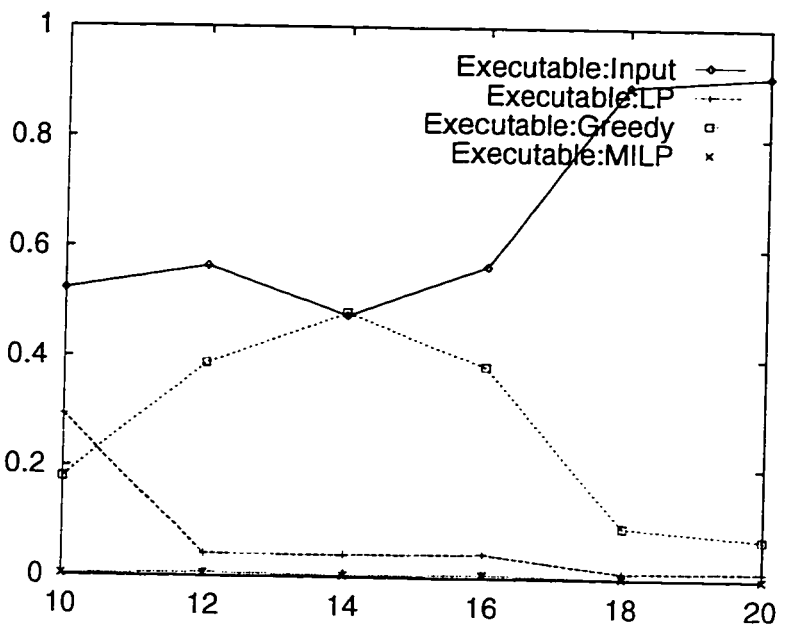

Figure 7.2: Fraction of tests shown executable by each algorithm component

Maximum Operations	10	12	14	16	18	20	Total
Actual Cases Tested	24310	21666	15058	13042	8916	981	83973
Average Run Time	.03	81.24	200.64	600.62	630.98	77.94	218
Worst-Case Run Time	93.7	3600	3600	3600	3600	1780	3600

Table 7.7: Evidence of result skew as flight operations approach 20

and 7.2 must be taken "with a grain of salt" due to a skew in the distribution of test cases with larger numbers of flight operations. This trend is seen in Table 7.7 and Figures 7.3 and 7.4.

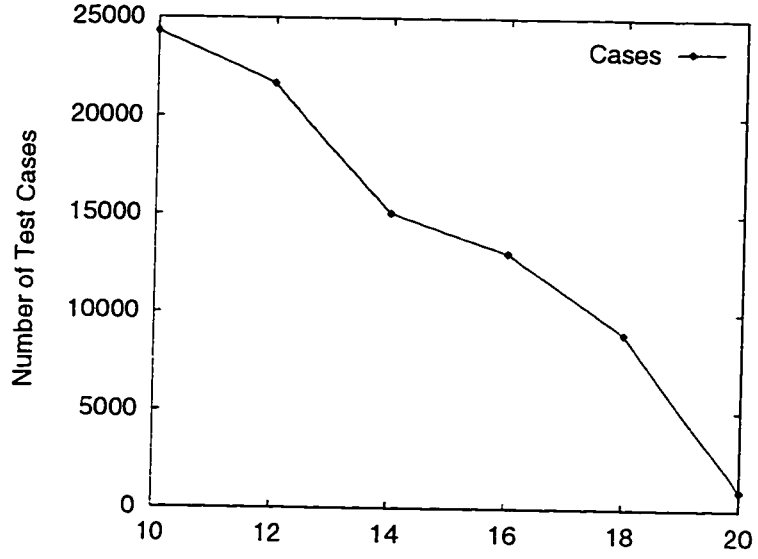

Figure 7.3: Decrease in distribution of test cases as the number of desired flight operations increase

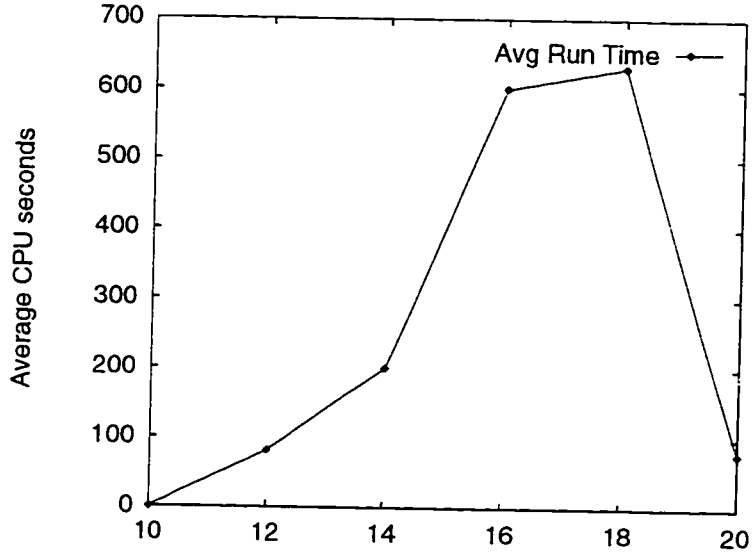

Figure 7.4: Abrupt change in average run time at the high end of desired flight operations

This trend has an interesting explanation. The focus of this empirical study is

on the practical effectiveness of applying the analytical techniques of this thesis. This is motivated by the asymptotic complexity of solving these problems, in general. Little concern was given to the case generation phase, which can be accomplished by polynomial methods. However, applying naive algorithms to case generation is shown inadvisable in this study. In particular, the number of possible schedules and requirements grows very rapidly with an increase in desired flight operations, as seen in Figure 6.3. Since this empirical study is designed in terms of fractions of all possible schedules and requirements, the test cases corresponding to the far end of desired flight operations are predominantly very large. Too large, in fact, for the majority of points to be generated using naive algorithms on available hardware.

7.4 Discussion

The results of this empirical study provide encouraging evidence that the techniques of this thesis may be applied effectively. Both in terms of the computational efficiency in practice, and the fraction of test cases for which executable solutions may be generated. However, we would not draw any general conclusions from such limited testing. Results on real-world problem data are required.

While it may be possible to discern additional relationships between problem parameters by a more careful examination of the empirical results, such relationships could only be claimed with respect to the empirical study itself. We concluded that this class of results would not be of compelling interest.

Chapter 8

Related Research

The techniques and applications of this thesis are related to work in many fields, including control theory, decision theory, operations research, systems engineering, static analysis, and real-time scheduling. In this chapter, we analyze the contributions of this dissertation in the context of existing literature. Most directly, this thesis is related to work in artificial intelligence that attempts to understand the tradeoffs involved in resource-bounded problem solving, and work on designing on-line decision-making systems to solve problems in specific domains.

8.1 Resource-Bounded Problem Solving

The interaction of bounded resources and problem solving brought together research communities originally devoted to each, including computer scientists and decision theorists. Early research at this origination point may be seen in the work of Good [30] on models of rationality that take deliberation costs into account.

One approach to resource-bounded problem solving is to consider the implications of problem solving with algorithms that are divisible into incremental units. More or less of these increments of computation can be applied to a problem, depending upon the available resources. Early work in this area include Simon and Kadane [67], who consider solving search problems by quantizing deliberation

into chunks.

Etzioni [21] considers a similar model in which the deliberative chunks correspond to different problem solving methods. Lesser [49] uses the term *approximate processing* to capture a similar notion in the domain of time-critical problem solving. In work on *design-to-time scheduling*, Garvey and Lesser [25] model the task structure of solution methods and their dependence, duration and quality. They use this structure to find execution methods that optimize quality for the available time.

In work on *anytime algorithms* Dean and Boddy [15, 7] consider a class of algorithms that achieve various levels of solution quality as a function of the allotted processing time. This relationship is captured by *performance profiles*. Liu *et. al.* [65] use the term *imprecise computation* in the context of real-time operating systems to refer to a computation that has both a necessary and an optional component. Whether or not the optional component is executed is resource-dependent.

One factor that differentiates work on divisible problem solving methods is the point a which the allocation of resources takes place. Horvitz [42] uses what he calls *flexible computations* to allocate computational resources at run time. Horvitz [43] constructs time-dependent utility models and models dynamically changing computation costs to determine on-line when to halt computation and take action. He also uses design-time analysis to indicate when on-line meta-reasoning is valuable.

Russell and Wefald [60, 61] describe a general approach in which deliberation is considered as a sequence of *inference steps* leading to the performance of an action. They explicitly compute the expected value of different sequences of inference steps using a technique similar to Howard's method for computing the expected value of information [44]. Approximations are used to minimize the resources required for on-line meta-deliberation.

There is a natural progression from solving a single problem with a flexible algorithm to solving sequences of problems by combining multiple algorithms or instances of the same flexible algorithm. Thus, we might consider these methods

with respect to on-line decision-making problems.

Dean and Boddy [15, 7] combine anytime algorithms by *deliberation scheduling*, to explicitly account for the costs and benefits of computational delays over time. In this work, a precursor event indicates to the deliberation scheduler a sequence of future events that need to be addressed. This either requires that deliberation scheduling be performed at run time, after the precursor event is observed, or deliberation schedules be pre-compiled at design time for expected sets of precursor events.

Zilberstein [70] discusses the problem of how anytime algorithms for complicated computations can be *compiled* at design time. Compiling anytime algorithms is the process of combining multiple algorithms with separate profiles into a single module with a new profile. He represents dependencies between anytime algorithms through performance profiles that are conditioned on various input parameters and their values. Additionally, he introduces the concept of *contract algorithms* as the combination of an anytime algorithm and a fixed allocation of resources.

Both Zilberstein [70] and Boddy and Dean [15, 7] analyze the complexity of deliberation scheduling/compilation with respect to classes of performance profiles. In general, when performance profiles are represented in closed form the compilation problem involves solving differential equations; when represented in tabular format the compilation problem becomes a search problem. Compilation problems in which the problem instances are independent are considerably easier than those in which the problem instances are dependent.

Recently, Russell and Subramanian [59] have presented *bounded optimality* as a general method for constructing intelligent agents. While a very general way to formulate problems, technical results thus far primarily divide an on-line decision-making problem into a series of independent episodes, each of which may be solved by the optimization techniques of [60].

Much of the effort involved in understanding resource-bounded problem solving is aimed at formulating decision-making problems so that the important resource tradeoffs can be evaluated. In this context, understanding tradeoffs include characterizing the computational options available to the decision maker. A primary contribution of this dissertation is a framework for formulating decision-making problems in which these tradeoffs are made precise. Additionally, the connection between resource tradeoffs and dynamics is explicitly modeled and analyzed.

The framework for understanding on-line decision-making presented in this dissertation differs from prior work in resource-bounded problem solving fundamentally in the approach to understanding decision-making behavior. In particular, we attempt to provide design-time guarantees of the behavior produced by the interaction of dependent sequences of actions with a dynamic, uncertain environment.

A common feature of the literature described above, is that behavior is abstracted into measures of quality. These measures of quality are manipulated with respect to partial computations or sequences of computations over time. Implicit in the summarization of behavior by numerical measures is the view that actions achieving similar measures have similar value. And, thus, it is possible to look at the maximization of this value in order to understand behavior.

However, when considering dependent sequences of actions, early actions have both immediate value and a sometimes subtle affect on the future trajectory of the system. While, in principal, dynamic programming methods can be used to optimize the value in these problems, we must also consider the affects of bounded resources on value. Modeling this combination of effects as a maximization problem is not necessarily an effective treatment. The view of this dissertation is that such combinations of effects are more effectively understood by a model of the possible trajectories. Without such understanding, behavior can only be well-understood under restrictive independence assumptions.

Work on deliberation scheduling and bounded optimality either focuses on

tradeoffs within a single deliberation interval [60, 42] or over sequences of independent deliberation intervals [15, 7, 59]. Zilberstein's work [70] is a notable exception in that dependence across deliberations is explicitly considered. However, without a model of uncertainty the behavior of these dependent sequences of actions cannot be fully understood.

A further contribution of this dissertation is to demonstrate that it is not always sufficient to optimize decision-making independently from the environment in which it is applied. As a consequence of this change of perspective, the analytical techniques of this dissertation are dissimilar and complementary to those of previous work on resource-bounded problem solving.

8.2 Reactive Control Policies

Reactive control policies are important components of on-line decision-making solutions. In the framework of this dissertation, reactive control policies are one particular form of decision-making circuit. Thus, understanding the limitations of the behaviors achievable with reactive control are an important step toward understanding the potential capabilities of decision-making circuits.

Reactive control takes the view that computational resources are too tightly constrained for any non-trivial on-line deliberation. This class of work attempts to compensate by pre-compiling any deliberation at design time. Execution-time behavior then consists of the implementation of a reactive control policy. In this section we consider work in artificial intelligence on pre-compilation of reactive behaviors and related work in control theory on optimal control laws.

Reactive behaviors One of the earliest uses of pre-compilation of actions is Nilsson's [22, 54] work on *triangle tables*. In this work, traditional plans are converted into sets of state-operator pairs that may be accessed in any order (independent of original plan sequence). Schoppers [62] generalizes this to the notion of *universal plans* in which planning continues until actions have been computed for all possible distinguishable situations. A universal plan is constructed prior to execution, for a particular domain with a given goal, and then executed

without modification. Schoppers [63] argues that production systems are similarly constructed and provide actions that are simply reactions to contents of working memory. Similarly, work has been done by Barto, Bradtke and Singh [3] on dynamic programming approaches that compile planning results into reactive strategies for real-time control.

The primary argument for this approach [62] is that by performing all reasoning prior to execution, time-critical reaction can be achieved even in the most complex, uncertain situations. The primary argument against this approach [29] is the enormous costs of anticipating all possible situations. Some have argued that pre-compilation of actions is appropriate in certain domains. Agre and Chapman's [1] research explores some domains that are effectively solved by pre-compiling behaviors into pure reaction. Kaelbling's work [48] compiles behavior into constant time-delay circuits. However, the timing guarantees are behavior-driven and do not necessarily reflect the limitations of a given execution environment.

Nilsson [55] in his *teleo-reactive* programs introduces another approach to pre-compilation of actions in which he allows for run-time binding of parameters and run-time compilation of action circuits. He argues that the advantage of run-time compilation is that it reduces the amount of circuitry that must be produced. In a sense, run-time compilation is a way to enforce that the system performs pre-compilation of actions as temporally close as possible to when those actions may be needed, without actually performing reasoning during execution (or providing a specific temporal relationship between decision-making time and action-execution time). All information known about the problem just prior to execution time can be used to compile action circuits for the problem. The effect is that only the circuits most relevant to the current problem are instantiated. Where relevance is defined in terms of initial conditions and not in terms of any assumptions about the environment itself.

Much work on pre-compilation of reactive behaviors make limiting assumptions about the interaction of control system and environment. For example, both Nilsson [55] and Schoppers [62] assume deterministic effects when planning

actions for nondeterministic environments. These assumptions may lead to unintended interactions between actions.

Control The literature on constructing control policies for achieving guaranteed behavior is vast. We focus here on one approach that is particularly relevant to the time-critical control system explored in this thesis. Namely, the work of Blanchini *et. al.* [5, 6] on feedback strategies to control periodic systems.

This work looks at resource distribution problems with periodic demand and unknown disturbances. Briefly, they use design-time analysis of the known periodic demand functions and a model of disturbances to figure out whether or not a control strategy may be generated. A control strategy must guarantee that storage levels are maintained within specified ranges.

If achievable, an optimal control strategy is designed for a subset of the state space. Namely, those states contained in a special sequence of sets of states referred to as the *target tube*. Additionally, a fast feedback strategy is included for states outside the target tube, such that the system is controlled back toward the target tube in a finite number of steps. The idea is that the target tube represents the known periodic structure of the problem and the closed-loop feedback strategy is intended to handle perturbations, disturbances and inaccuracies in the model.

We can compare this to the work of this dissertation by considering the combination of optimal control strategy for states in the target tube and fast feedback for the remaining states to be equivalent to the interaction of slow deliberation and fast reaction. Consistent with this thesis, the slow deliberation component is analyzed at design time and compiled into a fast control strategy that specifies the flow controls required maintain the storage levels in their allowable ranges.

The target tube represents a temporal pattern of situations for which guaranteed control may be achieved. Namely, a sequence of reachable states in each time step that is invariant to the period. The existence of controllable behavior is determined with respect to a known set of initial conditions.

There are complications involved with applying this approach in practice that

we won't discuss further. However, we include this work here as a example of techniques in control theory that may be applied toward generating the types of circuits required for solving on-line decision-making problems using the framework of this thesis. This form of analysis must be augmented with resource-specific constraints in order to be adapted to on-line decision-making solutions.

8.3 Deliberation/Reaction System Design

In addition to theoretical work on resource-bounded problem solving and reactive control behavior, there is a considerable literature on designing systems that integrate deliberation and reaction. In this section we first touch upon a handful of representative deliberation/reaction systems and then discuss two specific approaches to designing such systems.

Combined deliberation and reaction Chapman [11] argues that combining a classical planner with a reactive system seems to combine the worst features of both. Nevertheless, many systems have been proposed that combine deliberation and reaction, particularly for mobile robots. Examples of these systems include Gat's [26] and Simmons' [66]. In these systems a deliberative reasoning component is combined with a reactive control component. They differ in how the coordination between the components is managed. Pryor and Collins [56] argue that opportunistic on-demand reasoning is the key to combining deliberation and reaction.

Georgeff and Lansky's *procedural reasoning system* (PRS) [27] controls on-line computation using hand-coded procedures and on-line information. PRS is similar to Nilsson's [55] teleo-reactive programs in that it attempts to avoid over commitment by reducing the amount of advanced planning. A related approach is the work of Lyons and Hendriks [51]. Their approach is to construct reactive plans incrementally on-line in response to unanticipated situations. Hammond's [41] *case-based planning* and *opportunistic memory* perform similarly. In that work a planner reasons on-line about combining plans in response to execution-time opportunities. It then caches the results for future use (both in planning and

execution).

As opposed to the theoretical developments on resource-bounded problem solving, research on designing systems that combine deliberation and reaction is generally concerned with developing robust systems for specific environments, rather than modeling the behavior of the system in general. The framework and techniques of this dissertation might be applied toward better understanding the behavior of such systems.

CIRCA This dissertation shares many philosophical points with the work of Musliner on CIRCA (Cooperative Intelligent Real-Time Control Architecture) [52]. In particular, the viewpoint that control must be strictly guaranteed during deliberation is central in both works. While the techniques of this thesis use models of the environment in order to understand decision-making behavior and the corresponding predictable temporal patterns of situations, work on CIRCA makes assumptions about the ability to guarantee control over uncertainty.

CIRCA takes an extreme view of the interaction of deliberation and reaction. Namely, that deliberation time can not generally be bounded. Therefore, reaction must be compiled into control loops that guarantee that the deliberation system can process for an indeterminate amount of time without causing overall system failure. The deliberative process must generate control loops such that, once executed, the control loop may continue to execute indefinitely until the next control loop is generated.

Making such guarantees is equivalent to determining all possible situations that a system can reach, indefinitely, under the currently executing control loop. In order to be practically applied a control loop must effectively restrict the environment to a known subset of possible situations and guarantees that the system stay within that subset. Thus, this technique may only apply to environments for which exogenous uncertainty can be controlled at all times. This is similar to assumptions in supervisory control of discrete event systems (DES) [58] in which failure states are guaranteed unreachable.

Another commonality between this dissertation and CIRCA is that both are intended to handle dependent sequences of actions. CIRCA does so by making

assumptions about the ability to control uncertainty, while the analytical techniques of this dissertation assume that predictable temporal patterns of situations may be captured at design time.

Given its focus on hard real-time control, it is not surprising that the control loops of CIRCA may be viewed as static worst-case cyclic schedules. Under this view, the deliberation process of CIRCA may be viewed as an on-line version of the paging circuit of the schedule server of Chapter 5. CIRCA uses implicit assumptions about uncertainty to allow on-line deliberation of new schedules while this thesis relies upon explicit modeling of the environment in order to provide design-time generation of conditional schedules. Our approach allows for an understanding of global behavior at design time, while CIRCA cannot guarantee that any progress will be made across deliberation intervals.

Navigating large state spaces A complementary approach to the interaction of deliberation and reaction is that of Dean, Kaelbling, Kirman and Nicholson [17] This work makes use of anytime algorithms and deliberation scheduling to incrementally generate reactive control policies over subsets of a very large state space, over time.

In order achieve satisfactory reactive control without generating optimal control policies over the entire state space, this work assumes specific locality properties in the interaction of planning and execution. Namely, they assume the existence of an *envelope* of states surrounding the current state such that the system has low probability of leaving this envelope before a new reactive control policy is generated. Additionally, they assume a set of default reactions for states outside the envelope.

This is similar in general to the target tube and feedback policies described above [5, 6]. It differs in that the envelope is determined by the currently executing reactive control policy while the target tube is constructed to account for properties of the exogenous periodic system. As such, the target tube approach is more suitable to design-time behavior guarantees.

Compared to CIRCA, the envelope approach cannot make any guarantees

about the behavior of the system over time. At design time, they can only guarantee the behavior inherent in the default reactions. However, under favorable conditions, they will be able to make progress over time by computing new reactive control policies on-line.

8.4 Real-Time Scheduling

Throughout this thesis, a real-time scheduling application is used to motivate the techniques and demonstrate the use of the framework. Thus, a strong connection exists between the work of this dissertation and that of real-time scheduling. This is further exemplified by the conditional real-time avionics scheduling approach developed in Chapter 5. In this section we consider this connection in the context of recent work from the real-time scheduling literature.

The bulk of literature in real-time scheduling considers the construction of static schedules designed to handle periodic processes, or static schedules extended to accommodating dynamic tasks and aperiodic processes. Our approach to conditional real-time scheduling is explicitly designed to accomodate dynamic information about the scheduling problem. Thus, we focus our exploration on approaches that address the limitations of static scheduling. In these approaches dynamic information is used as a fundamental component of altering schedules over time.

Mode changes In work on *mode changes* [64, 23], static schedules are swapped at execution-time in response to dynamic information. A mode is analagous to a set of situations for which a static schedule may be designed. A mode change corresponds to a transition from one set of situations to another. Responding to a mode change event consists of switching from one static schedule to another. An important consideration in this work is the safe operation of the system during the transition period. A static schedule designed for a particular mode is equivalent to a situation-specific schedule developed in our conditional scheduling approach. Additionally, as in our conditional scheduling approach, design-time analysis is used to guarantee execution-time determinism.

The primary difference between our approach to conditional real-time scheduling and the work on mode changes is one of problem formulation and the subsequent impact on analysis techniques. In the real-time scheduling literature, mode changes are specified as part of the design problem. The modes and mode change events are represented using a graphical specification language that allows for representation of modes and relationships between modes. Solution techniques then involve the manipulation of this representation. The important technical issues center on the transitions between modes.

An additional contribution of our analytical techniques is a determination of both the requirements of an execution environment with respect to different modes, and the ability to provide a "mode-switching" solution given the constraints of the execution hardware. The basis for this determination are the situation-specific models of sampling frequency and the transition models of the uncertain flight environment. Thus, we are focusing on a different stage of analysis and design. And this change of focus impacts the techniques provided.

Therefore, although conceptually similar, the techniques of the mode-change literature and those of our approach to conditional real-time scheduling are complementary. A promising line of research would be to merge the techniques. Such a combination might achieve environment- and resource-specific mode changes, as well as guaranteed transition properties between pairs of modes.

Parametric dispatching A second real-time scheduling approach that relates to that of this thesis is work on *parametric dispatching* [28]. In parametric dispatching, tasks are statically ordered at design time but, the actual start and execution times of each task are determined dynamically.

Parametric dispatching consists of a design time component to determine the static ordering, an on-line scheduler to adjust start and execution times for each task, and a fast dispatcher to implement the schedule, taking into account any non real-time tasks. Given its reliance on on-line scheduling to handle dynamic events (as opposed to a predictive model of dynamics), parametric dispatching may be viewed as an interaction of fast reaction (dispatch tables) and slow deliberation (scheduling), the latter of which is divided between on-line and design-time

reasoning.

Thus, the parametric dispatching solution may be understood as an on-line decision-making solution. Given a predictive model of execution dynamics, the techniques of this thesis may then be applied toward explicitly understanding the computational choices of parametric dispatching.

8.5 Paging and Prefetching

Given the common terminology and borrowed abstractions, it is natural to compare this work to the literature on paging and prefetching. In this section we briefly discuss this relationship.

In work on prefetching, pages of memory are retrieved from secondary storage to primary storage in anticipation of page requests by user processes. The success of prefetching is measured in terms of "hits" and "faults." A *page fault* occurs when a user memory request can not be satisfied by any page in primary storage and, thus, causes the on-demand paging from secondary storage to satisfy the request. Given that this on-demand paging is very expensive, a system goal is to minimize page faults.

In our time-critical control systems, we explicitly require zero page faults. A page fault in this context is analogous to an unexpected delay in on-line deliberation. Such a delay potentially invalidates the decision-making behavior of the system: unless delay has been factored into the behavior directly. Each potential page fault may have a different impact on global behavior depending upon when the fault occurs. This cannot be captured in summary measures such as "hit ratio." Page faults may be tolerated in more forgiving environments, in which the effects of actions have only local impact and do not jeopardize global behavior. In the prefetching literature, zero page faults is referred to as *perfect prefetching*.

Much work on prefetching assumes *pure prefetching*. In pure prefetching, the cost of prefetching is constant, such that some fixed number of pages can always be retrieved between any two pages requests. This is equivalent to our fixed

retrieval interval design choice in arbitrary-circuit paging. However, arbitrary-circuit paging may only be considered pure with respect to a specific decision-making behavior. The ability to provide pure prefetching is one measure of the resource requirements of a decision-making behavior.

A related component of paging systems is the *page replacement* algorithm. A page replacement algorithm chooses which pages to replace whenever new pages are retrieved from secondary storage. Page replacement becomes necessary when the pure prefetching of the entire primary storage cannot be performed between page requests. In arbitrary-circuit paging, page replacement is not directly applicable. The new decision-making circuit entirely replaces the previous in every interval. However, in extended time-critical control systems that allow persistence (as discussed in Chapter 9), the page replacement analogy becomes relevant.

8.6 Personal Prior Research Results

This thesis is the result of many years of varied research activities at Brown University and elsewhere. In addition to providing its relationship to the wider literature, I conclude this chapter by presenting this work in an abbreviated personal historical context.

On-line decision making This dissertation primarily extends previous work, with Tom Dean, on understanding how slow, high-level systems might interact with faster, more reactive systems [37, 35, 31]. The prior work focused on exploiting restricted temporal patterns that capture predictable periodic fluctuations in the reachable states of a dynamic uncertain environment. The temporal patterns provide a basis for anticipating on-line computational demands at design time. We then employed deliberation scheduling techniques to allocate limited computational resources. This method results in open-loop strategies for allocating on-line resources to anytime algorithms and contract algorithms such that large variabilities in computational demand are smoothed across time.

In this work we also developed a specific table-driven control architecture targeted toward the application areas of traffic control of multiple robot delivery

vehicles and scheduling of runways and gates in local air traffic control. The open-loop strategies achieve time-critical responsiveness only in environments that posses periodic structure in reachable states over time that is somewhat invariant to the actions of the decision-making system itself. This research took a more heuristic approach to designing time-critical control systems than that of this dissertation. In this thesis, I develop a more comprehensive and precise framework, and am better able to provide rigorous analyses of resulting behavior and limiting conditions.

Time-critical scheduling In work on time-critical scheduling under uncertainty, with Tom Dean, we developed techniques for efficiently handling stochastic processes with both very large state and very large action spaces [34, 33, 36]. These techniques complement and extend work at Brown on time-critical planning under uncertainty [20] by focusing on the additional combinatorial complexity due to the large action spaces of scheduling problems. These issues are encountered in problems such as transportation scheduling for large-scale strategic military airlift, air-traffic control, and factory machine scheduling.

The planning work took advantage of locality properties of a stochastic process to iteratively decompose problems into bounded state-space regions that could be solved by exhaustive policy generation search methods. These locality properties are not necessarily maintained in scheduling domains. We re-evaluated both the regional decomposition techniques and the ability to apply exhaustive policy generation methods. Our solution was to apply on-line stochastic (Monte Carlo) simulation to selectively explore the state/action space and apply bottleneck-centered scheduling heuristics to iteratively constrain actions and improve initial control policies. As in the planning work, deliberation scheduling is employed at design time to allocate on-line computation time between the anytime algorithms that make up the decomposition and policy generation phases and across iterations of both phases.

Transportation problems My initial work on distributed transportation scheduling, with Tom Dean, involved exploring the problems of the United States

Transportation Command [16]. We developed a framework for precisely formalizing the static and dynamic sources of complexity in transportation problems, and described research issues in the context of previous work in both computer science and operations research. The merit of the resulting framework was demonstrated by the development of restricted transportation problems that isolated various sources of complexity. This dissertation can be viewed as addressing issues of control over dynamical systems formalized in the transportation work, though using a somewhat different modeling perspective.

We derived a restriction on the general formulation that we called the capacity constrained package transfer problem, in which vehicles with fixed schedules and limited capacity transport packages throughout a network of locations [38, 32]. Having shown this problem NP-complete, our solution approach was to apply randomized rounding techniques previously applied to multicommodity flow problems. Adapting these techniques, we provided solutions with provable constraint violation bounds and expected performance guarantees. We additionally provided a careful analysis of the limitations of these techniques. The analytical techniques and problem decomposition framework developed in this work have extensions to more general pickup/delivery problems, including those of information delivery. In separate work, I explored the application of a distributed greedy packet routing algorithm toward solving distributed transportation scheduling problems under restricted network topologies.

Parallel processing, pattern matching and machine vision In both my work at AT&T Bell Laboratories and early work at Brown University, I explored the use of parallelism for solving computationally intensive pattern matching tasks. While both sets of work looked at the logical partitioning of a problem so that pieces may be solved in parallel, they took two different approaches. At AT&T I looked at program-level parallelism to aid in the detection, characterization and classification of transient events in signal processing data [39]. The novel approach undertaken was to re-interpret broadband sonar signals as two-dimensional visual images. A side-effect of this approach was an increase in the computational burden of an already data-intensive task. Dr. A. Hanyong Yuhan

and I employed low-level pattern matching techniques from the machine vision literature combined with higher-level component clustering techniques to detect, characterize and classify transient events in noisy data. This work successfully demonstrated to Navy customers the use of the AT&T BT-100 parallel binary tree machine to speed-up computationally intensive signal interpretation tasks.

As opposed to the program-level parallelism of my work at AT&T, I have explored parallelism at the bit level in developing an innovative highly parallel programmable logic array with a team of students at Brown University, under the guidance of Professor Dan Lopresti [2]. The 2μ CMOS prototype chip holds 198 basic cells, each possessing two flip-flops and two independent ALU's capable of performing arbitrary boolean functions of up to three inputs. A unique alternating grid interconnection scheme formed a compromise between flexibility and chip area, and was designed specifically for pattern matching tasks. The chip was fabricated by MOSIS, and the work was awarded an Honorable Mention notice in the 1991 National VLSI Integrated Circuit Design Contest.

Chapter 9

Limitations and Future Work

The analytical techniques of this thesis are primarily framed with respect to a single form of time-critical control system. Namely, the two-level arbitrary-circuit paging architecture. Under this architecture, we are able to provide a rich set of techniques for solution analysis and design of a subset of on-line decision-making problems. Additionally, this architecture serves as the basis of a new approach to real-time avionics solution design.

However, the framework of this thesis is intended to apply to a wider range of on-line decision-making problems. In this chapter we discuss some of the limitations of the arbitrary-circuit paging architecture and propose additional time-critical control system building blocks to be used in the analysis and design of more general on-line decision-making solutions. One of the strengths of the framework of this thesis is that extensions to existing techniques are straightforward to conceptualize.

In Section 9.1 we discuss a fundamental change in perspective. Namely, allowing solutions in which complete coverage of all possible reachable situations can not always be guaranteed. In Section 9.2 we present a way to understand the limitations of time-critical control systems based on understanding the long- and short-term properties of the associated decision-making behaviors. In Sections 9.3 and 9.4 we describe extensions that address the form and timing of time-critical control system architectures, and limiting properties of circuits, respectively.

9.1 Relaxing the Complete Coverage Requirement

One extension that warrants separate treatment is relaxing our requirement that decision-making circuits always cover all reachable situations. For example, we might try to understand the implications of paging decision-making circuits that only partially cover a possible set of reachable situations. To do so, we must define the behavior of the decision-making solution when uncovered situations are experienced. Associating costs with "faults" would be one way to locally model this behavior. Under certain independence assumptions, a minimization of these costs might provide an effective solution. Without such assumptions, the global implications of faults must be evaluated with respect to the impact on achievable behavior.

We could directly apply our techniques to determine whether or not a given optimal policy described in terms of a set of partial policies implemented as circuits could be scheduled on a particular architecture. A more difficult problem, one which requires an extension of the techniques presented in this thesis, is to determine if there exists a decomposition of a given optimal policy into partial policies that could be scheduled on a particular architecture. An even more complicated extension would involve simultaneously choosing a policy and a decomposition of the policy so as to maximize some measure of performance.

While these extensions suggest interesting applications and raise exciting issues, we felt that they actually distracted from the issues specifically concerned with resource-bounded decision making. For this reason, we opted to work on the simpler version of the problem thereby avoiding orthogonal combinatorial issues.

9.2 Understanding the Limitations of Specific Time-Critical Control Systems

The arbitrary-circuit paging architecture introduced in Chapter 3 captures a specific form of tradeoff in the analysis and design of on-line decision-making solutions. Lemma 4.1 makes explicit the class of decision-making behaviors achievable under any executable solution to an arbitrary-circuit paging problem. The class of decision-making behaviors specified by Lemma 4.1 includes only those for which we can construct an executable paging strategy circuit of domain F^n mapping to a subcollection of executable decision-making circuits that covers the sets of situations R_0^n and R_s^n, for each $s \in F^n$. A desired behavior E^* must be a member of this class of decision-making behaviors in order to be implemented as a solution to an arbitrary-circuit paging problem.

We may use the results of Lemmas 4.1 and 4.4 to gain insight into the decision-making behaviors that *cannot* be implemented with any arbitrary-circuit paging solution. Through this process, we can evaluate the design-time choices of the arbitrary-circuit paging problem formulation. More specifically, we want to find direct correspondence between paging architecture and limiting achievable behavior. These choices can then be modified in order to achieve different classes of decision-making behavior.

These choices take the form of features of the paging architecture and assumptions about the form of paging strategies and decision-making circuits. In subsequent sections, we suggest extensions to overcome some of these limitations on achievable decision-making behavior.

Short-term and long-term properties of decision-making behaviors We can relate properties of decision-making behaviors to those of *Markov chains*. Briefly, a discrete-time Markov chain describes the stochastic behavior of a system that makes discrete-time steps through a sequence of *states*, drawn from a countable set. Informally, in the theory of Markov chains (*e.g.* see [46, 57]), a chain is considered to be *ergodic* if, in the limit, the probability of being in any state of the chain converges to a constant, independent of time tick or start state.

A contrasting concept is that of *periodicity*, in which the probability of being in a periodic state toggles, in the limit, between zero and a constant, depending upon the relationship between time tick, period and start state. All states of an ergodic chain are aperiodic.

Asymptotic ergodicity and periodicity are examples of *long-term* properties of decision-making behaviors, when modeled by Markov chains. A decision-making behavior that can be modeled by an ergodic Markov chain with a very large state space would be difficult to implement under any paging strategy that is fixed at design time. More specifically, the domain of such a strategy must cover all situations with non-zero probability in the limit. Since the strategy is fixed at design time, it must handle behavior in the limit.

On the other hand, asymptotic ergodicity presents less of a problem for the domains of conditionally paged decision-making circuits. The use of on-line observation allows us to avoid asymptotic measures when paging decision-making circuits. Instead, we can focus on the *short-term* properties of a decision-making behavior. For example, the rate of growth in uncertainty between the selection of a decision-making circuit and its subsequent execution.

A time-critical control system modeling an on-line decision-making solution can be evaluated based on how well the tradeoffs of the control system match the requirements of short-term and long-term properties of desired decision-making behavior. Lemma 4.4 extends Lemma 4.1 to provide a precise quantitative formulation of the relationship between long-term decision-making behavior, short-term decision-making behavior, and the decision-making circuits that implement such behavior, in any executable solution to an arbitrary-circuit paging problem. In particular, the set F^n represents the long-term structure that is deterministically known at design time, while the sets R_s^n represent the short-term structure that is conditioned on specific execution instances. The equations of Lemma 4.4 capture the relationships between long-term structure and any subcollection of decision-making circuits that provide a covering to satisfy Lemma 4.1.

9.3 Developing New Time-Critical Control System Architectures

In this section we present limiting features of the arbitrary-circuit paging architecture and discuss architectural extensions to overcome these limitations. The limitations are described in terms of long- and short-term properties introduced in Section 9.2.

9.3.1 Fixed paging strategy and long-term decision-making behavior

The arbitrary-circuit paging architecture includes a fixed paging strategy, completely specified at design time. This paging strategy must cover all situations that may be observed at the beginning of retrieval intervals, namely F^n. Any decision-making behavior for which F^n cannot be bounded below the entire situation space is problematic, especially in the face of a large situation space $\mathcal{S}$. This includes ergodic behavior and periodic behaviors for which the retrieval interval n does not coincide with the underlying period.

On the other hand, a fixed paging strategy may be satisfactory despite unbounded F^n, if the strategy circuit is of low-complexity. The complexity of the strategy circuit is represented by the functions T_γ and B_γ and constrained by the equations of Lemma 4.4, as described in Section 4.5. On-line resources consumed by a fixed strategy impact resources available for decision-making circuits.

Two alternatives for designing decision-making solutions to implement decision-making behaviors for which a fixed paging strategy is not effective are (1) removing the on-line strategy completely, and (2) generating a strategy that is not fixed at design time. These architectural alternatives are explored below.

Alternatively, a fixed paging strategy may be ineffective, not because the strategy is fixed at design time but, because the fixed retrieval interval n does not capture the long-term structure of the desired behavior. Extensions addressing this problem are presented in Section 9.3.2.

Open-loop paging of decision-making circuits To overcome very large F^n, we might consider architectures for which the paging of decision-making circuits is completely determined at design time. We refer to this as *open-loop* paging; and refer to the conditional paging strategy of the arbitrary-circuit paging architecture as *closed-loop*.

A closed-loop paging strategy allows us to provide execution-specific conditional sequencing of decision-making circuits. The ability to provide conditional sequencing comes with the additional cost of generating, storing and executing the paging strategy circuit itself.

While a closed-loop strategy may be considered a conditional plan or policy for sequencing decision-making circuits, an open-loop strategy is a fixed plan for sequencing decision-making circuits. Note that an on-line decision-making solution specifying an open-loop sequencing of decision-making circuits still maintains conditional interaction with the environment through the decision-making circuits themselves.

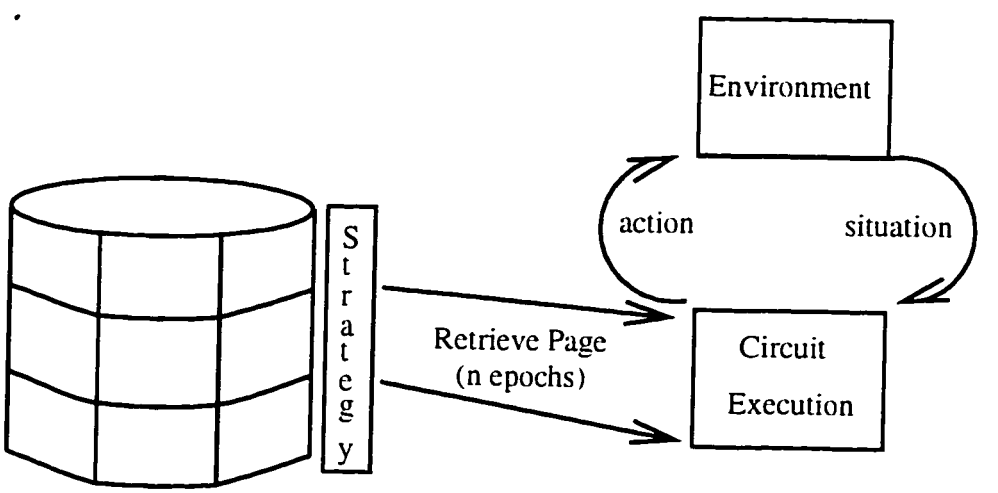

Figure 9.1: Open-loop paging architecture

An open-loop paging architecture is depicted Figure 9.1. This architecture differs from that of Figure 3.4 in that the strategy has been moved to the secondary storage device itself. By making the strategy part of the storage device, we are illustrating that the strategy and sequencing cannot be differentiated, *i.e.*

retrieval decisions are not conditioned on situations of the environment. Furthermore, the absence of an on-line strategy implies that capturing resource tradeoffs by bounding the size of secondary storage is not necessary. The number of pages stored on the storage device does not impact on-line resource limitations. This allows arbitrary fixed sequencings to be composed, regardless of length. Decision-making circuits are retrieved from disk to cache every n ticks according to the design-time sequencing.

Reliance on design-time reasoning in generating decision-making solutions requires an ability to model asymptotic properties of decision-making behavior. For example, decision-making behavior that may be modeled by a periodic Markov chain may be amenable to open-loop paging, provided that each period can be captured by a decision-making circuit that may be executed within the resources of the execution hardware.

Open-loop paging represents a tradeoff. It allows us to ignore the patterns of situations that make up the domain of a paging circuit, F^n, while at the same time removing the benefits of having an on-line paging circuit. Namely, the ability to consider the set of possible situations based on a recently observed situation. Since each π_i is determined at design time, each PS^i must be determined based solely on design-time information. In particular, for any retrieval interval, the set of possible situations under an open-loop paging architecture is equivalent to the union of the reachable sets of possible situations allowed in the arbitrary-circuit paging architecture. However, the absence of a paging circuit potentially provides additional on-line resources for decision-making circuits.

Paged strategy Requiring that the entire strategy be loaded into on-line storage prior to decision-making limits the class of decision-making behaviors that are executable. It is useful to consider approaches that do not require the entire strategy to be loaded prior to decision-making. For example, we might consider the paging of the strategy circuit itself.

One implication of paging the strategy circuit itself is that short-term patterns impact both strategy and decision-making circuits. Observe that, at any time tick nj, the set of situations that must be covered by the paging strategy

must also be covered by the currently executing decision-making circuit. This shared domain makes combined paging of strategy and decision-making circuits a reasonable architectural extension to consider. Figure 9.2 depicts a time-critical control system in which decision-making circuits and strategy circuits are paged simultaneously.

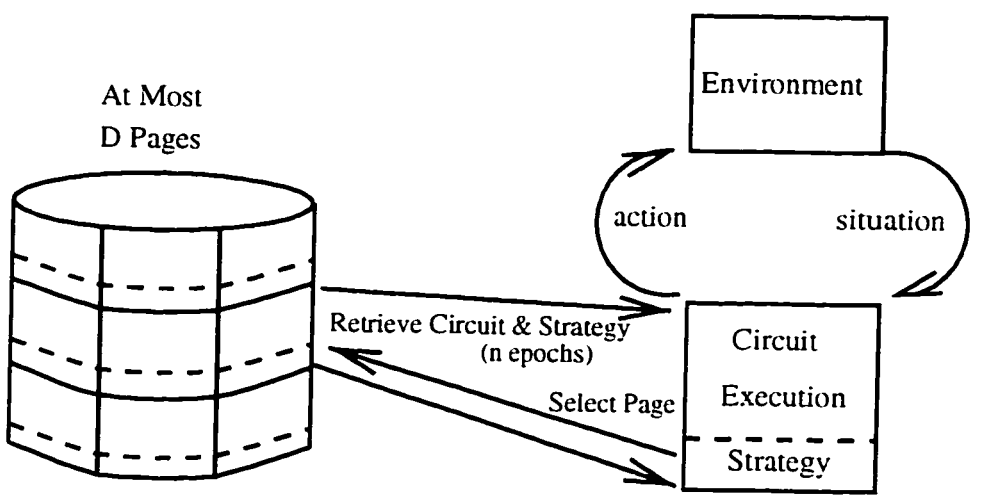

Figure 9.2: Architecture for simultaneously paging both decision-making circuits and strategy circuits

Under a paged strategy architecture, the domain of each strategy page is a function of P^{nj} rather than F^n. We might consider either open- or closed-loop paging of strategies.

9.3.2 Fixed retrieval intervals and short-term decision-making behavior

An arbitrary-circuit paging problem specifies a fixed retrieval interval n during which the current decision-making circuit remains constant and the subsequent circuit is retrieved. The impact of this time-critical control system design choice is briefly mentioned above with respect to long-term decision-making behavior. Furthermore, specifying a constant duration retrieval interval limits the flexibility of arbitrary-circuit paging solutions with respect to capturing variable of patterns of short-term behavior.

As mentioned, the sets R_0^n, and R_s^n, for each $s \in F^n$, capture all variations of short-term behavior that may occur in any execution of an arbitrary-circuit paging decision-making solution. The size of each these sets is directly related to the duration of retrieval and the rate of dynamics and growth of uncertainty. The number of such sets is directly related to the long-term structure described above. These figures impact the resources needed to retrieve and execute suitable decision-making circuits. Reducing the fixed size of n may beneficially impact short-term behavior while detrimentally impacting long-term behavior.

Variable and Indeterminate Retrieval Intervals We might consider two-level paging architectures for which the retrieval intervals vary in length across time, or are indeterminate at design time (*e.g.* determined based on on-line situations). Such extensions allow the exploration of decision-making behaviors that do not have rigid time-based structure.

One way to approach such an extension would be to view the retrieval process as a *semi-Markov* process. Under this view, we have a distribution over retrieval durations rather than a fixed retrieval duration.

Fixed retrieval intervals force time/space tradeoffs that are determined by the largest decision-making circuit. Variable retrieval intervals remove this bottleneck by allowing additional time to retrieve larger decision-making circuits. However, this extension introduces additional complexities in the design-time analysis and design of a paging strategies.

As exemplified in the intelligent control architecture of [52], we note that the ability to provide indeterminate retrieval intervals requires that decision-making circuits provide strict control over the environment during these intervals. In other words, we must guarantee at design time that any decision-making circuit executing during an indeterminate retrieval interval cover all situations that may occur during that interval. Thus, a bounded-resource decision-making circuit must have strict control over the environment in order to make this guarantee.

9.4 Extending the Properties of Strategy and Decision-Making Circuits

In developing analytical techniques for solving on-line decision-making problems with the arbitrary-circuit paging architecture, we explicitly assume Markov stationary decision-making circuits and strategy circuits. We might consider the implications of removing these assumptions.

Markov circuit assumptions The arbitrary-circuit paging problem explicitly restricts decision-making circuits and paging strategies to be Markovian. One implication of this restriction is that a decision-making circuit cannot rely on the results of any other on-line computations in responding to situations. Decision-making circuits must be designed to produce an acceptable response within a single tick's time based solely on the current situation and any information that could be embedded into the circuit at design time. Furthermore, Markov paging strategies can not overcome this limitation by sequencing decision-making circuits conditionally, depending upon sequences of on-line situations.

Since we would like to guarantee that decision-making circuits produce actions that are consistent with long-term behavior, they must be embedded with enough information at design time to reproduce long-term behavior based solely on the current situation. This calls into question the ability to produce decision-making circuits to implement high-complexity decision procedures that execute in the limited space and time specified by Lemma 4.4. In effect, the long-term effects on decision-making behavior of local decision-making actions must be restricted to correspond to the on-line resource limitations.

We might overcome these limitations by considering less restrictive assumptions on decision-making circuits and/or strategy circuits. In particular, we can loosen the Markov requirement of these circuits. Two alternative time-critical control system design choices that enable this generalization are (1) adding the ability to specify dependence between decision-making circuits so that a decision-making circuit may rely on the changes to the environment caused by a prior circuit, and (2) incorporating persistent storage into the on-line architecture to allow

the results of redundant and incremental computations to be carried forward in time. These alternatives are explored below.

Note that generalizations to non-Markov decision-making and/or strategy circuits requires the development of more complex design-time reasoning about the on-line sequencing of decision-making circuits. Rather than solving covering problems based on a single tick or group of consecutive ticks, we must now consider the ordering of decision-making circuits across time, including ordering decisions that are not determined until execution time. Thus, the decoupling result of Lemma 3.1 must be modified.

Dependent decision-making circuits An initial extension that incorporates dependence across decision-making circuits is to consider time-critical control systems in which each decision-making circuit specifies its domain in terms of pairs of consecutive situations, rather than independent situations. We might model this limited history dependence by decision processes that are *2-Markov*.

Specifying 2-Markov decision-making circuits is related to providing precedence constraints between decision-making circuits. To understand the behavior of time-critical control systems composed of 2-Markov decision-making circuits we might then require design-time analytical techniques that handled precedence constraints.

While the independence of decision-making circuits allows for paging solutions and design-time techniques based on guaranteeing coverage, the additional complexity introduced by dependence requires design-time techniques more similar to scheduling or planning. Note that an open-loop paging architecture can directly incorporate the static scheduling/planning of dependent decision-making circuits.

Explicit persistence As opposed to simple forms of dependence across circuits, persistence must be viewed as a general form of history-dependence. In other words, by allowing data or circuits to persist in on-line storage, we are connecting decision-making at arbitrary distances across time. Thus, explicit persistence is a powerful extension to the arbitrary-circuit paging architecture.

We might incorporate restricted forms of persistence such as maintaining data from decision-making circuits only until the next time tick, such as the pattern of persistence depicted in Figure 9.3. Note that the form of persistence depicted in Figure 9.3 is similar to that assumed by "envelope" approaches such as those of [17, 52]. Another form of persistence would be to allow the circuits themselves to persist, to be combined over time.

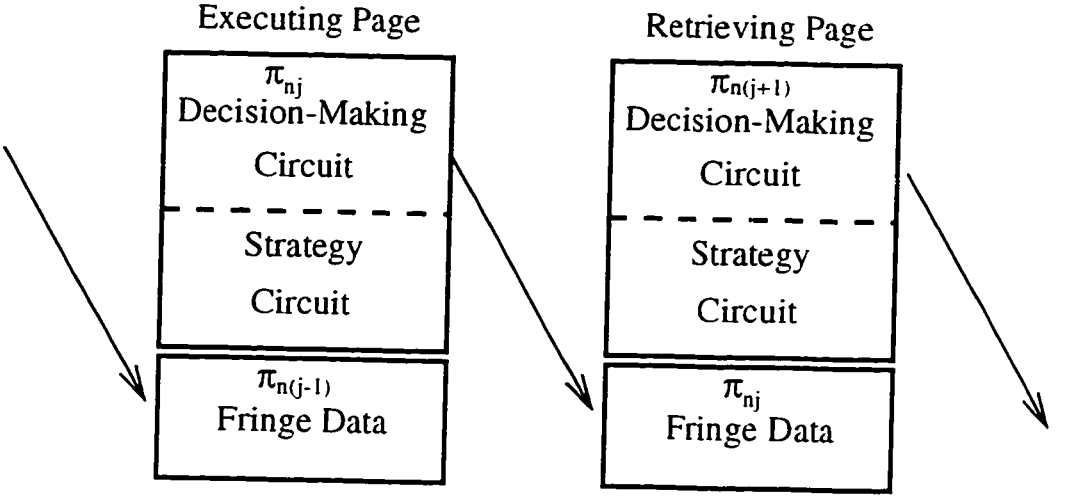

Figure 9.3: Bipartite cache, each half split to include decision-making circuit, strategy circuit and previously generated fringe data

A general approach to persistence might include modeling the change in content of on-line storage over time. Such a model could be incorporated into the design-time analysis of resource tradeoffs, taking into account space allocation both during processing and between processing.

9.5 Conclusion

This dissertation addresses an ill-formed class of problems characterized by the interaction of a bounded-resource decision-making system with an uncertain, dynamic environment. We have developed mathematical formulations for understanding these problems, evaluated the complexity of providing solutions and developed mappings to this class of problems from traditional computer science

problems. These mappings have produced analytical techniques for evaluating the behavior of on-line decision-making solutions, as well as design techniques for developing new solutions. In the process we have developed a new approach to real-time avionics system design. The practical application of the framework and techniques of this thesis have been validated by empirical study. Finally, we have identified the limitations of the techniques and proposed extensions to overcome these limitations.

To summarize, our contributions have been to:

1. Develop a framework for expressing and evaluating tradeoffs, at design time, in designing systems that interact with dynamic, uncertain worlds. Including:

   - Formulating an on-line decision-making solution as a time-critical control system with specific time/space tradeoffs: explicitly modeling the relationships between all time and space resources;
   - Characterizing the predictable temporal patterns in the interaction of a dynamic, uncertain world and a given time-critical control system: explicitly capturing the interdependency of decision-making responses across time;
   - Formulating a decision problem to capture the necessary and sufficient conditions under which a system can be shown, at design time, to solve the corresponding on-line decision-making problem; and
   - Providing analytical techniques for evaluating and solving a particular class of decision problems: including complexity analyses, approximation algorithms, and combinatorial optimization enumeration schemes;

2. Demonstrate the framework by developing a new approach to designing and analyzing real-time avionics systems;

3. Empirically evaluate the framework and analytical techniques with respect to the real-time avionics scheduling approach; and

4. Explore limitations and extensions of the framework.

Appendix A

Proofs and Omitted Details

In this appendix, we provide proofs for the lemmas, theorems and corollaries of previous chapters.

A.1 Decoupling Decision-Making Behavior Formulation and Time-Critical Control System Design

Lemma A.1 (3.1) *Let desired behavior E^* be interchangeable with respect to the collection of decision-making circuits C. For each time tick i, let $\{PS^i\}_{i\geq 0}$ be the possible situations attainable with respect to a given decision-making solution and execution.*

If desired behavior E^* is achievable under a given decision-making environment and a time-critical control system is guaranteed to only produce sequencings $(\pi_0, \pi_1, \pi_2, \ldots)$ such that each decision-making circuit $\pi_i = c_S \in C$ covers the set of possible situations PS^i, for each tick i, then any decision-making solution based on this control system implements desired behavior E^*.

Proof:

Proof by induction on prefixes of execution trajectories.

Let PS^i be the set of possible situations at time tick i. Let E^* be an achievable decision-making behavior interchangeable with respect to C. Let $(\pi_0, \pi_1, \pi_2, \ldots)$ be an arbitrary sequencing drawn from C produced by a given decision-making solution.

Base case. Let s_0 be any situation observed at time tick 0. By definition, PS^0 does not exclude s_0, *i.e.* $s_0 \in PS^0$. Furthermore, any achievable decision-making behavior must include some possible trajectory for each s_0 that may be observed at time tick 0, *i.e.* $(s_0, .) \in \mathrm{e}$ for some $\mathrm{e} \in \mathrm{E}^*$.

By antecedent, E^* is achievable and π_0 covers PS^0. Thus, since E^* is interchangeable with respect to C and $\pi_0 \in C$, then $\mathrm{e}_0'^- = (s_0, \pi_0(s_0))$ is a prefix of some trajectory $\mathrm{e}' \in \mathrm{E}^*$.

Induction. Assume $\mathrm{e}_i^- = ((s_0, \pi_0(s_0)), \ldots, (s_i, \pi_i(s_i)))$ is a prefix of some trajectory $\mathrm{e} \in \mathrm{E}^*$. Let s_{i+1} be the situation observed in tick $i+1$. By definition, $s_{i+1} \in PS^{i+1}$. Since e_i^- is the prefix of an achievable desired behavior, $(\mathrm{e}_i^- : (s_{i+1}, .))$ is prefix of some $\mathrm{e}' \in \mathrm{E}^*$.

By antecedent, E^* is achievable and π_{i+1} covers PS^{i+1}. Thus, since E^* is interchangeable with respect to C and $\pi_{i+1} \in C$, then $\mathrm{e}_{i+1}''^- = (\mathrm{e}_i^- : (s_{i+1}, \pi_{i+1}(s_{i+1})))$, is a prefix of some trajectory $\mathrm{e}'' \in \mathrm{E}^*$.

□/

A.2 The Existence of Arbitrary-Circuit Paging Solutions

Lemma A.2 (4.1) *Given a stationary Markov transition model P_s^k and initial set of situations $P^0 \in 2^S$, a decision-making solution exists that implements achievable desired behavior E^* for an* arbitrary-circuit paging problem *if and only if there exists a subset $C' \subseteq C$ of size $|C'| \leq D$ such that, for each $s \in F^n$, $R_s^n \subseteq V_S$ for some circuit $c_S \in C'$, $R_0^n \subseteq V_S$ for some circuit $c_S \in C'$, and there exists a fixed strategy γ with domain F^n and range C' that implements the mapping of situations to decision-making circuits and is executable under the given time and space limitations.*

Proof:

$\Leftarrow$

Let $(\pi_0, \pi_1, \pi_2, \ldots)$ be an arbitrary sequencing drawn from C that may be produced by the decision-making solution. Furthermore, the decision-making solution implements achievable desired behavior E^*. Show that the arbitrary sequencing must satisfy the given conditions. We show each condition by contradiction.

Let $(\pi_0, \pi_1, \pi_2, \ldots)$ include circuits drawn from $C' \subseteq C$ of size $|C'|$.

Assume $|C'| > D$. By definition of the arbitrary-circuit paging problem, $(\pi_0, \pi_1, \pi_2, \ldots)$ must be drawn from a set of decision-making circuits of at most size D. So, $(\pi_0, \pi_1, \pi_2, \ldots)$ cannot be produced by the decision-making solution. Contradiction.

Assume there exists some $s \in F^n$, such that $R_s^n \not\subseteq V_S$ for any circuit $c_S \in C'$. For $j \geq 0$, let nj be a time tick at which s is possible, based on design-time information. Then there exists some execution for which $s_{nj} = s$. For $n(j+1) \leq i < n(j+2)$, let $\pi_i = c_S \in C'$ be sequenced during this execution. By assumption, $R_s^n \not\subseteq V_S$. Since $R_s^n = \cup_{\{n(j+1) \leq i < n(j+2)\}} PS^i$, there exists some PS^i, $n(j+1) \leq i < n(j+2)$, which cannot be guaranteed not to exclude some possible situation. Contradiction.

Assume $R_0^n \not\subseteq V_S$ for any circuit $c_S \in C'$. Same reasoning as above for special case of initial interval.

Assume there exists some $s \in F^n$ that is not in the domain of γ. Let nj be a time tick at which s is possible, based on design-time information. Then there exists some execution for which $s_{nj} = s$. Since s is not in the domain of γ, $\pi_i = \gamma(s, nj)$, $n(j+1) \leq i < n(j+2)$, are undefined. Thus, $(\pi_0, \pi_1, \pi_2, \ldots)$ cannot be produced by the decision-making solution. Contradiction.

Assume that there exists some $c_S \in C'$ that is not in the range of γ. Without loss of generality, assume that $\pi_i = c_S$ in some execution of the decision-making solution. Since, c_S is not in the range of γ, it cannot be selected for sequencing at time tick i. Thus, $(\pi_0, \pi_1, \pi_2, \ldots)$ cannot be produced by the decision-making solution. Contradiction.

Finally, assume that γ is not executable under the given time and space limitations. In other words, assume there is some execution for which either γ cannot fit in cache or requires more than n time to select and retrieve the page for some time tick i. Thus, $(\pi_0, \pi_1, \pi_2, \ldots)$ cannot be produced by the decision-making solution. Contradiction.

$\Rightarrow$

By Lemma 3.1 we need only show that any decision-making solution satisfying these conditions is guaranteed to only produce sequencings $(\pi_0, \pi_1, \pi_2, \ldots)$ such that each decision-making circuit $\pi_i = c_S \in C'$ covers the set of possible situations PS^i, for each tick i.

Since γ is an executable paging circuit defined over all F^n with range from C', we know that all sequencings are well-defined and each π_i is drawn from C'.

We need only show that the set R_0^n and the sequence $\{R_s^n | s \in P^{nj}\}_{j \geq 0}$ are sufficient to derive any sequence of possible sets $\{PS^i\}_{i \geq 0}$, such that no $s_i \notin PS^i$ can ever be observed at time tick i. In particular, $\cup_{\{0 \leq i < n\}} PS^i = R_0^n$, and for $j \geq 0$, $\cup_{\{n(j+1) \leq i < n(j+2)\}} PS^i = R_{s_{nj}}^n$ satisfy our requirements.

□/

A.3 Arbitrary-Circuit Paging Complexity

Theorem A.1 (4.1) *The decision problem of Lemma 4.1 is NP-complete.*

Proof:

A positive answer to the following decision problem is a necessary condition for proving that any instance of the arbitrary-circuit paging problem has a solution.

Definition A.1 *The* circuit cover *decision problem consists of the following components.*

INSTANCE: *Collection C of executable circuits with domain drawn from finite set S, collection R of subsets drawn from finite set S (such that $R = \{R_0^n\} \cup \{R_s^n | s \in F^n\}$), positive integer D.*

QUESTION: *Does C contain a cover for R of size D or less, i.e. a subset $C' \subseteq C$ with $|C'| \leq D$ such that for every element $r \in R$, $r \subset c$ for at least one $c \in C'$?*

Since we can always guess a set $C' \subseteq C, |C'| \leq D$ and test each element $c \in C'$ against each element $r \in R$ in time polynomial in $|C'|$ (D), $|R|$[1] and $|S|$, the circuit cover problem is in NP. As mentioned, *minimum set cover* [24, 14, 13] is known to be NP-complete.

We can construct an instance of circuit cover from any instance of minimum set cover as follows. For each $y \in Y$ and $x \in X$ construct a circuit c_y such that $x \in V_y$ iff $x \in y$. For each $x \in X$, construct an $r = \{x\}$. It can be seen that Y' is a cover for X of size less than or equal to K if and only if $C' = \{c_y | y \in Y'\}$ is a cover for $R = \{r\}$ of size less than or equal to $D = K$. Furthermore, this transformation can be performed in time polynomial in $|Y|$ and $|X|$.

The careful reader will note that, unless $n = 1$ or $|X| = 1$, the set R constructed cannot correspond to a stationary source, by the following reasoning. By definition, $P_s^n \subseteq R_s^n$ for all n, s. Without loss of generality we can assume that P_s^k is never empty. If $|R_s^n| = 1$ (as is true in the construction), then we must have $P_s^k = R_s^n, \forall n \leq k < 2n$ and $P_s^{2n} = R_s^{2n}$. Unless X is singular, there must be some $R_s^n \neq R_s^{2n}$. Therefore, P_s^k defines a nonstationary source.

However, it is straightforward to augment the construction to always admit a stationary source. In particular, augment the finite set S of the circuit cover problem to include $|X|(n-1)$ elements, not in X. Then augment the domain of each circuit to include these elements and augment each r with $(n-1)$ of these elements. We now can define a stationary source from the construction without changing the result. This augmented transformation can be performed in time polynomial in $|Y|$, $|X|$ and n.

□/

[1] $|R| \leq |F^n| + 1 \leq |S| + 1$.

A.4 Reduction of Coverage Requirements

Lemma A.3 (4.2) *A solution exists to an* arbitrary executable circuit collection, bipartite cache, fixed retrieval interval, circuit paging problem *with possible set collection R if and only if a solution exists with possible set collection $R' = \{r' \in R | \forall r \in R. r \neq r' \rightarrow r' \not\subset r\}$.*

Proof:

$\Rightarrow$

Let $C' \subseteq C$ be a set of circuits that covers R' and satisfies the conditions of Lemma 4.1. For any $r \in R$ there exists a $r' \in R'$ such that $r \subseteq r'$. Therefore, any solution that covers R' also covers R.

$\Leftarrow$

Since each $r' \in R'$ is also in R, any solution that covers R covers R' as well.

□/

In evaluating the worst-case running time for constructing R' from any R and constructing Y, we assume that containment between two sets can be determined in time linear in the number of elements of the containing set, as long as the elements in each set have been ordered. Roughly, for each element of the contained set we step through the containing set until that element is found. We then switch to the next element, continuing the linear search from where we left off. If we ever fail to match an element then containment fails.

A.5 Lower Bound Based On Greedy-Set-Cover

Corollary A.1 (4.1) *Let $C' \subseteq C$ be the circuit covering produced by Greedy-Set-Cover applied to X and Y in the above construction. If $|C'| > D\mathcal{H}(|R'|)$ then an executable circuit covering ($C^* \subseteq C$, $|C^*| \leq D$) does NOT exist.*

Proof:

Assume that an executable circuit covering $C^* \subseteq C$ exists. By Lemma 4.1 we have $|C^*| \leq D$.

Let $Y^-_{opt} \subseteq Y^-$ be the set covering of minimum cardinality. By the classical analysis of Greedy-Set-Cover, $|Y^{-\prime}| \leq |Y^-_{opt}|\mathcal{H}(|X|)$. By construction, $|X| = |R'|$, so $|Y^{-\prime}| \leq |Y^-_{opt}|\mathcal{H}(|R'|)$.

Let C' and C_{opt} be the corresponding circuit coverings for $Y^{-\prime}$ and Y^-_{opt}, respectively. By construction $|C'| = |Y^{-\prime}|$ and $|C_{opt}| = |Y^-_{opt}|$. Thus, $|C'| \leq |C_{opt}|\mathcal{H}(|R'|)$.

By the antecedent $|C'| > D\mathcal{H}(|R'|)$, thus $D\mathcal{H}(|R'|) < |C'| \leq |C_{opt}|\mathcal{H}(|R'|)$. Which implies that $D < |C_{opt}|$.

By assumption, we have $|C^-| \leq D$. Therefore $|C^-| \leq D < |C_{opt}| \rightarrow |C^-| < |C_{opt}|$. This is a contradiction. Thus, an executable circuit covering $C^- \subseteq C$ does NOT exist.

□/

A.6 Executability of Paging Strategies

Lemma A.4 (4.3) *Let $D = \max d$, such that Equations 4.1, 4.2, 4.3 are satisfied. An executable strategy circuit exists for* any *collection $C' \subseteq C$ such that $|C'| \leq D$.*

Proof:

Given any $C' \subseteq C$, $T^-_\delta(C') \leq T^-_\delta(C) \leq 1$. So Equation 4.1 is satisfied.

Furthermore, $B^-_\delta(C') \leq B^-_\delta(C)$ and, given $|C'| \leq D$, $B_\gamma(|C'|, |F^n|) \leq B_\gamma(D, |F^n|)$. By definition, D satisfies $2 * B^-_\delta(C) + B_\gamma(D, |F^n|) \leq M$. Therefore $2 * B^-_\delta(C') + B_\gamma(|C'|, |F^n|) \leq M$ and Equation 4.2 is satisfied.

Finally, given $|C'| \leq D$, $T_\gamma(|C'|, |F^n|) \leq T_\gamma(D, |F^n|)$. By definition, D satisfies $T_\gamma(D, |F^n|) + T_\rho B^-_\delta(C) \leq n$. Therefore $T_\gamma(|C'|, |F^n|) + T_\rho B^-_\delta(C') \leq n$ and Equation 4.3 is satisfied.

□/

This proof implicitly requires monotonically non-decreasing strategy functions. Since the strategy functions depend only on the size of inputs and not the actual sets, it is straightforward to explicitly make this assumption.

A.7 Upper Bound on Easy Circuit-Covering

Lemma A.5 (4.5) *If $C' \subseteq C$ is a circuit-covering produced by Greedy-Set-Cover applied to X and Y in the construction of Section 4.4, then $|C'| \leq |R'|$.*

Proof:

Greedy-Set-Cover picks a set that covers at least one new member of X on each iteration. Therefore, there are at most $|X|$ iterations and at most $|X|$ sets selected for the covering. Since Y is constructed in one-to-one relationship with C and $X = R'$, then the corresponding circuit covering C' is at most $|C'| \leq |R'|$.

□/

A.8 Existence of Easy Circuit-Covering

Theorem A.2 (4.3) *If $|R'| \leq D$ then Greedy-Set-Cover always produces an executable circuit covering C', $|C'| \leq D$.*

Proof:

By Lemma 4.5, any set covering produced by Greedy-Set-Cover satisfies $|C'| \leq |R'|$. Furthermore, by Theorem 4.2, any set covering constructed from C and R has a circuit covering of equal cardinality. If $|R'| \leq D$, then $|C'| \leq D$. Therefore, by Lemma 4.3, C' is an executable circuit covering.

□/

A.9 Upper Bound on On-line Cache Resources for Easy Circuit-Covering

Lemma A.6 (4.6) *For any circuit paging problem described by possible situations sets R, circuits C, cache size M, and maximum pages D, if a circuit covering exists then an executable circuit covering can always be found in polynomial time if*

$$M = 2 * B_\delta^*(C) + B_\gamma(|R'|, |F^n|)$$

Proof:

In the text we assume that Equation 4.2 is binding. Therefore, as long as this equation is satisfied then an executable strategy can be constructed as specified in Lemma 4.4.

If $M = 2 * B^*_\delta(C) + B_\gamma(|R'|, |F^n|)$ then we can satisfy Equation 4.2 for any covering C' such that $|C'| \leq |R'|$. By Lemma 4.5 we know that Greedy-Set-Covering will produce such a covering in polynomial time, if one exists.

□/

A.10 Circuit-Specific Lower Bound Based On Greedy-Set-Cover

Corollary A.2 (4.2)

Let $Y^{-\prime} \subseteq Y^-$ be a set covering produced by Greedy-Set-Cover(X, Y^-) and let $y^*(Y^{-\prime}) = \max_{\{y_S \in Y^{-\prime}\}} |y_S|$ be the size of the largest set in the covering. Furthermore, let $C' = \{c_S \in C | y_S \in Y^{-\prime}\}$ be the corresponding circuit covering and let D' be the maximum d satisfying Equations 4.1, 4.2, 4.3 with $C = C'$.

If $|C'| > D'\mathcal{H}(y^*(Y^{-\prime}))$ then there does NOT exist an executable circuit covering for any collection $C^* \subseteq C$ such that $B^*_\delta(C') \leq B^*_\delta(C^*)$.

Proof:

Let $Y^-_{\text{opt}} \subseteq Y^-$ be the set covering of minimum cardinality. By the classical analysis of Greedy-Set-Cover, $|Y^{-\prime}| \leq |Y^-_{\text{opt}}|\mathcal{H}(y^*(Y^-))$. Furthermore, it can be immediately seen that Greedy-Set-Cover always chooses the largest set first, thus $y^*(Y^-) = y^*(Y^{-\prime})$ and $|Y^{-\prime}| \leq |Y^-_{\text{opt}}|\mathcal{H}(y^*(Y^{-\prime}))$.

Let C' and C_{opt} be the corresponding circuit coverings for $Y^{-\prime}$ and Y^-_{opt}, respectively. By construction $|C'| = |Y^{-\prime}|$ and $|C_{\text{opt}}| = |Y^-_{\text{opt}}|$. Thus, $|C'| \leq |C_{\text{opt}}|\mathcal{H}(y^*(Y^{-\prime}))$. By the antecedent $|C'| > D'\mathcal{H}(y^*(Y^{-\prime}))$, thus $D'\mathcal{H}(y^*(Y^{-\prime})) < |C'| \leq |C_{\text{opt}}|\mathcal{H}(y^*(Y^{-\prime}))$. Which implies that $D' < |C_{\text{opt}}|$.

Assume that an executable circuit covering $C^* \subseteq C$ exists such that $B^*_\delta(C') \leq B^*_\delta(C^*)$. By Lemma 4.4 and the given decision-making circuit size bound there

exists a D^* such that

$$\begin{aligned} |C^*| &\leq D^* \\ T_\delta^*(C^*) &\leq 1 \\ 2 * B_\delta^*(C') + B_\gamma(D^*, |F^n|) &\leq M \\ T_\gamma(D^*, |F^n|) + T_\rho B_\delta^*(C') &\leq n \end{aligned}$$

By definition, D' is the maximum bound that satisfies the latter two equations. So, $D^* \leq D'$ and we can replace D^* by D' in those equations and maintain feasibility. Also, since our problem statement assumes that all decision-making circuits are individually executable, we have $T_\delta^*(C') \leq 1$. Thus we have shown that there exists a D^* such that

$$\begin{aligned} |C^*| \leq D^* &\leq D' \\ T_\delta^*(C') &\leq 1 \\ 2 * B_\delta^*(C') + B_\gamma(D', |F^n|) &\leq M \\ T_\gamma(D', |F^n|) + T_\rho B_\delta^*(C') &\leq n \end{aligned}$$

However, since $|C_{\text{opt}}| \leq |C^*|$, from above we have $D' < |C_{\text{opt}}| \leq |C^*| \rightarrow D' < |C^*|$. This is a contradiction. Thus, an executable circuit covering $C^* \subseteq C$ such that $B_\delta^*(C') \leq B_\delta^*(C^*)$ does NOT exist.

□/

A.11 Relationship Between $|\mathcal{O}|$, $\bar{x}$, and cf

Lemma A.7 (6.1) *Given a common reduction factor of* $cf \in [0, 1]$, *such that all but* $\bar{x} \in [0, 10]$ *flight operations may be sampled at the rate of* $100cf$ *per cycle, the maximum number of flight operations,* $|\mathcal{O}| \geq 10$, *that may be accommodated in a given schedule are*

$$|\mathcal{O}| \leq \frac{10 - \bar{x}}{cf} + \bar{x}.$$

Proof: The total number of flight operations $|\mathcal{O}|$ are partitioned into $\bar{x}$ worst-case sampling requirements (100) and $|\mathcal{O}| - \bar{x}$ reduced sampling requirements ($100cf$). The combined number of instantiations must be less than 1000. Therefore,

$$\begin{aligned} 100\bar{x} + 100cf(|\mathcal{O}| - \bar{x}) &\leq 1000 \\ \bar{x} + cf(|\mathcal{O}| - \bar{x}) &\leq 10 \\ \bar{x} + cf|\mathcal{O}| - cf\bar{x} &\leq 10 \\ |\mathcal{O}| &\leq (10 - \bar{x})/cf + \bar{x} \end{aligned}$$

□/

A.12 Executability Under Empirical Study

Lemma A.8 (7.1) *Let* $0 \leq B^*_\delta(C) \leq 1000$ *be the maximum number of bits of any conditional schedule* $c_S \in C$.

Given a collection of conditional schedules C, *overriding maximum pages/server space* D^*, *number of flight operations* $|\mathcal{O}| \geq 10$, *worst-case sampling frequency* $\bar{x} = 100$, *and conditional sampling frequency* $0 \leq cf \leq 1$, *such that*

$$|\mathcal{O}| \leq \frac{10 - \bar{x}}{cf} + \bar{x}$$

a collection of conditional schedules $C' \subseteq C$ *is* executable *if and only if*

$$|C'| \leq \min_{\{c_S \in C'\}} d_S.$$

where $d_S = (|\mathcal{O}| * 100 * cf * D^*)/B_\delta(c_S)$, *for each* $c_S \in C$.

Proof:

First note that, $|C'| \leq (|\mathcal{O}| * 100 * cf * D^*)/B_\delta(c_S)$ implies that $|C'| \leq D^*$. This can be seen by noting that $B_\delta(c_S) \geq |\mathcal{O}| * 100 * cf$, for all $c_S \in C$.

Now, it remains to be shown that $|C'| \leq \min_{\{c_S \in C'\}} d_S$ if and only if $|C'| * B_\delta^-(C') \leq |\mathcal{O}| * 100 * cf * D^-$

$\Rightarrow$ Assume $|C'| * B_\delta^-(C') \leq |\mathcal{O}| * 100 * cf * D^-$. Then for all $c_S \in C'$, $B_\delta(c_S) \leq (|\mathcal{O}| * 100 * cf * D^-)/|C'|$. In other words, $|C'| \leq (|\mathcal{O}| * 100 * cf * D^-)/B_\delta(c_S)$, for all $c_S \in C'$; and $|C'| \leq \min_{\{c_S \in C'\}}(|\mathcal{O}| * 100 * cf * D^-)/B_\delta(c_S)$.

$\Leftarrow$ Assume $|C'| \leq \min_{\{c_S \in C'\}}(|\mathcal{O}| * 100 * cf * D^-)/B_\delta(c_S)$.

$B_\delta^-(C') \geq \min_{\{c_S \in C'\}} B_\delta(c_S)$.

Therefore, $(|\mathcal{O}| * 100 * cf * D^-)/B_\delta^-(C') \leq \min_{\{c_S \in C'\}}(|\mathcal{O}| * 100 * cf * D^-)/B_\delta(c_S)$. In other words, $|C'| \leq (|\mathcal{O}| * 100 * cf * D^-)/B_\delta^-(C')$ or $|C'| * B_\delta^-(C') \leq |\mathcal{O}| * 100 * cf * D^-$.

□/

A.13 MILP Provides Correct and Complete Solution

Theorem A.3 (7.1) *MSCMILP is a correct and complete enumeration scheme that finds an executable conditional scheduling solution or shows that no executable conditional scheduling solution exists.*

Proof (Sketch):

If-then constraints satisfy Lemma 7.1. Other constraints provide covering that satisfies Lemma 5.1.

□/

Bibliography

[1] Agre, Philip E. and Chapman, David. Pengi: An Implementation of a Theory of Activity. *Proceedings AAAI-87. Seattle, Washington.* AAAI, 1987, 268–272.

[2] Apgar, Scott, Greenwald, Lloyd, and Lopresti, Daniel P., *The Lark Project: Design of a Highly Parallel Programmable Logic Array.* Technical Report CS-91-04, Brown University Department of Computer Science, Providence, RI, January 1991.

[3] Barto, Andrew G., Bradtke, Steven J., and Singh, Satinder P., Learning to Act using Real-time Dynamic Programming. *Submitted to AIJ Special Issue on Computational Theories of Interaction and Agency.* (1993).

[4] Bellare, M., Goldwasser, S., Lund, C., and Russell, A., Efficient Probabilistic Checkable Proofs and Applications to Approximation. *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing.* 1993, 294–304.

[5] Blanchini, Franco, Queyranne, Maurice, Rinaldi, Franca, and Ukovich, Walter, A feedback strategy for periodic network flows. *Networks,* (to appear).

[6] Blanchini, Franco and Ukovich, Walter, Linear Programming Approach to the Control of Discrete-Time Periodic Systems with Uncertain Inputs. *Journal of Optimization Theory and Applications,* **78**(3) (1993).

[7] Boddy, Mark and Dean, Thomas, Solving Time-Dependent Planning Problems. *Proceedings IJCAI 11. Detroit, Michigan.* IJCAII, 1989, 979–984.

[8] Boutilier. Craig and Dearden. Richard. Using Abstractions for Decision Theoretic Planning with Time Constraints. *Proceedings AAAI-94. Seattle. Washington.* AAAI. 1994.

[9] Brooks. Rodney A.. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation.* **2** (1986) 14–23.

[10] Carpenter. T.. Driscoll. K.. Hoyme. K.. and Carciofini. J.. ARINC 659 Scheduling: Problem Definition. *Proceedings of the 15th IEEE Real-Time Systems Symposium. San Juan. Puerto Rico.* 1994.

[11] Chapman. David. Penguins Can Make Cake. *AI Magazine.* **10**(4) (1989) 45–50.

[12] Chvátal. V.. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research.* **4** (1979) 233–235.

[13] Cormen. Thomas H.. Leiserson. Charles E.. and Rivest. Ronals L.. *Introduction to Algorithms.* (The MIT Press / McGraw Hill. Cambridge. Massachusetts. 1990).

[14] Crescenzi. P. and Kann. V.. *A Compendium of NP Optimization Problems.* Technical Report SI/RR-95/02. Department of Computer Science. University of Rome "La Sapiensa". Oct 1995.

[15] Dean. Thomas and Boddy. Mark. An Analysis of Time-Dependent Planning. *Proceedings AAAI-88. St. Paul. Minnesota.* AAAI. 1988. 49–54.

[16] Dean. Thomas and Greenwald. Lloyd. A Formal Description of the Transportation Problem. *DARPA Knowledge-Based Planning and Scheduling Initiative. First Annual Report.* March 1992.

[17] Dean. Thomas. Kaelbling. Leslie. Kirman. Jak. and Nicholson. Ann. Planning With Deadlines in Stochastic Domains. *Proceedings AAAI-93. Washington. D.C..* AAAI. 1993. 574–579.

[18] Dean. Thomas and Lin. Shieu-Hong. Decomposition Techniques for Planning in Stochastic Domains. *Proceedings of the 1995 International Joint Conference on Artificial Intelligence.* 1995.

[19] Dean. Thomas and Wellman. Michael. *Planning and Control.* (Morgan Kaufmann. San Mateo. California. 1991).

[20] Dean. Thomas L.. Greenwald. Lloyd. Kaelbling. Leslie Pack. Kirman. Jak. and Nicholson. Ann. *Time-Critical Planning and Scheduling Research at Brown University.* Technical Report CS-94-41. Brown University Department of Computer Science. Providence. RI. October 1994.

[21] Etzioni. Oren. Embedding Decision-Analytic Control in a Learning Architecture. *Artificial Intelligence.* **49**(1-3) (1991) 129–160.

[22] Fikes. Richard E.. Hart. Peter E.. and Nilsson. Nils J.. Learning and Executing Generalized Robot Plans. *Artificial Intelligence.* **3** (1972) 251–288.

[23] Fohler. Gerhard. Realizing Chnages of Operational Modes with a Pre Run-Time Scheduled Hard Real-Time System. *Proceedings of the Second International Workshop on Responsive Computer Systems.* 1992.

[24] Garey. Michael R. and Johnson. David S.. *Computers and Intractibility: A Guide to the Theory of NP-Completeness.* (W. H. Freeman and Company. New York. 1979).

[25] Garvey. Alan and Lesser. Victor. *A Survey of Research in Deliberative Real-Time Artificial Intelligence.* Technical Report 93-84. University of Massachusetts Department of Computer Science. Amherst. MA. November 1993.

[26] Gat. Erann. Integrating Reaction and Planning in a Heterogeneous Asynchronous Architecture for Mobile Robot Navigation. *SIGART Bulletin.* **2**(4) (1991).

[27] Georgeff. Michael P. and Lansky. Amy L.. Reactive Reasoning and Planning. *Proceedings AAAI-87. Seattle. Washington.* AAAI. 1987. 677–682.

[28] Gerber, Richard, Pugh, William, and Saksena, Manas. Parametric Dispatching of Hard Real-Time Tasks. *IEEE Transactions on Computers*. **44**(3) (1995).

[29] Ginsberg, Matthew L.. Universal planning: An (almost) universally bad idea. *AI Magazine*. **10**(4) (1989) 40–44.

[30] Good, I. J.. Twenty-Seven Principles of Rationality. Godambe, V. P. and Sprott, D. A.. (Eds.). *Foundations of Statistical Inference*. (Holt, Rinehart, Winston, 1971). 108–141.

[31] Greenwald, Lloyd. How to Avoid Thinking on Your Feet. *Proceedings of the AAAI Spring Symposium on Extending Theories of Action*. Stanford, California, 1995.

[32] Greenwald, Lloyd and Dean, Thomas. *Package Routing in Transportation Networks with Fixed Vehicle Schedules: Formulation, Complexity Results and Approximation Algorithms*. Technical Report CS-92-26. Brown University Department of Computer Science, Providence, RI, June 1992.

[33] Greenwald, Lloyd and Dean, Thomas. *Deliberation Scheduling for Time-Critical Scheduling in Stochastic Domains*. Technical Report CS-94-35. Brown University Department of Computer Science, Providence, RI, September 1994.

[34] Greenwald, Lloyd and Dean, Thomas. Monte Carlo Simulation and Bottleneck-Centered Heuristics for Time-Critical Scheduling in Stochastic Domains. *ARPI Planning Initiative Workshop*. 1994.

[35] Greenwald, Lloyd and Dean, Thomas. Solving Time-Critical Decision-Making Problems with Predictable Computational Demands. *Second International Conference on AI Planning Systems*. 1994.

[36] Greenwald, Lloyd and Dean, Thomas. Time-Critical Scheduling in Stochastic Domains (Abstract). *Proceedings AAAI-94. Seattle, Washington*. AAAI. 1994.

[37] Greenwald, Lloyd and Dean, Thomas. Anticipating Computational Demands When Solving Time-Critical Decision-Making Problems. Goldberg, K., Halperin, D., Latombe, J.C., and Wilson, R., (Eds.). *The Algorithmic Foundations of Robotics*. A. K. Peters, Boston, MA, 1995. Proceedings from the workshop held in February, 1994.

[38] Greenwald, Lloyd and Dean, Thomas. Package Routing in Transportation Networks with Fixed Vehicle Schedules. *Networks*, **27** (1996).

[39] Greenwald, Lloyd and Yuhan, A. Hanyong. *A Parallel Processing Proposal for Transient Signal Processing Algorithms*. Technical report, AT&T Bell Laboratories Technical Memorandum, 1989.

[40] Halldórsson, Magnús M., *Approximating Set Cover via Local Improvements*. Technical Report IS-RR-95-0002F, Japan Advanced Institute of Science and Technology, Mar 1995.

[41] Hammond, Kristian J., Opportunistic Memory. *Proceedings IJCAI 11, Detroit, Michigan*, IJCAII, 1989.

[42] Horvitz, Eric J., Reasoning About Beliefs and Actions Under Computational Resource Constraints. *Proceedings of the 1987 Workshop on Uncertainty in Artificial Intelligence, Seattle, Washington*, 1987.

[43] Horvitz, Eric J., *Computation and Action Under Bounded Resources*. Ph.D. Thesis, Stanford University Departments of Computer Science and Medicine, Stanford, California, 1990.

[44] Howard, Ronald A., Information Value Theory. *IEEE Transactions on Systems Science and Cybernetics*, **2**(1) (1966) 22–26.

[45] Hoyme, K. and Driscoll, K., SAFEbus(tm). *IEEE Aerospace Electronics and Systems Magazine*, (1993) 34–39.

[46] Isaacson, Dean L. and Madsen, Richard W., *Markov Chains, Theory and Applications*, (John Wiley & Sons, New York, 1976).

[47] Johnson, D., Approximation Algorithms for Combinatorial Problems, *Journal of Computer and System Sciences*, **9** (1974) 256–278.

[48] Kaelbling, Leslie Pack, Goals as Parallel Program Specifications, *Proceedings AAAI-88, St. Paul, Minnesota*, AAAI, 1988, 60–65.

[49] Lesser, Victor R., Pavlin, Jasmina, and Durfee, Edmund, Approximate Processing in Real-Time Problem Solving, *AI Magazine*, **9**(1) (1988) 49–61.

[50] Lovasz, L., On the Ratio of Optimal Integral and Fractional Covers, *Discrete Mathematics*, **13** (1975) 383–390.

[51] Lyons, D. M. and Hendriks, A. J., Testing Incremental Adaptation, *Second International Conference on AI Planning Systems*, 1994.

[52] Musliner, David John, *CIRCA: The Cooperative Intelligent Real-Time Control Architecture*, PhD thesis, The University of Michigan, Ann Arbor, MI, 1993.

[53] Nicholson, Ann and Kaelbling, Leslie Pack, Toward Approximate Planning in Very Large Stochastic Domains, *Proceedings of the AAAI Spring Symposium on Decision Theoretic Planning*, Stanford, California, 1994.

[54] Nilsson, Nils, *Triangle Tables: a Proposal for a Robot Programming Language*, Technical report, SRI International, 1985.

[55] Nilsson, Nils, Teleo-Reactive Programs for Agent Control, *Journal of Artificial Intelligence Research*, **1** (1994) 139–158.

[56] Pryor, Louise and Collins, Gregg, Reference Features as Guides to Reasoning about Opportunities, *Proceedings Fourteenth Annual Conference of the Cognitive Science Society*, Cognitive Science Society, 1992, 230–235.

[57] Puterman, Martin L., *Markov Decision Processes*, (John Wiley & Sons, New York, 1994).

[58] Ramadge, Peter and Wonham, Murray. The Control of Discrete Event Systems. *Proceedings of the IEEE*, **77**(1) (1989) 81–98.

[59] Russell, Stuart J. and Subramanian, Devika. Provably Bounded-Optimal Agents. *Artificial Intelligence Research*, **2** (1995) 575–609.

[60] Russell, Stuart J. and Wefald, Eric H.. Principles of Metareasoning. Brachman, Ronald J.. Levesque, Hector J.. and Reiter, Raymond. (Eds.). *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*. (Morgan-Kaufmann, Los Altos, California, 1989). 400–411.

[61] Russell, Stuart J. and Wefald, Eric H.. *Do the Right Thing: Studies in Limited Rationality*. (MIT Press, Cambridge, MA, 1991).

[62] Schoppers, Marcel J.. Universal Plans for Reactive Robots in Unpredictable Environments. *Proceedings IJCAI 10, Milan, Italy*. IJCAII, 1987, 1039–1046.

[63] Schoppers, Marcel J.. In Defense of Reaction Plans as Caches. *AI Magazine*, **10**(4) (1989) 51–60.

[64] Sha, L.. Rajkumar, R.. Lehoczky, J.. and Ramamritham, K.. Mode Change Protocols for Priority-Driven Preemptive Scheduling. *Real-Time Systems*, **1**(3) (1989) 243–265.

[65] Shih, W-K.. Liu, J. W. S.. and Chung, J-Y.. Fast Algorithms for Scheduling Imprecise Computations. *Proceedings of the Real-Time Systems Symposium, Santa Monica, California*, IEEE, 1989, 12–19.

[66] Simmons, Reid. Coordinating Planning, Perception, and Action for Mobile Robots. *SIGART Bulletin*, **2**(4) (1991).

[67] Simon, Herbert A. and Kadane, Joseph B.. Optimal Problem-Solving Search: All-or-None Solutions. *Artificial Intelligence*, **6** (1975) 235–247.

[68] Slavík, Petr. *A Tight Analysis of the Greedy Algorithm for Set Cover*. Technical report, Department of Computer Science, SUNY at Buffalo, Nov 1995.

[69] Winston, Wayne L., *Introduction to Mathematical Programming: Applications and Algorithms*, (PWS-Kent, 1991).

[70] Zilberstein, Shlomo, *Operational Rationality through Compilation of Anytime Algorithms*, PhD thesis, University of California at Berkeley, 1993.