Optimization of Linear Max-Plus Systems with Application to Timing Analysis

by

Elizabeth A. Walkup

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

University of Washington

1995

Approved by____
(Chairperson of Supervisory Committee)

Program Authorized
to Offer Degree______________________________________

Date____

University of Washington

Abstract

Optimization of Linear Max-Plus Systems with Application to Timing Analysis

by Elizabeth A. Walkup

Chairperson of the Supervisory Committee: Professor Gaetano Borriello
Department of Computer Science
and Engineering

This work provides a joint solution to two different problems — the temporal verification of hardware interface logic, and the solution and optimization of linear max-plus systems.

The interface timing verification problem seeks to determine the maximum time separations possible among the input and output signal changes of two or more interconnected hardware modules. These possible separations are then compared against the time separations allowed by the hardware modules' communication protocols. We present here the ShortCircuit algorithm, which is the first correct algorithm for this problem. In addition, we discuss the difficulties inherent in solving the interface hardware timing synthesis problem, which asks us to choose time separations between signal transitions to ensure that the timing requirements of all hardware modules' communication protocols are met.

The ShortCircuit algorithm leads to the first reasonable method for solving arbitrarily sized systems in the linear max-plus algebra. The linear max-plus algebra is similar to the familiar linear algebra except that the binary maximum and scalar addition operations take the place of addition and scalar multiplication.

The linear max-plus system solution employs an iterative technique in which the system is re-phrased with a system of simpler constraints. Maximum solutions to special subsets of these constraints are then found using the max-plus matrix closure operation. Each such maximum solution bounds the system's maximum solution

from above, and guides the search for a new subset which yields a tighter bound on the system's solution. The technique is easily extended to maximize and minimize linear max-plus expressions over a linear max-plus system and is thus the max-plus analogue of linear programming.

While the practical motivation and examples discussed in this work are drawn from the realm of hardware interface logic, the solutions provided apply to any system of events whose temporal relationships can be described as a combination of two primitives: synchronization and bounded delay. An event synchronizes a set of events if it occurs immediately after all events of the set have occurred. Two events have a bounded delay relationship if one always happens within a specified bounded time interval after the other.

Table of Contents

List of Figures

List of Tables

ACKNOWLEDGMENTS

This work would never have come about without the support and guidance of many persons, only some of whom I have listed below.

I thank Lynn Friesen, John Hackwell, and the late Charles Randall, all of the Aerospace Corporation, for getting me started doing some real science.

I thank my advisor, Gaetano Borriello, the other members of my committee — Paul Beame, Steven Burns, Carl Ebeling, and Ted Klastorin — and faculty members Martin Tompa and Vic Klee of the University of Washington, and Andrew Kahng of UCLA for their academic and technical guidance.

I thank my family, without whose support and encouragement this could not have been attempted.

And I thank just a few of the friends — Ronelle Coburn, Suzanne Bunton, Alex Klaiber, Eric Koldinger, Raj Vaswani, Dylan McNamee, Radhika Thekkath, David "Paco" Hubbell, David "Pardo" Keppel, David "Pablo" Cohn, Devon Wiel, Diana Allen, Scott Hauck, Pai Chou, Ross Ortega, Soha Hassoun, Lauren Bricker, Matthew Bricker-Mounts, Kevin Mounts, Miche, Bruce and Griffin Baker-Harvey, Melanie Lewis Fulgham, and Greg Barnes — who over the past several years have provided me with all the extras I needed to continue, whether it was scintillating conversation, hot meals, C++ programming tips, or a sympathetic ear.

Part I

Introduction to Timing Analysis

Chapter 1

CONCEPTUAL INTRODUCTION

One approach to verifying the correct behavior of complex systems is to model such systems as a collection of simpler sub-systems. Such a technique may be required when the verification task is to be shared among several persons or organizations, and may be desirable when different verification techniques are best suited to different portions of the system. In some cases such a technique may be applied repeatedly, resulting in a hierarchy of sub-systems.

Once the verification tasks corresponding to each sub-system are completed, their results must be combined and interpreted as they apply to the entire system. Depending on both the choices which were made when partitioning the system into sub-systems and the properties being studied, some information about the system may be lost. While an attempt is made to prevent this as much as possible, generally speaking, there is a tradeoff between how accurate the final model is and how long it takes both to verify the individual sub-systems and to combine their information to model the whole.

While there are many system properties which may be verified in this manner, this work is concerned with **temporal properties** of systems. In particular, we are interested in determining the possible ranges of timing separations between durationless **discrete events** occurring in a system. In our study, we will use discrete events to mark the beginning and ending points of activities carried out by a system.

Determining the time separations between events in our systems is an example of the "recombining" of sub-system information. In our case we wish to efficiently and correctly determine the possible relative times of pairs of discrete events in a system with the following four properties:

(1) each sub-system is a single activity with discrete start and completion events,
(2) each sub-system activity requires an unknown but bounded time to complete,
(3) the times of individual activities are independent of each other, and
(4) the activities are partially ordered.

It is this last item, the partial ordering of activities, which makes this problem useful, as it allows us to represent concurrent behavior. It is also what makes the problem difficult since two activities which have no apparent ordering may be forced into a temporal relationship based on their relationships to other activities.

There are a wide variety of applications and domains in which temporal knowledge is desired and in which the sub-system processes meet the above requirements. Applications of this sort may range from the planning and management of large projects, as done with PERT charts [SPO58, HM61], where the individual jobs comprising the project have durations measured in weeks or months, to the design and verification of computer hardware interfaces, for which the basic operations have durations measured in nanoseconds.

This second application, the verification and synthesis of timing properties of hardware interface logic, occurs when one wishes to interconnect two or more off-the-shelf hardware devices. These devices follow specific protocols that precisely define two things: the desired timing behavior of the environment into which the device will be connected, and the responses of the hardware to changes of the values of its inputs. The specified behavior of the device is guaranteed only if its environment — those devices with which it interacts — meets the given timing specification.

In order to more correctly model the behavior of hardware interfaces, we extend our system model to include additional timing information which may span different sub-systems. We can then model any system whose behavior can be represented as the combination of two primitives — **bounded delay** and **synchronization**. For two events e_i and e_j occurring at times t_i and t_j a **bounded delay** relationship between them is expressed as the pair of inequalities

$$t_i + \delta \leq t_j \leq t_i + \Delta. \tag{1.1}$$

This indicates that event e_j happens between δ and Δ time units after event e_i. When an event happens just as soon as all events in a set of two or more occur, we say that that event is the **synchronization** of the others. That e_k **synchronizes** events e_i and e_j is represented with the single equation

$$t_k = \max(t_i, t_j). \tag{1.2}$$

For systems whose temporal behavior is expressed with the synchronization and bounded delay primitives, we can express both of the following fundamental problems:

- **timing verification problems:** Does a given system respect a set of restrictions made upon the relative times of its events, and
- **timing synthesis problems:** Can such restrictions be met by altering the ordering or duration of specific activities represented by the system?

A key contribution of this work is the `ShortCircuit` algorithm. Developed to verify the temporal behavior of hardware interface logic, this algorithm provides the first correct method for solving these timing verification problems. This is particularly useful since, as mentioned above, the synchronization and bounded delay primitives can express the temporal behavior of systems other than interface hardware.

A sizable body of research in control theory studies the **max-plus algebra**. Many interesting problems, including the timing verification problem for systems whose behavior is described with the synchronization and bounded delay primitives, can be expressed using systems of equations over the **linear max-plus algebra**. This algebra has form similar to the familiar linear algebra, except that the binary **maximum** and **scalar addition** operations replace **addition** and **scalar multiplication**. Previous research results for the max-plus algebra [CMQV89, BCOQ92] have concentrated on properties that result from its being a member of the larger class of **dioid** algebras. Dioids are structures composed of two **monoids** with an **idempotent** first operation. However, the max-plus algebra has useful properties not common to all dioids. We will exploit these additional properties to develop the first practical method for solving arbitrary linear max-plus systems and show that solving this problem is equivalent to solving the same interface hardware timing problems `ShortCircuit` solves.

The timing synthesis problem mentioned above is, unfortunately, much more difficult than timing verification. This is not surprising since the synthesis problem asks us not simply to determine how a given system behaves, but instead to modify a given system so that it produces the desired temporal behavior. Even when the allowed range of system modifications is quite small, such modifications combine to interact throughout the system so that a local change may have global system repercussions. In addition, it is easy to define many different classes of synthesis problems based on the types of system modifications which are allowed. We will show that even the most straightforward and simple timing synthesis problems are NP-hard.

Together the timing verification and synthesis problems presented and solved in this work not only provide a further refinement of the techniques available for the synthesis of interface glue logic, but provide a greater understanding of the max-plus algebra and a fundamental algorithm for the solution and optimization of linear max-plus systems.

1.1 Thesis Organization

This thesis consists of five major parts, whose dependencies are pictured in Figure 1.1. The first part, *Introduction to Timing Analysis*, consisting of this chapter and the next, is intended to give the reader a practical, intuitive grasp of the fundamental mathematical problems studied here without requiring any specific knowledge of an application domain such as interface timing verification or synthesis.

Part II, *Interface Timing Fundamentals*, consists of Chapter 3, in which we define and discuss the interface timing verification and synthesis problems, and Chapter 4, which covers previous solutions to the interface logic verification problem and introduces the `ShortCircuit` algorithm for interface timing verification.

Part III, *Linear Max-Plus Systems*, reviews the max-plus algebra, and provides a general solution technique for linear max-plus systems. The basis for this technique is the `UBCsolv` method for solving a restricted class of linear max-plus systems analogous to the interface timing verification problems solved by `ShortCircuit`. An extension to `UBCsolv` allows for the solution and optimization of linear max-plus systems. Since this material is of more general interest and applicability, it is described independently of interface timing.

Part IV, *Practical Algorithmic Solutions*, consists of Chapter 8 in which the `ShortCircuit` algorithm is discussed in greater detail than in Chapter 4, and Chapter 9 which discusses the requirements of interface timing synthesis algorithms and gives preliminary results towards developing practical timing synthesis algorithms.

Part V, *Conclusions and Related Work*, concludes with a discussion of future work and open problems and places this work within a greater context of related work than earlier chapters do.

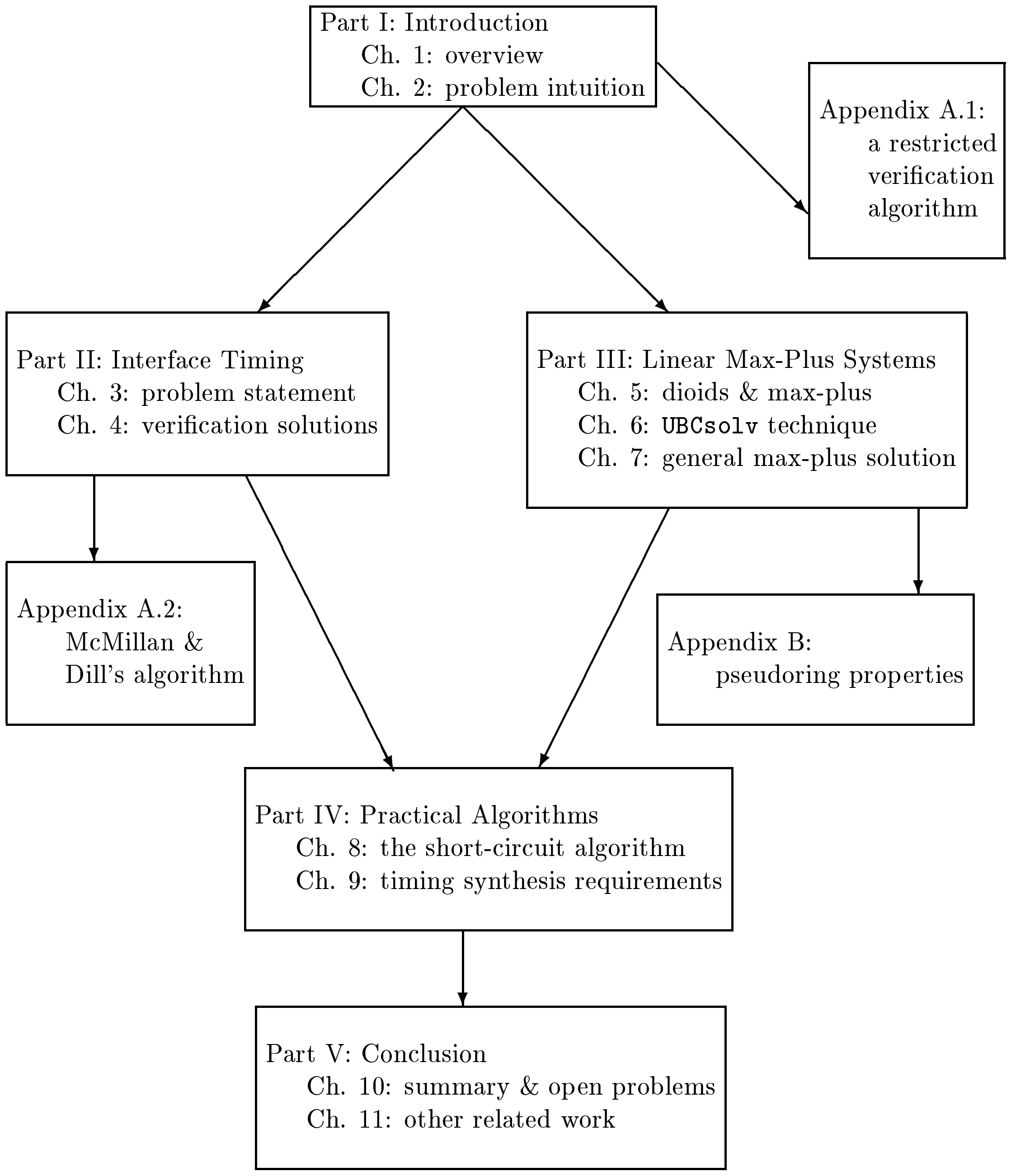

Part I: Introduction
Ch. 1: overview
Ch. 2: problem intuition
Appendix A.1:
a restricted
verification
algorithm
Part II: Interface Timing
Ch. 3: problem statement
Ch. 4: verification solutions
Part III: Linear Max-Plus Systems
Ch. 5: dioids & max-plus
Ch. 6: UBCsolv technique
Ch. 7: general max-plus solution
Appendix A.2:
McMillan &
Dill's algorithm
Appendix B:
pseudoring properties
Part IV: Practical Algorithms
Ch. 8: the short-circuit algorithm
Ch. 9: timing synthesis requirements
Part V: Conclusion
Ch. 10: summary & open problems
Ch. 11: other related work

Figure 1.1: Chapter ordering dependencies.

Chapter 2

SYSTEMS OF SYNCHRONIZATIONS AND BOUNDED DELAYS

As stated in Chapter 1, this work concerns itself with determining and manipulating the timing properties of systems whose temporal behavior can be described using only the bounded delay and synchronization primitives of Equations 1.1 and 1.2. This chapter is intended to give the reader an intuitive feel for the types of systems these primitives can represent. To this end, we will define two classes of systems built from the synchronization and bounded delay primitives. The first, and more intuitive, will include only the four initial properties listed in Chapter 1, and the second will model the full range of behavior allowed by the synchronization and bounded delay primitives.

Throughout this chapter we will employ an example modeling a two-person morning carpool. Within this framework, we will be able to define and give motivation for the **timing verification** and **timing synthesis** problems as they apply to the two classes of systems. In addition, we will introduce a general constraint form, the **upper bound constraint (UBC)**, and its corresponding graph-theoretical representation. UBCs are capable of describing all of the systems studied in this work, and are a key structure underlying the `UBCsolv` method of Part III and its proof of correctness.

2.1 Synchronization of Independent Bounded-Delay Processes

We begin by introducing a basic class of systems of synchronizing processes. This class is well known, corresponding to both PERT charts, an Operations Research construct used to plan the Polaris missile project [SPO58, HM61], and Vanbekbergen's "Type II" constraints [Van93] for digital logic. While this class can express a varied range of system behaviors, it is insufficient to represent the interface timing verification and synthesis problems. In Section 2.3 we will build upon this class to produce the system model we require.

Morning Commute			
Process	Description	Duration in minutes	Predecessor Relations
B1	Bert showers, dresses, and calls Ernie	[35,45]	B1 $\prec$ B2
E	Ernie showers, dresses, and eats	[50,60]	B2 $\prec$ B3
B2	Bert eats breakfast	[10,15]	B3 $\prec$ BE
B3	Bert drives to Ernie's home	[15,20]	B1 $\prec$ E
BE	Bert and Ernie drive to work	[30,40]	E $\prec$ BE

Figure 2.1: A morning carpool plan.

The systems we first wish to discuss are most intuitively thought of as a set of partially-ordered independently-executing bounded-time processes. Here a **bounded-time process** is simply an activity which takes some unknown but bounded positive time to complete. More formally speaking, a bounded-time process, p_i, has associated with it a pair $[\delta_i, \Delta_i]$ where $0 \leq \delta_i \leq \Delta_i$. Whenever process p_i is initiated it completes in some time d_i, where $\delta_i \leq d_i \leq \Delta_i$. The duration d_i may either vary within the bound $[\delta_i, \Delta_i]$ for different executions of the process, or may take an exact time which is unknown beyond the tolerance of the bound.

When a system is composed of several bounded-delay processes, they are said to be **independent** if the knowledge of the exact execution times d_i corresponding to any set of processes p_i cannot narrow the bounds of $[\delta_j, \Delta_j]$ on the value of d_j for any p_j not in that set. Another way of saying this is that a valid simulation of the system exists for d_i's chosen independently within their respective bounds $[\delta_i, \Delta_i]$.

In any interesting system of processes, it will be the case that some processes must require that other processes complete before they begin. We can express this with the predecessor relation, "$\prec$," on the processes, where $p_i \prec p_j$ indicates that process p_i must complete before p_j begins. In order for the predecessor relation to be valid, it must result in the processes being **partially ordered**. This corresponds to making sure that no process must precede itself in order to satisfy the local ordering information of the $\prec$ relation.

Figure 2.1 summarizes one such system – a morning carpool for two persons, Ernie and Bert. When Bert wakes he showers, dresses and then calls Ernie, who then

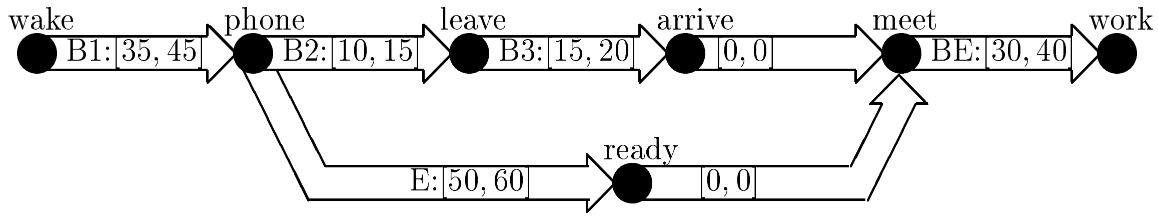


Figure 2.2: Graphical representation of the carpool of Figure 2.1.

begins to get ready for work. While Ernie gets ready, Bert eats breakfast and drives to Ernie's house, from which location they drive to work together. The left hand side of the table in Figure 2.1 gives a list of the events and processes that each of the carpoolers participate in, with the minimum and maximum time ranges required to complete each activity. The right hand side of the figure lists the necessary orderings of the activities.

Although the time bounds on the activities and their partial ordering are sufficient to represent this commute, it is convenient to include **events** in our representation. Events are durationless and serve to mark the "points of interest" where processes begin and end.

Figure 2.2 gives a graph-theoretical representation of the information in Figure 2.1. Events are represented as nodes in the graph, and processes, or activities, are represented with directed arcs. Each process arc originates at the event at its tail, concludes with the event at its head, and is labeled with its duration bounds. The partial ordering of the processes is determined by the arcs' entrance into and exit from the event nodes – events happen once all processes entering them complete, and processes cannot happen until the event whose node they leave occurs. Since the times at which Ernie becomes ready and at which Bert arrives at Ernie's home may be of interest to us — perhaps we wish to determine how long each might wait for the other — they each have their own event, respectively `ready` and `arrive`. Incorporating the arc representing their drive to work might then seem problematic, since it must follow both events, but this is easily remedied here by inserting zero-length dummy activities leading from each event to the event describing their departure

Original Equations				
wake + 35	$\leq$	phone	$\leq$	wake + 45
phone + 10	$\leq$	leave	$\leq$	phone + 15
leave + 15	$\leq$	arrive	$\leq$	leave + 20
phone + 50	$\leq$	ready	$\leq$	phone + 60
max(arrive, ready)	$\leq$	meet	$\leq$	max(arrive, ready)
meet + 30	$\leq$	work	$\leq$	meet + 40

Figure 2.3: Equations corresponding to Figure 2.2.

together. This event, `meet` can then be seen to synchronize the two commuters.

If we assume that once a process can begin it does so immediately, then it is easy to represent the possible relative times of the events in the system with a simple set of inequalities. For the system in Figure 2.2, these equations are given in Figure 2.3.

We are now ready to formalize the systems discussed above with the following definition.

Definition 2.1 *A system of* **independent bounded-delay synchronizations***, abbreviated* **I-BDS***, is a 5-tuple $\langle N, A, B, P, I\rangle$ where*

- *N is a set of nodes, also called events, and $|N|$, the size of N is n.*
- *A is a set of activities, also called processes,*
- *B is a function bounding the duration of the activities. It is given as $B : A \rightarrow R^+ \times R^+$ where R^+ is the set of non-negative real numbers, and for all $a \in A$, if $B(a) = [\delta(a), \Delta(a)]$, then $0 \leq \delta(a) \leq \Delta(a)$,*
- *$P : N \rightarrow 2^A$ is the event predecessor function indicating which activities in A terminate with a given event in N,*
- *$I : A \rightarrow N$ is the activity predecessor function; it indicates which event in N* **initiates***, or is at the tail of a given activity in A.*

For a given I-BDS, with $N = \{e_0, e_1, \ldots, e_{n-1}\}$ a **valid timing***, t is any function $t : N \rightarrow R$ such that there exists a value $d(a)$ for each activity a so that for every*

Table 2.1: Time separations for events in carpooling arrangement.

Time Separations							
from | to	wake	phone	leave	arrive	ready	meet	work
wake	0	45	60	80	105	105	$\boxed{145}_{\text{Q2}}$
phone	-35	0	15	35	60	60	100
leave	-45	-10	0	20	50	50	90
arrive	-60	-25	-15	0	35	35	75
ready	-85	-50	-35	$\boxed{-15}_{\text{Q4}}$	0	0	40
meet	-85	-50	-35	-15	0	0	40
work	$\boxed{-115}_{\text{Q1}}$	-80	$\boxed{-65}_{\text{Q3}}$	-45	-30	-30	0

event $e_i \in N$

$$t(e_i) = \max_{a \in P(e_i)} (t(I(a)) + d(a)), \tag{2.1}$$

where $d(a)$ satisfies $\delta(a) \leq d(a) \leq \Delta(a)$. The value of $d(a)$ represents any possible duration of activity a within its given bounds $[\delta(a), \Delta(a)]$.

2.2 Verification of I-BDS's

Using the graphical or constraint data in Figures 2.2 and 2.3, it is possible to ask questions about the relative times of the events represented. Assuming that Bert wakes at 7am and that once he arrives at Ernie's home he waits patiently in the car for him, you might ask the following questions, which are given here with their answers:

$Q1$:	What is the earliest time they get to work?	8:55
$Q2$:	What is the latest time they get to work?	9:25
$Q3$:	What is the least amount of time Bert spends in the car?	65 minutes
$Q4$:	Will Ernie ever be ready when Bert arrives?	No.

These are all called **timing verification** questions. In general, they ask whether certain restrictions on the relative times of events hold in a given system. For now, we will confine ourselves to asking what is the maximum or minimum time separation possible between two events in a given system.

Each of these questions can be answered by calculating the **maximum time separations** between each pair of events in the system. The maximum time separation between the times $t(e_i)$ and $t(e_j)$ of two events e_i and e_j is defined as the maximum value of

$$t(e_j) - t(e_i) \tag{2.2}$$

occurring in any valid timing of the system as defined in Definition 2.1. Although question $Q1$ above asks for the value of

$$\min(t(\text{work}) - t(\text{wake})), \tag{2.3}$$

this is equivalent to

$$-\max(t(\text{wake}) - t(\text{work})), \tag{2.4}$$

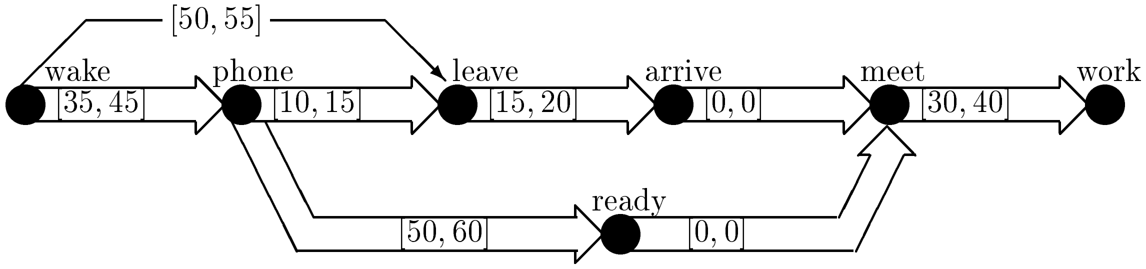
and thus may be readily obtained from the maximum separation between `work` and `wake`. The separations between all pairs of events in the carpooling arrangement of Figure 2.1 are given in Table 2.1, with the specific separations answering each question surrounded by a labeled box.

Note how synchronization comes into play here — although the total driving time from Bert's home to work via Ernie's home is between 45 and 65 minutes, it will always take Bert at least 65 minutes and may take as long as 95 minutes to get to work once he leaves home.

An efficient algorithm, due to McMillan and Dill [MD92], for determining absolute bounds on the relative times of pairs of events in an I-BDS can be found in Appendix A.1.

2.3 Synchronization of Dependent Bounded-Delay Processes

While the I-BDS model of synchronizing processes with independent bounded delay can model a wide variety of systems, there are several obvious and appealing extensions which it cannot model. In this section we discuss and motivate some of those extensions, and introduce a new model, **Dependent Bounded-Delay Synchronizations (D-BDS)**, in which certain types of dependencies among the execution times of different processes are allowed. We will also introduce the **upper bound constraint** type (**UBC**), which gracefully represents the relative times of events in a



Original Equations				
wake + 35	$\leq$	phone	$\leq$	wake + 45
phone + 10	$\leq$	leave	$\leq$	phone + 15
leave + 15	$\leq$	arrive	$\leq$	leave + 20
phone + 50	$\leq$	ready	$\leq$	phone + 60
max(arrive , ready)	$\leq$	meet	$\leq$	max(arrive, ready)
meet + 30	$\leq$	work	$\leq$	meet + 40
Additional Equations				
wake + 50	$\leq$	leave	$\leq$	wake + 55

Figure 2.4: Graphical representation of carpooling arrangement with additional constraint information.

D-BDS system, and which comprises the core structure underlying the methods and proofs of Part III.

An example of the sort of additional system behavior we might like to model can be found if we consider Bert's first two activities in Figure 2.1. Suppose, that although Bert takes 35 to 45 minutes to dress, and 10 to 15 minutes to eat breakfast, it never takes him less than 50 or more than 55 minutes to complete these two activities in series. Under the I-BDS model the times for both activities are independent, and we must assume, given the data in Figure 2.1 that it may take Bert as few as 45 or as many as 60 minutes to perform both. If we allow ourselves to add additional constraints to the diagram, we might express our new knowledge with the additional thin arc shown in Figure 2.4. This thin arc indicates a constraint independent of the other constraints in the figure. The equations represented by the full set of arcs are given at the bottom of the figure.

This new relationship between the `wake`, and `leave` events could not be represented by adding an IBDS process with duration $[50, 55]$ between the two events.

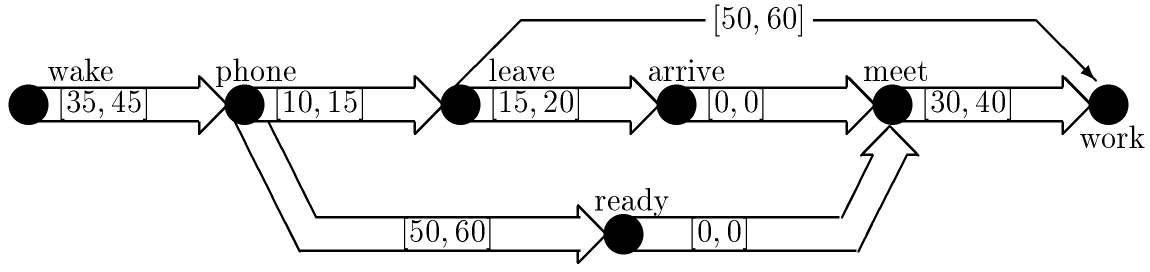
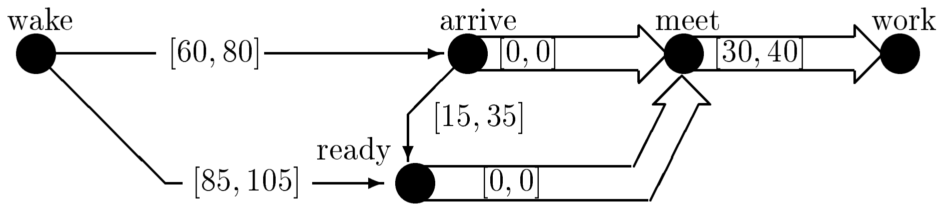


Figure 2.5: Graphical representation of carpooling arrangement with inconsistent additional constraint information.

While such an addition would assure that `leave` never happens earlier than 50 minutes after `wake`, it would still allow the two events to be 60 minutes apart, as occurs when all process require their maximum times.

It is crucial to recognize that adding a constraint with bound of $[50, 60]$ from the time Bert leaves until the time the two arrive at work, as is shown in Figure 2.5, expresses something other than a total driving time of 50 to 60 minutes. Such constraints should be interpreted as a guarantee on the time separation between the events it relates. This interpretation of the added constraint, coupled with the other information as given in Figure 2.5 leads to what must be an inconsistent system. The information that Bert arrives at work no later than 60 minutes after he leaves home yet takes at least 30 minutes to get from Ernie's home to work combine to require that the two meet no later than 30 minutes after Bert leaves home. Similarly, since Ernie is ready no fewer than 50 minutes after Bert's call, he must be called at least 20 minutes before Bert leaves. However, we know that Bert leaves home no later than 15 minutes after the phone call.

Even when such extensions are neither desired nor required, it is sometimes useful to abstract away some of the information about the individual processes, as in the system diagram of Figure 2.6. Here we do not know the times of the individual processes, but instead have been given bounds on the relative times of some of the system events. We refer to the graph in Figure 2.6 as being **causally incomplete** because we do not know the activities which cause each of the events to occur. This can be useful when the underlying I-BDS structure of the system is quite large, and



Equations				
wake + 60	$\leq$	arrive	$\leq$	wake + 80
wake + 85	$\leq$	ready	$\leq$	wake + 105
arrive + 15	$\leq$	ready	$\leq$	arrive + 35
max(arrive, ready)	$\leq$	meet	$\leq$	max(arrive, ready)
meet + 30	$\leq$	work	$\leq$	meet + 40

Figure 2.6: Graphical representation of carpooling arrangement with incomplete causal structure.

may be required when working with systems others have built since it is common practice in some application areas to hide the implementation details of the system from the user and only summarize their behavior.

To handle both of these situations, we now introduce dependent bounded-delay synchronizations (D-BDSs) and their associated constraint type, the upper bound constraint. Section 2.3.1 will demonstrate that the two classes, I-BDSs and D-BDSs, are not equivalent.

Definition 2.2 *A system of* **dependent bounded-delay synchronizations***, abbreviated* **D-BDS***, is a set of m equations over n variables $\{t_0, \ldots t_{n-1}\}$, where each equation is of the form*

$$t_i \leq \max(t_0 + a_0, t_1 + a_1, \ldots t_k + a_k). \tag{2.5}$$

Each t_i can be taken to indicate the time of an event e_i. The a_i's are reals and may be negative, and not all t_i terms need be present in the right hand side of the equation. We call these constraints **upper bound constraints (UBCs)***.*

Note that all the equations we've seen in Figures 2.3, 2.4, and 2.6 can be expressed in the form of Equation 2.5. Figure 2.7 gives the UBC translation of the equations

UBC Equations									
wake	$\leq$	phone	-	35	phone	$\leq$	wake	+	45
phone	$\leq$	leave	-	10	leave	$\leq$	phone	+	15
leave	$\leq$	arrive	-	15	arrive	$\leq$	leave	+	20
phone	$\leq$	ready	-	50	ready	$\leq$	phone	+	60
arrive	$\leq$	meet			meet	$\leq$	max(ready, arrive)		
ready	$\leq$	meet							
meet	$\leq$	work	-	30	work	$\leq$	meet	+	40
wake	$\leq$	leave	-	50	leave	$\leq$	wake	+	55

Figure 2.7: UBC representation of carpooling arrangement

in Figure 2.4. For example, the equation

$$\max(\text{arrive}, \text{ready}) \leq \text{meet} \tag{2.6}$$

is split into two UBCs,

$$\begin{aligned} \text{arrive} &\leq \text{meet} \quad \text{and} \qquad (2.7) \\ \text{ready} &\leq \text{meet}, \qquad (2.8) \end{aligned}$$

since if `meet` happens later than the maximum of two events, it naturally happens later than each of them individually. Whenever a constant offset appears on the left hand side of such an equation, it can be subtracted from both sides of the equation to maintain the UBC form. This is expressed more formally with the following proposition.

Proposition 2.1 *Any I-BDS $\langle N, A, B, P, I\rangle$ can be expressed using the following UBCs:*

- *For every $e_i \in N$*
$$t(e_i) \leq \max_{a \in P(e_i)} (t(I(a)) + \Delta(a)), \tag{2.9}$$
- *and for every combination of $e_i \in N$ and $a \in P(e_i)$,*
$$t(I(a)) \leq t(e_i) - \delta(a). \tag{2.10}$$

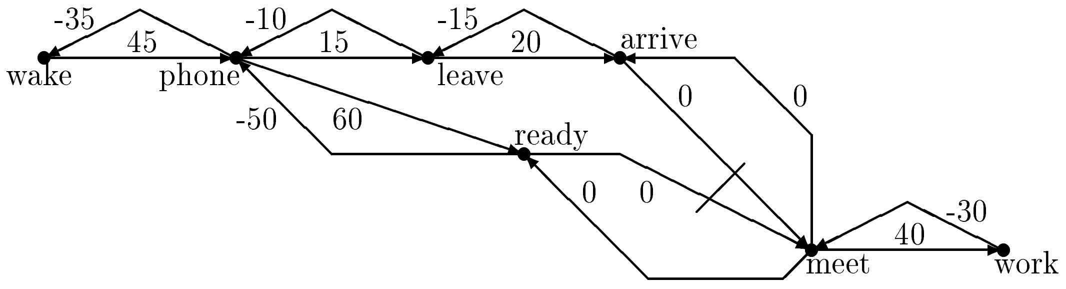


Figure 2.8: UBC representation of carpooling arrangement of Figure 2.2.

For a system of upper bound constraints, there is a natural graph-theoretical representation as follows:

- for each event e_i in the system, there is a corresponding **node**, labeled e_i,
- for each UBC bounding $t(e_i)$ from above, there is a set of arcs, one for each term $t(e_j) + a_j$. Each such arc
  - is directed from node e_j to node e_i, and
  - has label a_j,
- arcs arising from the same UBC are grouped together by a single bar passing through all such arcs near their heads.

Figure 2.8 gives the graph-representation corresponding to the diagram and equations of Figure 2.2.

Previous attempts to develop methods for determining the maximum time separations of D-BDSs are the subject of Chapter 4 and Appendix A.2, with Part III providing our new method.

2.3.1 I-BDS and D-BDS are not identical

One of the most notable differences between the I-BDS and D-BDS classes is that D-BDS allows the use of negative a_i terms in any UBC. This allows one to express what appear to be nonsensical causalities. For example, the relationship among the

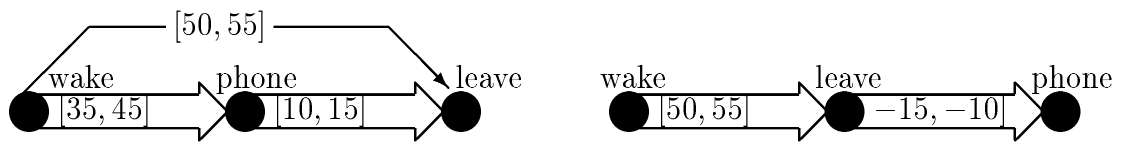


Figure 2.9: A nearly I-BDS representation.

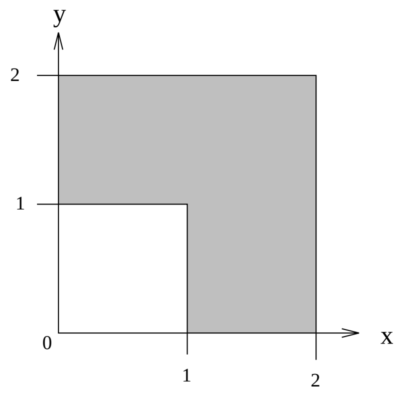
events `wake`, `phone` and `leave` in Figure 2.4, while not expressible as an I-BDS, can be correctly related as shown in Figure 2.9.

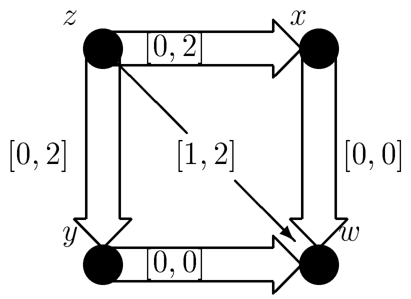
One might therefore be tempted to suppose that allowing negative terms in the bounds $[\delta, \Delta]$ of an I-BDS would allow one to model UBC-based systems as I-BDS systems and exploit existing efficient methods for determining maximum time separations in I-BDSs. However, it turns out that there are UBC-based systems which cannot be modeled using this extension.

Figure 2.10 gives an example of a system which can be expressed with UBCs but not as an I-BDS. Here we assume that we wish to model a system in which in response to an event z at time zero, events x and y occur throughout the range of times expressible with the equations in the table at the top left corner of the figure. This behavior is shown as the shaded area in the diagram at top right. This can be done by creating the system with the four events, $\{w, x, y, z\}$ pictured at the bottom left of the figure. The bottom right of the figure gives the system's UBC description.

No I-BDS, no matter what size, and no matter whether its bounds $[\delta(a), \Delta(a)]$ may contain negative values or not, can model this same behavior. To demonstrate this, we note that since the behavior is in response to event z, the I-BDS graph must have no activity preceding z and all events must be produced through some partial ordering of activities which begins with z. If this is the case, then when all bounds $d(a)$ on all activities a occur at their minimum values, $\delta(a)$, we will have the earliest possible times of x and y relative to z. If the I-BDS can achieve the full range of values indicated for x and y, then these minimum values must both be 0. However, we should not have both $x - z$ and $y - z$ equal to 0 at the same time and thus an I-BDS cannot model this system.

Original Equations				
z	$\leq$	x	$\leq$	$z+2$
z	$\leq$	y	$\leq$	$z+2$
$z+1$	$\leq$	$\max(x,y)$		





UBC Equations					
x	$\leq$	$z+2$	y	$\leq$	$z+2$
z	$\leq$	x	z	$\leq$	y
w	$\leq$	$\max(x,y)$	w	$\leq$	$z+2$
x	$\leq$	w	z	$\leq$	$w-1$
y	$\leq$	w			

Figure 2.10: Timing behavior which cannot be expressed with an I-BDS.

2.4 Simulation of I-BDS's and D-BDS's

While the main focus of this work is the analytical study of the possible time separations between events in certain classes of systems, it is important to note that many applications require not only an analytical solution, but the ability to simulate the behavior of a system. There are several reasons why this may be the case including:

- the BDS system may interact with other systems which have been or can only be modeled as simulations, or
- the system designers may wish to study the simulated system behavior to determine if their specified system requirements actually produce the behavior they desire.

Because the times of the individual activities of an I-BDS are defined to be independent, chosing a wide variety of feasible times for the events is quite straightforward — this may be easily accomplished simply by randomly choosing a time for each activity within its given bound, and then calculating the times at which the events occur.

In the case of D-BDS systems, the problem is more complicated — a choice of time for one activity may restrict the feasible times for another. An example of this is found in Figure 2.4, in which the separation between the events `wake` and `phone` may be as great as 45 minutes, and the separation between `phone` and `leave` may be 15 minutes, but both pairs cannot exhibit their maximum separations in the same simulation. The simplest way to assure feasible times is to choose a time for one activity, calculate the possible maximum and minimum time separations relative to it, and then choose a time for a new activity such that it falls within its allowed ranges. This procedure is then repeated for each successive event. Related work in this area can be found in [KDC^{+}93, KDC91, Org91], which describe methods for simulating systems described with **timing diagrams**.

2.5 Timing Synthesis of Bounded Delay Processes

Recall from Chapter 1 the **timing synthesis** question, which asks, given a desired relationship among the time separations of a group of events, how can one manipulate the activities modeled by that system to yield that desired separation? Many different timing synthesis problems may occur, varying both in the type of systems to which they apply (I-BDS or D-BDS for example) and in how the relative timing and ordering of the activities may be altered.

For instance, for the carpool of Figure 2.1, given a wake-up time of 7am and that the driving times cannot be shortened, nor the order of activities be changed, how can Bert assure that he gets to work by 9am each day? In this case, note that there are several ways in which this can be accomplished without changing the partial ordering of activities. Two possible revisions are given in Figure 2.11.

In the first, Bert takes less time to dress and shower so that Ernie is telephoned earlier in the morning; in the second, Ernie makes changes to his morning routine to allow him to be ready within thirty minutes of Bert's call. Note that while the maximum separation between the time Bert wakes and the two of them arrive at

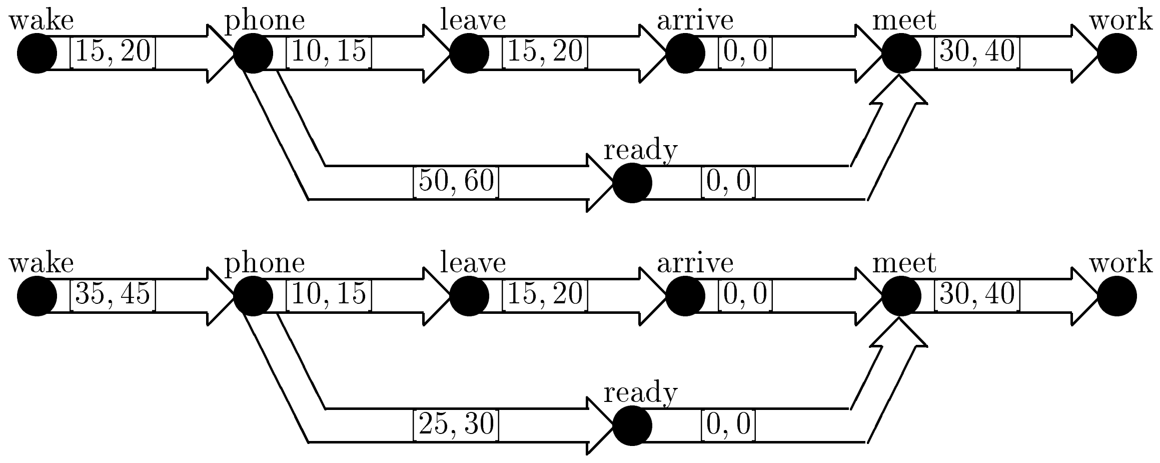


Figure 2.11: Graphical representation of two revised carpooling arrangements.

work is 120 minutes in both cases, Bert may prefer the second arrangement since under the first, he will always spend from 15 to 35 minutes waiting at Ernie's house, while under the second arrangement, he need only wait a maximum of 5 minutes, and sometimes Ernie will wait 10 minutes for him.

More generally, this problem can be represented as given in Figure 2.12 in which the variables α, β, and γ represent the times of the changeable, or "un-synthesized"

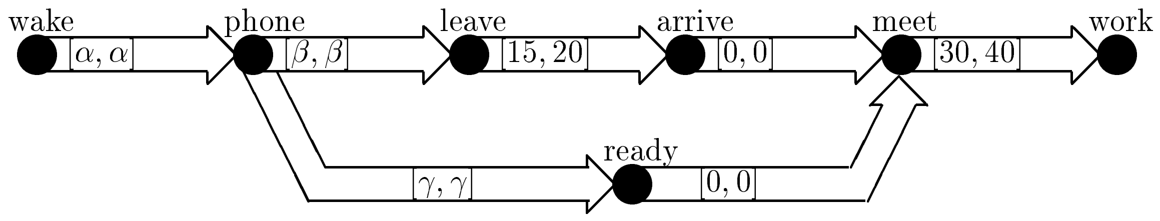


Figure 2.12: Graphical representation carpool arrangement with synthesis variables.

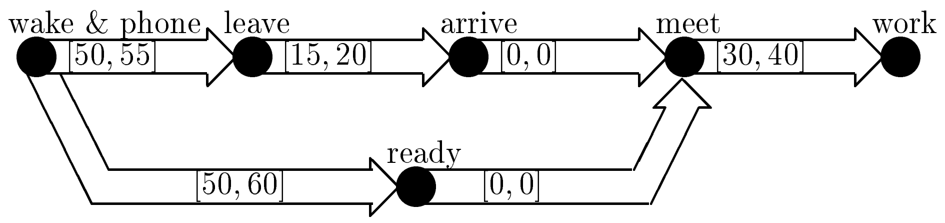


Figure 2.13: Graphical representation of revised carpooling arrangement.

activities. The maximum time separation between Bert waking and arriving at work is then

$$\alpha + \max(\beta + 20, \gamma) + 40, \tag{2.11}$$

and if we wish to meet the 120 minute constraint we must find a solution for which

$$\alpha + \beta \leq 60 \quad \text{and} \tag{2.12}$$

$$\alpha + \gamma \leq 80, \tag{2.13}$$

simultaneously.

The synthesis methods we will discuss in Chapter 9 are all for systems such as these in which where we are not allowed to re-arrange the order of activities. However, it should be noted that for many practical problems, including our commuting example, solutions in which we re-arrange the partial order of processes may in fact be worth investigating. The situation of Figure 2.13 may be desirable, since it does not require either party to revise the timing of his morning activities.

When revising the order of activities it is important to note that the relative orderings of some activities are required by their functions, but some relative activity orderings are the product of scheduling decisions and may be changed. An example of a required ordering is that Bert must drive to Ernie's house before they drive to work together, while an example of an arbitrary ordering is the order in which Bert dresses and eats breakfast. In general, the types of changes that are allowed in a synthesis problem depend intimately upon the application domain and tend to reflect functional information which is lost in the purely temporal abstraction given here.

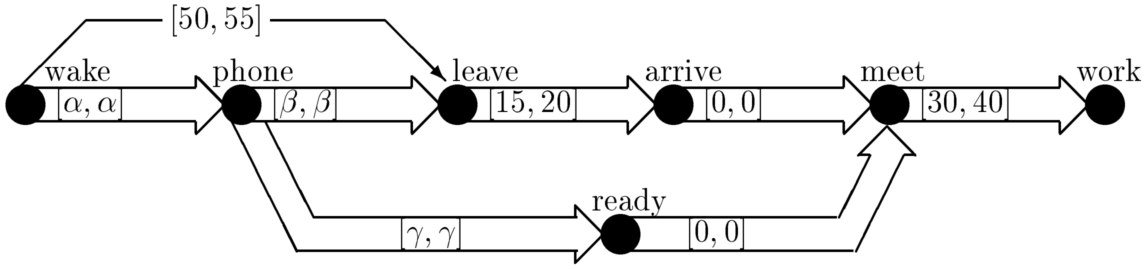


Figure 2.14: A nonsensical synthesis problem.

In addition, problems can be posed which make little physical sense — one is unlikely to see additional timing information as found in Figure 2.4 relating events whose times one is able to change through synthesis. Figure 2.14 gives an example of just such a nonsensical problem – we are allowed to choose any values for α and β and yet we are guaranteed that Bert will leave the house within 55 minutes of waking.

2.6 Summary

In this chapter, we have introduced two different classes of systems for modeling timing relationships between discrete events. The first, and simpler class, I-BDS systems, consists of a set of partially ordered activities requiring bounded time to complete; the second, D-BDS systems are an extension of I-BDS systems which allow additional constraints of the form $e_i \leq e_j + \alpha$ to relate the times of the events marking the start and finish of the activities. The two classes were shown to be inequivalent. In the context of these two system classes, we discussed the **timing verification** and **timing synthesis** problems and gave examples of each.

Part II

Interface Timing Fundamentals

Chapter 3

INTERFACE TIMING VERIFICATION AND SYNTHESIS

The interface timing verification and synthesis problems first mentioned in Chapter 1 are two of numerous problems that arise when assuring that hardware modules are correctly interconnected. Each hardware module has a set of **communication protocols** specifying the manner in which it may communicate with its environment. When two hardware modules are connected or **interfaced** to each other, each acts as the other's environment. If the two modules' protocols are complementary, then each will satisfy the other's restrictions on its environment. Should two such hardware modules not have complementary communication protocols, it is necessary to introduce **interface glue logic** between the two modules to "translate" communications between them.

Two fundamental problems in interface timing design are discussed in this chapter: interface timing verification and interface timing synthesis. Interface timing verification is the simpler of the two — given the communication protocols of two hardware modules and the interface circuitry connecting them, the goal is to check whether each device satisfies the other's timing requirements on the communication. Interface timing synthesis, which requires the design of an interface that will meet all such timing requirements, is much more difficult.

This chapter begins with a practical discussion and example of the specification of hardware module communication protocols and the design of interface glue logic, and finishes with a formal model of interface event timing.

3.1 Hardware Module Communication Protocols

In most cases, a hardware module is intended to communicate with other hardware modules, and thus its design must include a protocol for implementing that communication with those modules. A hardware module's interface protocol describes the **what, where** and **when** of communicating with it. Ideally, a module's protocol will require of its environment only those inputs and outputs which are inherent in an abstract description of the module's functionality. This protects the user from having to know about the inner structure of the module, and allows the module to be used in as general a fashion as possible.

For example, we begin with the following abstraction of a `Read` protocol for a simplified memory controller chip. The functional description of a memory read operation is that an `Address` is provided and the corresponding `Data` are returned to the user. Thus, at a minimum, the `Address` and `Data` are the **what** of the protocol. Furthermore, assuming the memory also has a `Write` operation, it is necessary to have an additional control input to indicate whether the requested operation is a read or a write. The **where** are the chip pins through which the address and operation selection enter the module and the data leave it. Assuming the user selects the `Read` option and provides the memory with an address, it must also know **when** the pins from which it will read data values are guaranteed to be driving the data value stored in memory at the given address.

In order to communicate this information, hardware designers use databooks to describe the behavior of hardware modules. Along with other physical information about the chip such as its dimensions, temperature and voltage tolerances and typical power consumption, a hardware module's databook entry describes the protocol suite for communicating with the hardware module. The databook entry for a given protocols is built using a simple vocabulary of signal level changes and timing constraints.

Figure 3.1 gives examples of these signal level changes and timing constraints as they are depicted in timing diagrams. At the top of the figure is a signal shown transitioning (left to right) from the states **valid (V)** to **low (L)** to **high (H)**. The states **low** and **high** indicate logic levels of zero and one, respectively, while **valid** is used to describe a logic level that is known to be either zero or one. At the middle of Figure 3.1 we see a signal transitioning from **high impedance (Z)** to **invalid (I)** to **valid** and back to invalid and then high impedance. The **high impedance** value indicates that no device is driving a value onto a given signal. This is used when one of several devices may provide the value on a signal wire. A common example of this can be found in those memories and busses that take in input data for a `Read` operation and provide output data from a `Write` operation through the same ports. In these cases the **invalid** signal level is used to indicate that a device is currently

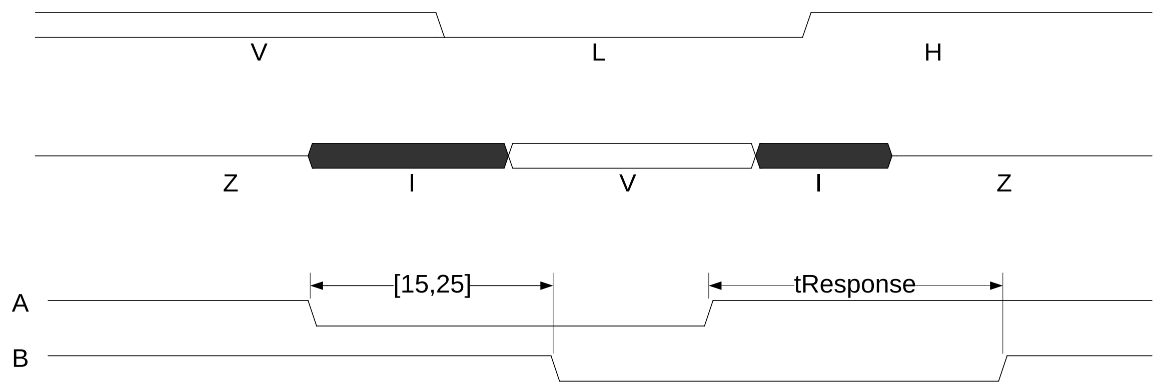


Figure 3.1: Signal level changes as depicted in timing diagrams.

driving the signal to some logic level, but that that level may not yet have reached the desired logic level.

At the bottom of Figure 3.1 we see how timing diagrams relate information about the relative times of signal level transitions. From the $H \rightarrow L$ transition on signal A to the $H \rightarrow L$ transition on signal B, there is an arc with label $[15, 25]$. This indicates that B's transition happens within 15 to 25 time units (generally nanoseconds) of A's. These time ranges may also be represented by a variable such as `tResponse`, whose bounds would be expressed in a table given with the timing diagram.

While engineers have used timing diagrams for decades to describe the timing behavior of electronic devices, their use as a specification method for CAD problems is a more recent invention. This is largely due to the fact that timing diagrams as they have been published in databooks require a high level of contextual information in order to be rendered unambiguous. The first attempt to formalize the timing diagram and thus make it suitable for use as input to CAD tools was due to Borriello [Bor88, Bor92]. Others have since used them to represent and study the timing properties of hardware interfaces [Gah90, BGM91, WB94a], hardware-software interfaces [WB93a, CWB94], and to specify system behavior for simulation [KDC91, KDC$^+$92, KDC$^+$93, Org91]. In addition, commercial products which facilitate the creation, editing, and constraint satisfaction of timing diagrams are now available [Gla93].

Figure 3.2 gives a sample databook entry for our abstract memory controller. Its databook entry consists of a pin diagram for the chip, **timing diagrams** for the `Read` and `Write` operations, and a table of time separations for the timing diagrams. Each timing diagram, along with the table of timing separations provides a partial ordering of the signal transitions, or **events**, of an interface protocol for the module. In this case, the `Read` protocol consists of the following events:

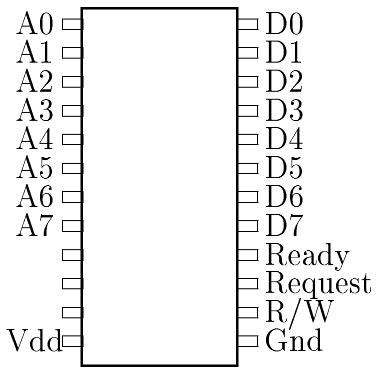
- user provides a memory address on pins `Addr:0` through `Addr:7`,
- user drives `R/W`, the operation selector, low,
- user drives `Request` low, activating the operation
- user waits until it detects the low pulse on `Ready`,
- user reads data from pins `Data:0` through `Data:7`, and
- user drives pin `Request` high, indicating it has finished reading the data.

Note that this protocol is a bit more complex than the one we sketched above. The additional signal `Request` allows the user to "pause" the memory while it changes other inputs, and to force the memory to hold its internal state, and therefore the output data value, constant for as long as the user requires. Similarly, the `Ready` signal allows the memory to indicate when its outputs are valid. Such an indication could come either from a single bit control output, as given here, or a guarantee that data are always valid within some time interval after the operation is initiated.

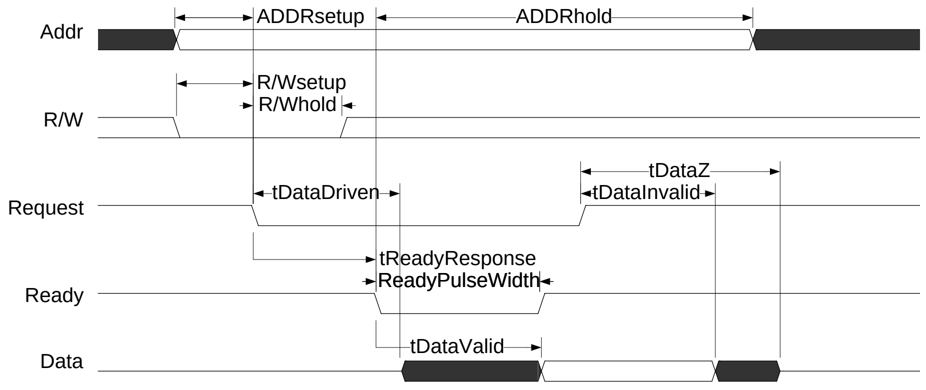
3.1.1 Connecting Hardware Modules

Consider the problem of connecting a memory controller to a bus. Both the controller and the bus have protocols for reading and writing data, but the two protocols may not fit together precisely. If it can be determined that together the memory's and bus's read protocols satisfy each others' requirements, then the two may be interfaced without additional hardware.

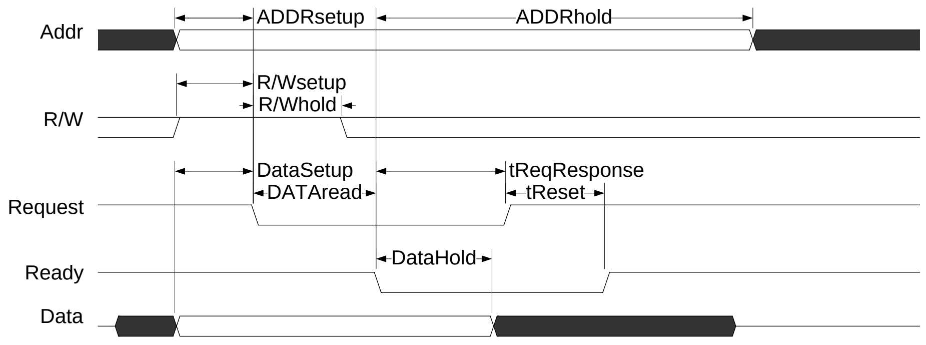
Figure 3.3 gives the read protocol a bus read operation we might wish to interface with the memory read operation of Figure 3.2. In order to determine if the two will interface properly, we must formally define the **requirement**, **delay**, and **guarantee** types of timing parameters appearing in the tables within each of the Figures 3.2 and 3.3. We do this in the next section, which shows that this problem may be phrased as a maximum timing separation problem over a D-BDS system.



Timing Parameters for Memory			
constraint	Min	Max	type
ADDRsetup	25	–	requirement
ADDRhold	15	–	requirement
R/Wsetup	25	–	requirement
R/Whold	15	–	requirement
DATAsetup	25	–	requirement
DATAhold	15	–	requirement
tReqResponse	0	–	requirement
tReadyResponse	35	80	delay
tDataValid	40	90	delay
tDataDriven	5	–	guarantee
tDataInvalid	0	–	guarantee
tDataZ	0	5	guarantee
ReadyPulseWidth	30	60	guarantee
DATAread	–	90	guarantee
tReset	–	30	guarantee



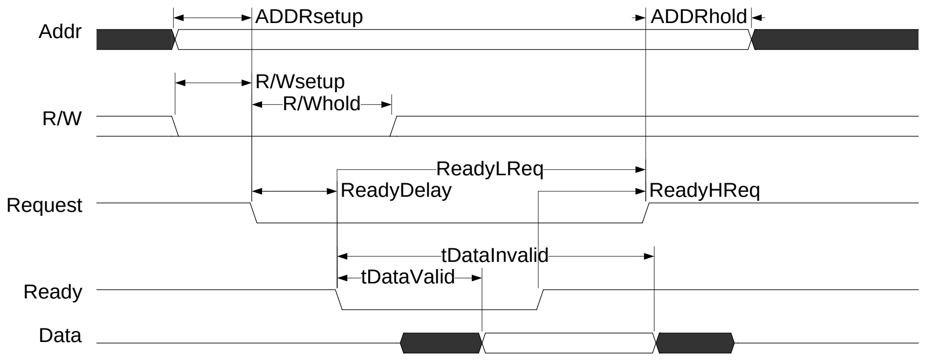
Read Operation



Write Operation

Figure 3.2: Databook entries for a simplified memory controller.

Timing Parameters for Bus			
constraint	Min	Max	type
tDataInvalid	140	–	requirement
tDataValid	–	100	requirement
ReadyDelay	30	–	requirement
ReadyLRequest	150	180	delay
ReadyHRequest	10	30	delay
ADDRsetup	30	–	guarantee
ADDRhold	20	–	guarantee
R/Wsetup	30	–	guarantee
R/Whold	20	–	guarantee



Read Operation

Figure 3.3: Databook read-from-device protocol for a simplified bus.

3.2 A Formal Model for Interface Timing

In Chapter 2, we introduced the upper bound constraint form (UBC) and indicated that we would later show how to solve an arbitrary system of such UBCs with the `UBCsolv` technique. In this section, we will show that UBCs are adequate to represent both the timing relationships and events required for the analysis and synthesis of interface timing.

We define our **events** to be transitions of signal values. These events can be explicit changes in signal values, as when the `Ready` signal of the read protocol in Figure 3.3 changes from high to low to indicate that memory has received the request, or implicit changes, such as the change of data values from invalid to valid in the same protocol. While it is quite probable that at least some of the invalid data bits will actually be correct, and thus not make a transition, we use this notion of "transition" to indicate the time at which the data are known to be correct.

The timing separations in the table at the top of Figures 3.2 and 3.3 are used to specify the relative times of protocol events and come in three types: **requirements, guarantees**, and **delays**. All three types are generally represented as a pair of min and max constants, $[\delta, \Delta]$, which apply *from* one event *to* another.

The most common example of a timing requirement is a **setup** constraint. An example of this can be found in the `Read` and `Write` operations of Figure 3.2 — in each operation the min value of 25 for the variable `ADDRsetup` tells us that the address provided to the memory must be stable for at least 25ns before the `Request` line is lowered. More generally, a **requirement** of $[\delta, \Delta]$ from event a to event c requires that t_a and t_c, the relative times of events a and c, satisfy the equation

$$t_a + \delta \leq t_c \leq t_a + \Delta, \tag{3.1}$$

if the hardware module is to respond correctly. When min, max pairs of the form $[-, \Delta]$ and $[\delta, -]$ occur, they indicate that only the right or left inequality is required. One may also think of the pair $[-, \Delta]$ as $[-\infty, \Delta]$ and of $[\delta, -]$ as $[\delta, +\infty]$.

Similarly, a **guarantee** of $[\delta, \Delta]$ from a to c indicates that the hardware guarantees that its responses will satisfy the timing relationship

$$t_a + \delta \leq t_c \leq t_a + \Delta \tag{3.2}$$

as long as all requirements are met.

Delay relationships are slightly more complex, as they are used to represent the causal structure of events. A **delay** of $[\delta_a, \Delta_a]$ from a to c has the same semantics as an I-BDS activity with bounds $[\delta_a, \Delta_a]$, initiated by event a, and terminating with event c. If this is the only delay *to* c, then the semantics are

$$t_a + \delta_a \leq t_c \leq t_a + \Delta_a, \tag{3.3}$$

as with a guarantee. However, if there is another delay to c, say $[\delta_b, \Delta_b]$ from b to c, then the timing relationship is

$$\max(t_a + \delta_a, t_b + \delta_b) \leq t_c \leq \max(t_a + \Delta_a, t_b + \Delta_b). \tag{3.4}$$

In general, if there are k delays from events 1 through k to event a, then all delays are combined into a single pair of inequalities as

$$\max(t_1+\delta_1, t_2+\delta_2, \ldots, t_k+\delta_k) \leq t_a \leq \max(t_1+\Delta_1, t_2+\Delta_2, \ldots, t_k+\Delta_k). \quad (3.5)$$

where $[\delta_i, \Delta_i]$ is the original delay range from event i to event a.

Some of the interface timing verification methods discussed in the next chapter add another constraint type similar to the delay, but with the following semantics:

$$\min(t_1+\delta_1, t_2+\delta_2, \ldots, t_k+\delta_k) \leq t_a \leq \min(t_1+\Delta_1, t_2+\Delta_2, \ldots, t_k+\Delta_k). \quad (3.6)$$

As we will see in Section 4.2, the timing verification problem over events related with guarantees, delays, and this additional "min-delay" constraint type was proven by McMillan and Dill [MD92] to be NP-complete. We do not include the min-delay type in our study for two reasons: many hardware interface protocols can be modeled without them, and when they do occur in a protocol there are generally far fewer of them than delays, thus making a branch and bound technique based upon the problem without min-delays a good choice for solving the NP-complete problem.

Note that the three timing constraint types given above in Equations 3.1, 3.2 and 3.4 are easily expressed with UBCs. Each requirement and guarantee as given in Equations 3.1 and 3.2 becomes the two inequalities

$$t_c \leq t_a + \Delta \quad \text{and} \quad (3.7)$$

$$t_a \leq t_c - \delta. \quad (3.8)$$

Although they contain no max term, these inequalities are still UBCs, since the max of one term $t_i + \delta_i$ is always itself. Translating delay inequalities into this form is nearly as easy. The delay relationship expressed in Equation 3.4 becomes the three equations:

$$\begin{aligned} t_c &\leq \max(t_a+\Delta_a, t_b+\Delta_b), && (3.9)\\ t_a &\leq t_c - \delta_a, && (3.10)\\ t_b &\leq t_c - \delta_b. && (3.11) \end{aligned}$$

The second two inequalities follow since if $\max(t_a+\delta_a, t_b+\delta_b) \leq t_c$, it must be the case that t_c is larger than each term $t_i + \delta_i$ on the left hand side of the inequality. A delay relationship combining k terms results in $k+1$ UBCs: one expressing the max of k terms $t_i + \Delta_i$, and k UBCs with a single $t - \delta_i$ on their right hand side as in Equation 3.11.

3.2.1 Interface Timing Verification

Given a system of events and the delay and guarantee relationships which govern their relative times, the interface timing verification problem asks whether these relationships guarantee that the time separations between pairs of events will fall within the range of all requirements between them. As in Section 2.2, we define the **maximum time separation**, or maximum skew, of event b relative to event a to be the maximum value of $t_b - t_a$ subject to all timing guarantees and delays. Once these skews are calculated for every pair of events, we check whether for all requirements $[\delta, \Delta]$ from event a to event b,

$$\begin{aligned} t_b - t_a &\leq \Delta \quad \text{and} \qquad (3.12) \\ t_a - t_b &\leq -\delta. \qquad (3.13) \end{aligned}$$

Generally we say that determining these maximum separations is sufficient, since checking them is straightforward.

Figure 3.4 gives the graphical UBC representation of the interface of the bus and memory read operations of Figures 3.3 and 3.2. Guarantees and delays supplied by the memory's protocol are shown in black, while those of the bus are shown in gray. The zero-length arcs from `Data valid` to `Data driven` and from `Data Z` to `Data invalid` are an example of constraints which are obtained from contextual knowledge of the device. They indicate that data may only be valid while the line is being driven.

Three of the memory's requirements — `ADDRsetup`, `R/Wsetup`, and `R/Whold` — are easily verified by comparing the required values in Figures 3.2 to the corresponding single arc lengths in Figure 3.4. However, it is not as straightforward to check the bus' requirement that the returned data not become invalid for at least 140ns after the `Ready` signal is lowered. The memory makes its corresponding guarantee relative to the time at which the `Request` signal is raised, requiring that we determine what the possible time separations of the lowering of the `Request` and `Ready` signals are.

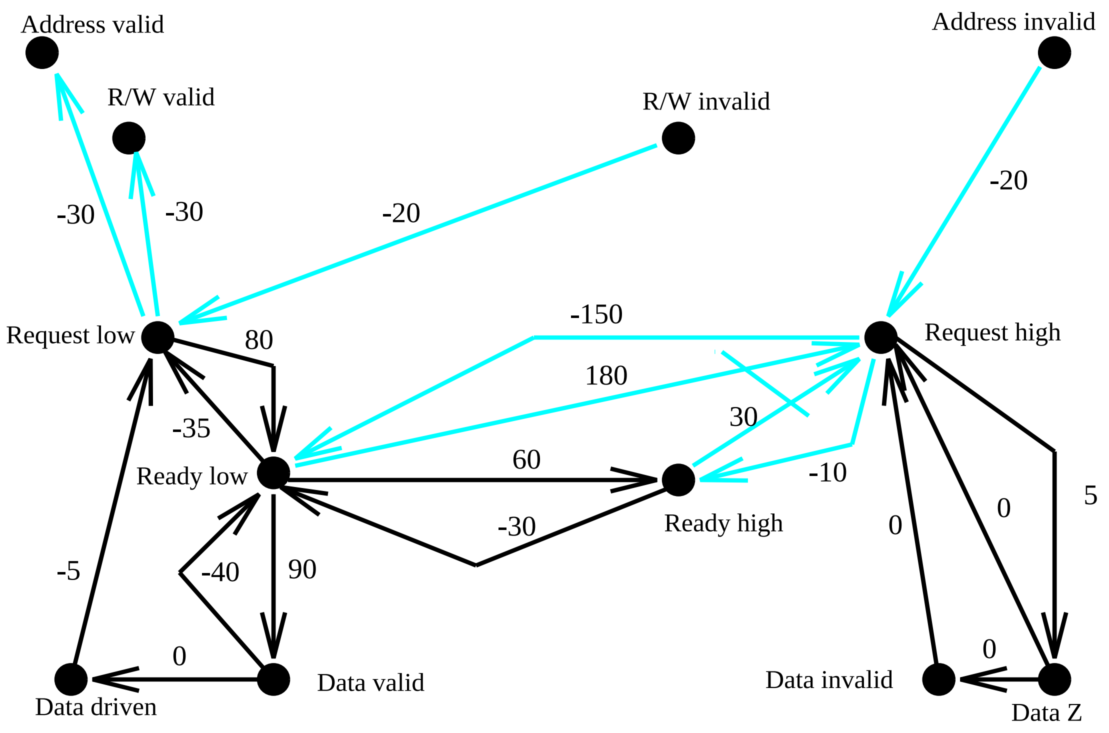


Figure 3.4: UBC representation of the interface of bus and memory read operations. Black arcs are supplied by the memory's protocol, gray by the bus' protocol.

We check the interface's timing requirements using a table of timing separations as we did in Table 2.1 The corresponding time separations for a subset of the events are given in Table 3.1. Those separations which must be compared to bus or memory timing requirements are outlined with a box. The timing separations pictured here can be compared to the bus' timing requirements and the memory's `ADDRhold` requirement. The maximum separation of -205ns from `Addr invalid` to `Request low` indicates a minimum 205ns hold on the `Addr` line, and thus that the memory's requirement of a minimum 15ns separation from `Request low` to `Addr invalid` is met. Similarly, the bus' requirements of a minimum 140ns for `tDataInvalid`, maximum of 100ns for `tDataValid`, and minimum of 30ns for `ReadyDelay` are also satisfied.

Table 3.1: Time separations for a subset of the events in Figure 3.4.

Time Separations							
from | to	Addr I	Req L	Req H	Ready L	Ready H	Data V	Data I
Addr I	0	$\boxed{-205}$	−20	−170	−110	−80	−15
Req L	∞	0	260	80	140	170	265
Req H	∞	−185	0	−150	−90	−60	5
Ready L	∞	$\boxed{-35}$	180	0	60	$\boxed{90}$	185
Ready H	∞	−65	150	−30	0	60	155
Data V	∞	−75	140	−40	20	0	145
Data I	∞	−185	0	$\boxed{-150}$	−90	−60	0

3.2.2 Interface Timing Synthesis

Before we can verify that an interface is correct, we must have an interface, even if it is a trivial interface in which pins of the two modules are directly connected to each other as was the case for the memory and bus example above. Many properties of the interface such as which signals are connected to which ports and how, and the calculation of any intermediate signals generated are determined by the intended functionality of the hardware. Once this functionality is determined, however, it is possible to insert elements which, by delaying the times at which various signals appear to the hardware modules, alter the relative times of events in the interface circuitry. These alterations may be required whenever the relative timings of two hardware module protocols do not agree, and will insert additional events and timing relationships into the system.

Timing Parameters for Bus			
constraint	Min	Max	type
tDataInvalid	120	–	requirement
tDataValid	–	75	requirement
ReadyDelay	30	–	requirement

Figure 3.5: Bus timing parameter variation.

For example, suppose that the bus requirements given in Figure 3.3 are replaced with those of Figure 3.5. This requires that the maximum time from `Ready low` to `Data valid` is 75ns, but as Table 3.1 shows, this time separation could be as great as 90ns. The easiest way to fix this is to maintain two `Ready` signals — one from the memory itself, and another which is the same signal delayed by 20ns. The delayed signal is then used as the bus' `Ready` input and thus from the bus' point of view, the data are guaranteed to be valid from 70ns to 130ns after the `Ready` signal is lowered.

In the case above, simply delaying the time at which a signal propagates across the interface is sufficient to assure the modules' correct interaction. However, it is sometimes the case that more sophisticated transformations are required. Suppose that we return to the bus timing requirements of Figure 3.3 and memory timing delays and guarantees of Figure 3.2 except that the memory's guarantee of a minimum of 0ns for `tDataInvalid` is instead replaced with a minimum guarantee of 120ns from the time `Ready` goes low. In this case, simply delaying the `Ready` signal will not work — the data can only be assumed to be valid for a period of 30ns, yet the period the bus requires is 40ns long. In this case we must insert hardware into the interface that will latch, or write into a register, the valid data the memory outputs, and the bus must read the data from the latch.

When we discuss timing synthesis in Chapter 9, we assume that only transformations of the first type above — determining allowable delay values — will be required of our synthesis method. When transformations of the second type are required, we assume that a separate tool evaluates a simpler interface's timing violations and provides us with a skeleton structure matching the required transformation, with some designated timing delays unspecified. Our problem then becomes, "given a system of events which includes some delay relationships which are not assigned bounds, determine delay ranges for those unassigned bounds, such that the maximum timing separations of all events, subject to system timing requirements and delays plus the new delay elements are within the timing requirements given." It should be noted, however, that the usefulness of this technique goes beyond such assumptions. For example, a synthesis tool could supply the timing synthesizer with a set of such timing skeletons and then use the delay synthesis results to guide its search for an easily synthesizable solution.

3.3 Summary

In this chapter we have defined the timing verification problem for hardware interface logic and briefly introduced the problem of interface hardware timing synthesis. As presented here the interface timing verification problem consists of determining for every pair of events a and b, with each event corresponding to a signal level transition, the maximum time separation from a to b

$$t_b - t_a, \tag{3.14}$$

when the times of the events are governed by a systems of equations of the form

$$t_a \leq \max(t_1 + \alpha_1, \ldots, t_k + \alpha_k). \tag{3.15}$$

Chapter 4
INTERFACE TIMING VERIFICATION ALGORITHMS

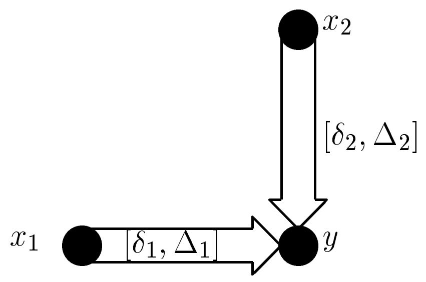
As shown in the previous chapter, timing diagrams can represent a wide variety of timing behaviors. However, while timing diagrams have been used for decades to describe the interface behavior of digital circuits, the earliest efforts to automate the generation of correct hardware interfaces did not directly employ them. Instead, hardware description languages were augmented so that behavioral descriptions of the circuits being synthesized could include a description of their interface with other hardware. Section 4.1 gives a brief overview of these hardware language extensions. Sections 4.2 and 4.3 discuss algorithms for timing verification as divorced from questions of logic synthesis or functional behavior, with Section 4.2 providing an overview of timing verification for a range of primitives more general than the requirement, guarantee, and delay as given in Section 3.2, and Section 4.3 covering algorithms for the interface timing verification problem as described in Section 3.2.1. One of these algorithms is the `ShortCircuit` algorithm [WB93b, WB94a], which is the subject of Chapter 8.

4.1 Early Research in Timing Verification

As mentioned above, the first explorations into the automation of interface timing verification grew from efforts to accommodate the descriptions of interface timing protocols in hardware description languages [PP87]. Such augmentation was necessary because the existing circuit synthesis tools could not describe the timing behavior and requirements of off-the-shelf hardware as expressed by timing diagrams. In general, the range of timing behaviors these interface description languages allowed were quite restricted. For example, at its lowest level the SLIDE language [PW81] describes interface behavior with assignments to signals, actions in response to signal values or changes in signal values, and quantified delays. These actions are embedded textually into what amounts to a series-parallel graph, a structure which is fundamentally mismatched with timing diagrams since the former cannot represent the full range of partial orderings expressible by the latter. Another example [Vis76] expresses interface timing relationships with state machines. While state machines are useful for describing series of interface actions as they occur in practice, they cannot efficiently represent protocols in which several independent series of events happen in parallel. However, the BSI/ISPS language [NT86] allowed true partial orderings of events by attaching labels to events and allowing simple timing constraints to be expressed between them. These constraints were of the form found in Equations 3.1 or 3.2. Additionally, although these early interface descriptions were written to describe hardware being synthesized to correctly interact with a target device, the temporal properties of the target device are not explicitly represented, and design decisions had to be made at the time the description was written in order to assure that the interface would be correct. Timing guarantees of the target device might be lost in the translation and therefore result in less efficient interfaces.

The work of Borriello [Bor88] took these specifications to a more sophisticated level: synthesizing interface logic to correctly interconnect two or more devices whose behavior and requirements are described with timing diagrams. Rather than specifying the interface events in a hardware description language format, he specified them with a **formalized timing diagram**. This format, which is essentially the timing diagram introduced in the previous chapter, not only allows events to be specified in any partial ordering, but allows both **requirements** and **guarantees**, as described in the previous chapter, to be expressed. Included in this work is an efficient algorithm for verifying the timing behavior of such a system of constraints. The solution is an all-pairs shortest paths algorithm that uses a priority-queue based algorithm originally developed for circuit layout compaction [LW83, BN86]. In addition, as is discussed in Chapter 9, there is also performed **timing synthesis** — the addition of delay elements to a skeletal interface circuit so that timing constraints are met. As with previous interface timing solutions, his work contains solutions to a wider variety of interface design problems including the logic synthesis component of the problem as well as issues that arise in the composition of timing diagrams to create state-based behavior.

While the timing verification algorithm presented by Borriello is only a small portion of the material in his Ph.D. thesis [Bor88], others have continued on to explore the problems of timing verification of interface hardware. Related research spans



Gahlinger's Constraint Types				
type	equations			
general	$max(x_1 + \delta_1,$	$x_2 + \delta_2)$	$\leq$	y
or **linear**	$min(x_1 + \Delta_1,$	$x_2 + \Delta_2)$	$\geq$	y
early	$min(x_1 + \delta_1,$	$x_2 + \delta_2)$	$\leq$	y
or **min**	$min(x_1 + \Delta_1,$	$x_2 + \Delta_2)$	$\geq$	y
late	$max(x_1 + \delta_1,$	$x_2 + \delta_2)$	$\leq$	y
or **max**	$max(x_1 + \Delta_1,$	$x_2 + \Delta_2)$	$\geq$	y

Figure 4.1: Gahlinger's three constraint types.

from practical approaches to deeply mathematical approaches. The practical tools include the *enVision* timing resolver [Org91] and Khordoc et.al.'s *Timing Diagram Interpreter* [KDC91, KDC$^+$93], both of which generate testing simulation data consistent with timing diagrams, the *TDS* expert system for timing synthesis [KRK88], and *Timing Designer* [Gla93], a commercial timing diagram editor with verification facilities intended to facilitate exploration of the timing design space. Mathematical approaches include Tiedemann's process calculus formulation [Tie91], temporal logic approaches such as *trio* [CPMS91], which integrates timing and logic behavior, and Cingel's technique for using an automated theorem prover [Cin93].

4.2 General Timing Verification Algorithms

The first effort to expand and rigorously define the timing properties of hardware interfaces as separate from logical functionality is due to Gahlinger [Gah90]. His Ph.D. thesis is concerned with answering two timing verification problems:

- is a given timing description consistent; and
- does the composition of two devices, each with its own consistent timing description meet all timing requirements of the combination?

In addition, the range of allowed timing constraints is extended beyond delays, guarantees, and requirements to include relationships as given in Equation 3.6.

Gahlinger's three different timing constraints are modeled in Figure 4.1. These constraint systems can be thought of as having an I-BDS style description with the following augmentation: each node has a type that is one of **general**, **early**, or **late** which modifies the way in which when each event's time is described relative to its immediate predecessors in the graph. These semantics are given in the table at the right of Figure 4.1. Note that under this description, an I-BDS system consists of events which are all of type **late**. Since Gahlinger's system requires that the underlying graph be acyclic and each graph node be of only one type, it cannot describe all systems that a D-BDS can — in a D-BDS system it is perfectly legal to produce a graph in which cycles are formed through the max constraints. Similarly, D-BDS systems cannot express the range of systems Gahlinger can, since they cannot express the **early** constraints.

As McMillan and Dill have shown [MD92], accurately determining the maximum time separations of events which are built from these constraint types is NP-complete. They provide a reduction of the problem to 3-SAT, and example of which is given in Figure 4.2 for the formula

$$(a + b + c)(a' + b' + d). \tag{4.1}$$

Square nodes indicate **max** events (**late** events in Gahlinger's taxonomy) and circular nodes indicate **min** events (**early** events in Gahlinger's taxonomy). The maximum separation between nodes q and r is 1 if and only if there exists an assignment of values to the variables such that q happens 0 time units after s, and r happens 1 time unit after s. To see this, note the following:

- All events happen within 0 to 1 time units after s;
- Node q only happens at time 0 if for all formula variables x, at least one of x or x' is 0;
- Each term t_i is set to be equal to the value of a single 3-SAT term by constraining it to be the max of the values of all literals in the term; and
- r is 1 if and only if all terms t_i happen at time 1.

Gahlinger's solution to verifying such a system is (with some exceptions) to only propagate timing information forward in the graph along the graph arcs starting at a single source node. In the case of his **early** type arcs, it must be decided when their target node is first encountered, which arc provides the lower bound. Once that decision is made, it will not change. Thus, under Gahlinger's scheme, when the

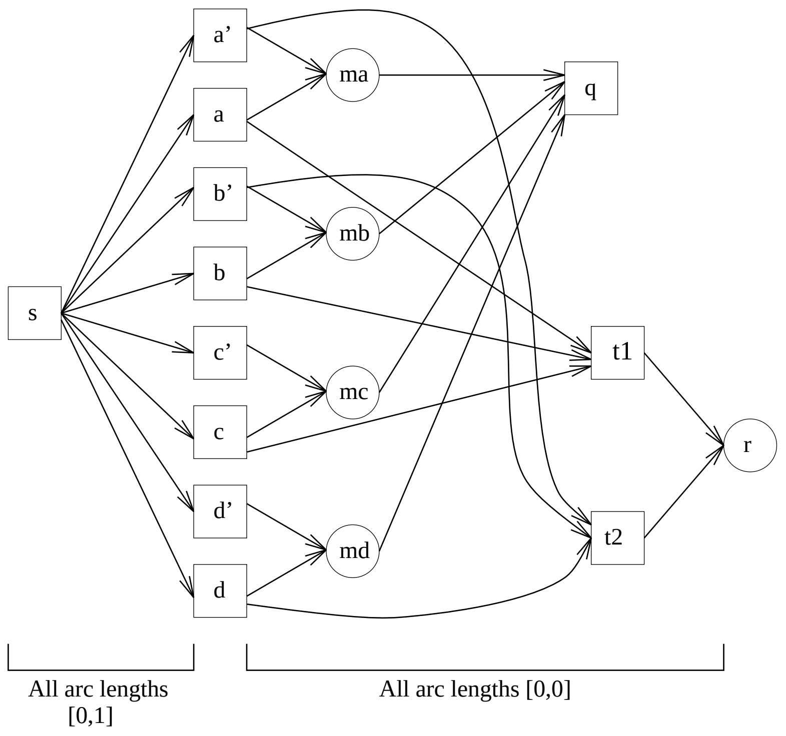


Figure 4.2: An example of McMillan and Dill’s 3-SAT transformation for the formula $(a+b+c)(a'+b'+d)$. Square nodes indicate **max** events and circular nodes indicate **min** events.

node *ma* is first reached, one of a or a' is arbitrarily picked to provide *ma*’s lower bound, and the separation between q and r can then only be 1 if the choice of a or a' for *ma*’s lower bound corresponds to the assignment of that variable to 0 in a satisfying assignment. McMillan and Dill accurately solve this problem using a branch and bound method based on an algorithm which solves systems of **max** and **linear** (**general** in Gahlinger’s taxonomy) constraints, but which does not include the **min** constraints. The combined **max** and **linear** algorithm is discussed in Section 4.3.1.

More recently, Yen et. al. [YICW94] have also developed an algorithm which solves the problem using a branch and bound method built upon a solution technique for the problem without **min** constraints. As with Gahlinger’s approach, their input is a directed, acyclic graph in which each arc has associated with it a lower and upper time bound. As with McMillan and Dill’s technique, instead of assigning a type to each node, each arc is of type **linear**, **max**, or **min**. Section 4.3.3 discusses this algorithm as it applies to **linear** and **max** constraints only.

In the context of interface timing verification and synthesis with an emphasis on asynchronous behavior Vanbekbergen [Van93] considers two problems somewhat less complicated than the combination of **linear**, **max**, and **min** constraints, which Vanbekbergen refers to as types **I**, **II**, and **III**, respectively. The first, type II only, is precisely the I-BDS verification problem. Although an algorithm is given in [Van93], it was later shown to be incorrect, with a correct algorithm given by McMillan and Dill [MD92] and reproduced here in Appendix A.1. The need for an algorithm to solve combination type I and II systems, which are expressible as D-BDS systems is also mentioned, although Vanbekbergen found no satisfactory method for this problem.

4.3 Verification Algorithms for D-BDS Systems

In this section we discuss algorithms for determining maximum time separations in D-BDS systems. All of these algorithms were developed for use in interface timing verification. McMillan and Dill give the first such algorithm [MD92], which they refer to as the generalized max-only constraint problem, and which includes systems which may also be referred to as mixed type I & type II systems (Vanbekbergen) or mixed general & late systems (Gahlinger). As we will see in Chapter 6, this problem is expressible in the max-plus algebra, but for the moment, we concentrate on the basic algorithmic structure of the problem as it relates to interface timing.

4.3.1 McMillan and Dill's Algorithm

In this section we discuss McMillan and Dill's solution to the timing verification problem for D-BDS systems, which they refer to as the "generalized max-only constraint problem," and which we will sometimes refer to as the $\mathcal{MD}$ algorithm.

Given constraints relating n events $\{x_0, \ldots, x_{n-1}\}$, the $\mathcal{MD}$ algorithm calculates the maximum value of all n^2 node separations $x_j - x_i$. We present in Figure 4.3 a simplified version of the algorithm in which only the n maximum separations relative to the single event x_0 are calculated. In order to determine all n^2 separations the algorithm of Figure 4.3 must be run n times with each x_i taking on the role of x_0. Some efficiency is lost in this variation, and thus the original is provided in Appendix A.2.

McMillan & Dill's algorithm in UBC terms
Inputs: a system of m UBCs over n variables x_0 through x_{n-1}
Outputs: maximum possible value for each x_i when $x_0 = 0$

```
$x_0 \leftarrow 0$
**Forall** $x_i$
    **if** $i \neq 0$ **then** $x_i \leftarrow \infty$
**Do:**
    **For** each UBC $u_i : x_{\tau(i)} \leq \max_j(x_j + \alpha_{i,j}, \ldots)$ in sequence **do:**
        **If** $x_{\tau(i)} > \max_j(x_j + \alpha_{i,j}, \ldots)$ **then**
            $x_{\tau(i)} \leftarrow \max_j(x_j + \alpha_{i,j}, \ldots)$
    **If** $x_0 < 0$ **then**
        Report constraints inconsistent and exit algorithm
**Until** no $x_i$ changes
Report values of all $x_i$.
```

Figure 4.3: McMillan and Dill's algorithm modified to determine maximum time separations relative to a single event.

However, the forthcoming discussion of the $\mathcal{MD}$ algorithm as presented here applies to the original algorithm as well. In both cases, we phrase the algorithm in terms of UBCs rather than using the two constraint forms McMillan and Dill employ, but translation between the two is simple.

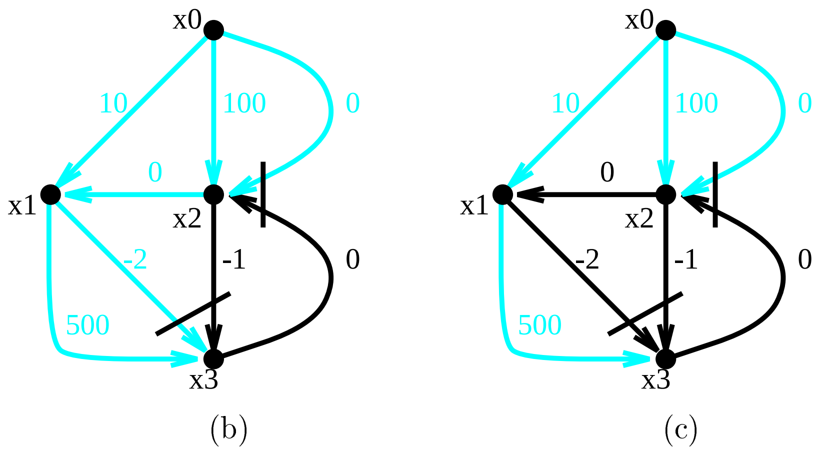
The table at the bottom of Figure 4.4 shows how the algorithm of Figure 4.3 determines the maximum time separations relative to event x_0 for the constraints pictured at the top of the figure. We begin by using the constraints $x_1 \leq x_0 + 10$ and $x_2 \leq x_0 + 100$ to change the upper bounds on x_1 and x_2 from ∞ to 10 and 100, respectively. At first, no other constraint may be used to decrease x_1's upper bound, since x_2 is greater than $x_0 + 10$. However, the two constraints

$$x_2 \leq \max(x_0, x_3) \quad \text{and} \tag{4.2}$$

$$x_3 \leq \max(x_1 - 2, x_2 - 1) \tag{4.3}$$

are then applied repeatedly one after the other, decreasing the bounds on x_2 and x_3 by 1 on each trip through the `Do`-loop in Figure 4.3, until the beginning of the 93rd

$$\begin{aligned} x_1 &\leq x_0 + 10 \\ x_1 &\leq x_2 \\ x_2 &\leq x_0 + 100 \\ x_2 &\leq \max(x_3, x_0) \\ x_3 &\leq \max(x_1 - 2, x_2 - 1) \\ x_3 &\leq x_1 + 500 \end{aligned}$$



variable	start	pass 1	pass 2	pass $0 < k \leq 92$	pass $93 \leq k \leq 101$	pass 102
x_0	0	0	0	0	0	0
x_1	∞	10	10	10	$102 - k$	0
x_2	∞	100	99	$101 - k$	$101 - k$	0
x_3	∞	99	98	$100 - k$	$100 - k$	-1

Figure 4.4: Progress of McMillan and Dill's algorithm on a set of example constraints. Cycles described by the darker arcs in each graph correspond to the maximum terms occurring in repeatedly applied constraints during passes 2-92 (graph b) and 93-101 (graph c) through the `Do`-loop.

iteration of the loop, at which point $x_2 = 9$, and the constraint $x_1 \leq x_2$ applies.

Once the constraint $x_1 \leq x_2$ applies, we again see a pattern of repeated constraint application, this time using the three constraints

$$x_1 \leq x_2 \tag{4.4}$$

$$x_2 \leq \max(x_0, x_3) \text{ and} \tag{4.5}$$

$$x_3 \leq \max(x_1 - 2, x_2 - 1) \tag{4.6}$$

for passes 93 through 102 through the do loop. After the 102nd pass, the algorithm terminates since no constraint may be applied to reduce any of the variables' bounds.

The reader may notice that the algorithm of Figure 4.3 is performing a calculation very much like the single source shortest paths from x_0. The difference, of course, is that whereas the input to the shortest paths problem consists of single edge lengths which can be expressed with inequalities of the form

$$x_i \leq x_j + \alpha_{i,j}, \tag{4.7}$$

the maximum operation of upper bound constraints may bind together several graph edges into a single constraint. Thus the existence of negative weight cycles (the darker arcs in Figure 4.4(b) and (c)) among the constraints does not necessarily indicate that the system of constraints is inconsistent. Instead, it indicates that among the constraints inducing arcs along a negative weight cycle, at least one constraint must have its maximum term correspond to an arc not on that cycle. The re-application of constraints continues until the maximum term of one of the active constraints changes — in this case when the term $x_1 - 2$ becomes larger than $x_2 - 1$ in Equation 4.3.

Note that the number of passes required for convergence depends on the coefficient in the constraint

$$x_2 \leq x_0 + 100. \tag{4.8}$$

If the constraint offset were changed from 100 to 1000, the process would take nearly ten times as long, since the value of x_2 would then have to decline from a value of 1000 to 9 before the constraint of Equation 4.4 would apply to x_1.

This is the first of two key problems with McMillan and Dill's approach – the time to calculate all n^2 maximum separations is

$$O(n^3 \cdot \sum_i |\delta_i|), \tag{4.9}$$

where n is the number of variables and the δ_i's are the values of the constant terms in the constraints. This is intimately dependent on those constraint values, and undesirable since many instances of practical timing verification problems include a wide range of values in their scalar terms.

$$\begin{aligned} x_1 &\leq x_2 \\ x_1 &\leq x_0 + 10 \\ x_2 &\leq \max(x_3, x_0) \\ x_3 &\leq \max(x_1 - 2, x_2 - 1) \end{aligned}$$

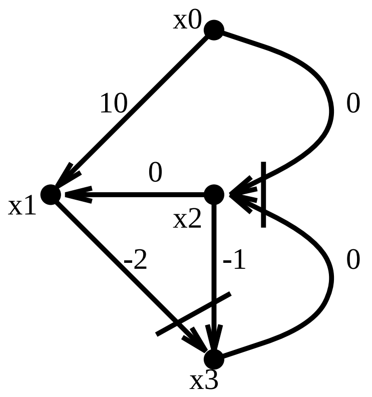


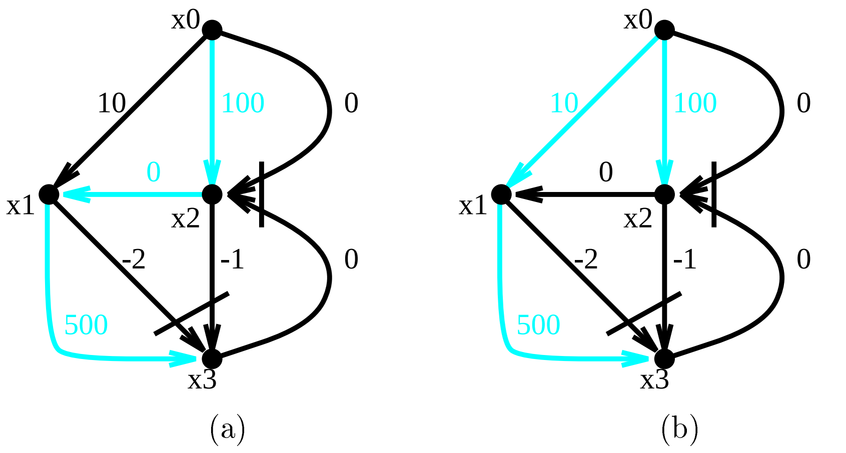
Figure 4.5: Constraint set for which McMillan and Dill's algorithm fails.

The second problem is illustrated by the constraint set in Figure 4.5. The constraints $x_1 \leq x_0 + 10$, $x_2 \leq x_0 + 100$, and $x_3 \leq x_2 + 500$ in Figure 4.4 are not tight and are therefore their removal from the constraint set should not affect the final values of the maximum separations. However, when the constraints $x_3 \leq x_1 + 500$ and $x_2 \leq x_0 + 100$ are removed from the set, as pictured in Figure 4.5, the $\mathcal{MD}$ algorithm cannot find tighter bounds than ∞ for x_2 and x_3 relative to x_0. To see how this happens, note that when x_2, and x_3 have upper bound ∞, the equation $x_3 \leq \max(x_1 - 2, x_2 - 1)$ does not change x_3's value since $\infty - 1 = \infty$.

4.3.2 The Short-Circuit Algorithm

One of the main contributions of this work is the `ShortCircuit` algorithm [WB93b, WB94a, WB94b], which is the subject of Chapter 8. This algorithm solves the two problems given above for McMillan and Dill's algorithm. The techniques for achieving this rely on two innovations which we discuss here for the example given in Figure 4.4.

Recall that during passes 1-92 of McMillan and Dill's algorithm, the same constraints (whose maximum terms correspond to the dark arcs in Figure 4.4(b)) are re-applied, causing the values along the cycle $x_2 \rightarrow x_3 \rightarrow x_2$ to decrease with each pass. As we will show in Chapter 8, whenever there is a negative weight directed cycle in the graph induced by the constraints most recently used to update each nodes' bound, the values of all nodes on such cycles will continue to decrease until some



variable	start	$\mathcal{MD}$ rounds				short circuit	$\mathcal{MD}$ rounds				short-c	regular
x_0	0	0	0	0	0	0	0	0	0	0	0	0
x_1	∞	10	10	10	10	10	9	8	7	6	1	0
x_2	∞	100	99	98	97	9	8	7	6	5	0	0
x_3	∞	99	98	97	96	8	7	6	5	4	-1	-1

Figure 4.6: Progress of the `ShortCircuit` algorithm on the constraints of Figure 4.4. Dark arcs in diagram (a) correspond to constraints active after the first four $\mathcal{MD}$ rounds, those in (b) correspond to active constraints for the second set of $\mathcal{MD}$ rounds.

node on the cycle has its value set equal to the maximum term corresponding to an arc exterior to that cycle.

Thus, after four passes through the `Do`-loop of the $\mathcal{MD}$ algorithm, we may deduce that the constraints of Equations 4.2 and 4.3 may be reapplied until either $x_2 = 0$ (the current value of x_0) or $x_3 = 8$ (the current value of $x_1 - 2$).

Suppose that all values of nodes on the negative cycle can be shown to decrease together at an even rate. Since the current values of x_2 and x_3 are 97 and 96, respectively, the cycle will be "broken" when $x_3 = 8$ since x_2 must decrease by 97 to break the cycle, while x_3 need only decrease by 88. Chapter 8 shows that while applying constraints directly might not result in such an even decrease in node bounds, such a decrease is consistent with the constraints. Therefore, we may short-circuit the process by setting x_3 to 8 and x_2 to 9.

In general, the `ShortCircuit` algorithm proceeds as follows:

- Perform n rounds of the $\mathcal{MD}$ algorithm, keeping track of which constraint was last applied to reduce each node's bound,

- Short-circuit each negative weight directed cycle in the graph induced by those most recently used constraints:

- repeat the process of performing n rounds of the $\mathcal{MD}$ algorithm and then short-circuiting until either

  - the node values converge or
  - a directed negative weight cycle is found that has no exterior arcs, indicating that the constraints can be reapplied until the node bounds fall below any finite value and thus the set is unsatisfiable.

The short-circuiting process itself consists of the following steps:

- on a given negative weight cycle in the graph induced by the most recently used constraints, find all nodes to which there is one or more directed arc from a node not on that cycle

- determine what each node's value would be if the maximum among only those terms corresponding to such "exterior" arcs were used to update its bound

- find the node with the smallest difference between its current bound and the bound calculated from the "exterior" arcs only

- subtract that smallest difference from all nodes on the cycle.

Because several negative cycles may overlap, the short-circuiting portion of the algorithm is actually applied not to simple cycles, but instead to each strongly connected component (SCC), with node value differences calculated only for those nodes to which there are arcs directed from outside the SCC. An example of this is given in Figure 4.6(b), in which there are two overlapping negative weight cycles. The first corresponds to the cycle in Figure 4.4(a), and indicates that for the constraint set to be consistent, either

$$x_3 > x_2 - 1 \quad \text{or} \tag{4.10}$$

$$x_2 > x_3, \tag{4.11}$$

since otherwise all arcs on that cycle correspond to the maximum terms in their corresponding constraints. The second is the cycle $x_1 \rightarrow x_3 \rightarrow x_2$ and similarly indicates that for the constraint set to be consistent, either

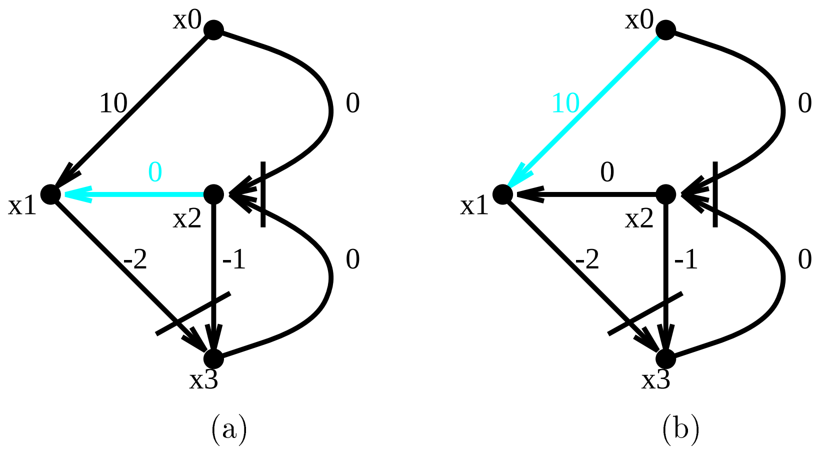
$$x_3 > x_1 - 2 \quad \text{or} \tag{4.12}$$

$$x_2 > x_3. \tag{4.13}$$

Now, we know that Equations 4.4 through 4.6 must hold for any solution to the whole system. Equation 4.6 tells us that Equations 4.10 and 4.12 cannot both be satisfied, and thus one of Equations 4.11 and 4.13 must hold. Since these two equations are identical, we know that both must hold and therefore since Equation 4.5 must hold, and we may therefore deduce that $x_2 \leq 0$, thus breaking both cycles.

As we will see in Chapter 8, the worst case run time of `ShortCircuit` can be expressed independently of the values of the constant constraint offset terms. The method above solves the first deficiency of McMillan and Dill's algorithm — run time intimately dependent on constraint values — but another addition is required to solve the problem exhibited in Figure 4.5. Recall that the problem arose because we cannot compare values such as $\infty - 1$ and ∞, but initial upper bounds on all node values are ∞. We solve the problem by noting that any single consistent assignment of node values has a largest node value. We use the value $\mathcal{V}$ to represent that value, and then use $\mathcal{V}$ as an upper bound on all node values in that particular solution. If we can obtain upper bounds independent of $\mathcal{V}$ for all events in the system, then we know that that bound applies to any instance of the events.

Figure 4.7 shows how the algorithm would progress for the constraints in Figure 4.5. In the first short circuiting round, the values $x_2 = \mathcal{V} - 3$ and $x_3 = \mathcal{V} - 4$ are compared with possible cycle breaking values 0 and 8, respectively. Although we do not know the value of $\mathcal{V}$, we do know that $(\mathcal{V} - 4) - 8 = \mathcal{V} - 12$ is smaller than $(\mathcal{V} - 3) - 0 = \mathcal{V} - 3$, and thus subtract $\mathcal{V} - 12$ from both x_2 and x_3 at the end of the first short circuiting round. This removes all $\mathcal{V}$-terms from the problem, and the algorithm then proceeds as before.



variable	start	$\mathcal{MD}$ rounds				short-c	$\mathcal{MD}$ rounds				short-c	$\mathcal{MD}$
x_0	0	0	0	0	0	0	0	0	0	0	0	0
x_1	$\mathcal{V}$	10	10	10	10	10	9	8	7	6	1	0
x_2	$\mathcal{V}$	$\mathcal{V}$	$\mathcal{V}-1$	$\mathcal{V}-2$	$\mathcal{V}-3$	9	8	7	6	5	0	0
x_3	$\mathcal{V}$	$\mathcal{V}-1$	$\mathcal{V}-2$	$\mathcal{V}-3$	$\mathcal{V}-4$	8	7	6	5	4	-1	-1

Figure 4.7: Progress of `ShortCircuit` on the constraints of Figure 4.5.

4.3.3 Yen et. al.'s Algorithm

Since the `ShortCircuit` algorithm was first presented [WB93b], Yen et. al. have proposed an interface timing verification algorithm, `MaxSeparation` [YICW94], for combined **linear** and **max** timing constraints. The algorithm appears promising, but no proof of correctness has yet been published.

As mentioned above, the input is a directed, acyclic graph in which each arc has associated with it a both lower and upper time bounds, as well as a constraint type of **linear** or **max**. Like the `ShortCircuit` algorithm, this algorithm finds maximum time separations for all events relative to a single fixed event, and thus must be performed n times to obtain the full set of n^2 maximum time separations.

The `MaxSeparation` algorithm differs from the $\mathcal{MD}$ and `ShortCircuit` algorithms in that it bounds the maximum separations from below and then increases them until it converges upon a maximum. The method begins by first determining the minimum solution solving all linear constraints and all lower bounds of the maximum constraints. It then engages in a iterative process in which the **maximum slack** — the maximum amount by which each variable may increase without violating any currently satisfied constraints — is found for each variable and then added to the variables current value. As with the `ShortCircuit` method, the process is an iterative one, calculating new maximum slacks based on the updated lower bounds on the variable's maximum separations.

As did McMillan and Dill, Yen et. al. also give results for the min-max-linear time problem as solved using branch and bound with their max-linear algorithm as a subroutine.

Part III

Linear Max Plus Systems

Chapter 5
DIOIDS AND DIOID CLOSURE: THE MAX-PLUS ALGEBRA AND LONGEST PATHS

In this chapter, we introduce the max-plus algebra, which formalizes the synchronization and delay primitives introduced in Chapter 1. The material covered in this chapter is a review of the previous results of other researchers, upon which a technique for the solution and optimization of linear max-plus systems will be built. The max-plus algebra was first used to describe longest path problems [GM77] in Operations Research, and has more recently been of interest to those studying synchronizing systems [CMQV89, BCOQ92].

5.1 Dioids

The max-plus algebra is one of many algebraic structures which are called **dioids**. This is a structure which does not include an inverse for its "addition" operation, but otherwise has what is essentially the structure of a ring. For our purposes here, the important feature of dioids is that the "plus" operation is **idempotent**. That is, a plus a always equals a. We begin by introducing the **monoid**, the simple algebraic unit from which a dioid is built.

Definition 5.1 *A* **monoid**, $(\mathcal{M}, \oplus)$ *is an algebraic structure consisting of a set of elements,* $\mathcal{M}$ *and an operation* $\oplus$ *on the elements of* $\mathcal{M}$ *such that:*

- $\mathcal{M}$ *is* **closed** *with respect to* $\oplus$,

$$[a, b \in \mathcal{M} \wedge a \oplus b = c] \quad \Rightarrow \quad c \in \mathcal{M}. \tag{5.1}$$

- $\mathcal{M}$ *is* **associative** *with respect to* $\oplus$,

$$\forall a, b, c \in \mathcal{M} : \quad (a \oplus b) \oplus c = a \oplus (b \oplus c). \tag{5.2}$$

- *$\mathcal{M}$ has a* zero *or* **identity element**, *$\epsilon \in \mathcal{M}$ such that*

$$\forall a \in \mathcal{M} : \quad \epsilon \oplus a = a \oplus \epsilon = a \tag{5.3}$$

A **commutative monoid**, *$(\mathcal{M}, \oplus)$ is a monoid in which*

$$\forall a, b \in \mathcal{M} : \quad a \oplus b = b \oplus a. \tag{5.4}$$

Definition 5.2 *A* dioid, *$(\mathcal{D}, \oplus, \otimes)$ is an algebraic structure consisting of a set $\mathcal{D}$ with a pair of associated operations $\oplus$ and $\otimes$ such that*

- *$(\mathcal{D}, \oplus)$ is a* **commutative monoid** *with identity element ϵ.*

- *$(\mathcal{D}, \otimes)$ is a* **monoid** *with identity element e.*

- *ϵ (the identity element for $\oplus$) is* **absorbing** *for $\otimes$*

$$\forall a \in \mathcal{D} : \quad a \otimes \epsilon = \epsilon \otimes a = \epsilon. \tag{5.5}$$

- *$\otimes$ is both* **left** *and* **right distributive** *over $\oplus$*

$$\forall a, b, c \in \mathcal{D} : \quad c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b) \tag{5.6}$$

$$\forall a, b, c \in \mathcal{D} : \quad (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c) \tag{5.7}$$

- *$\oplus$ is* **idempotent**.

$$\forall a \in \mathcal{D} : \quad a \oplus a = a. \tag{5.8}$$

When the $\otimes$ operation is commutative, we say that the dioid $(\mathcal{D}, \oplus, \otimes)$ is a **commutative dioid**. *When a dioid is closed for infinite series of $\oplus$ operations and the left and right distributive properties apply to these infinite series, we say that it is a* **complete dioid**.

We will be using this definition of a **dioid**, which is due to Baccelli et. al. [BCOQ92] and which is a somewhat more restrictive definition than that of Gondran and Minoux [GM84]. Rather than require idempotency (Equation 5.8), Gondran and Minoux's definition requires the following:

$$a = b \oplus c \text{ and } b = a \oplus d \quad \Rightarrow \quad a = b. \tag{5.9}$$

In either case, $\oplus$ induces an ordering, $\geq$ on the elements of $\mathcal{D}$ as follows:

$$a \geq b \quad \Leftrightarrow \quad a = a \oplus b, \tag{5.10}$$

for the idempotent case, and

$$a \geq b \quad \Leftrightarrow \quad \exists c \in \mathcal{D} : a = b \oplus c, \tag{5.11}$$

for the less stringent requirement. In either case, the relation $\geq$ must be a partial ordering (i.e. $a \geq b$ and $b \geq a \Rightarrow a = b$, and $\geq$ is transitive), and we will use $a \geq b$ and $b \leq a$ interchangably. One can see that the more general definition only requires that $a \oplus a \geq a$ instead of $a \oplus a = a$. This weaker definition classifies $(\mathcal{R}^{+}, +, \times)$ as a dioid.

5.1.1 Example – The Max-Plus Dioid

We now introduce a dioid which can express the D-BDS systems of Chapter 2, and therefore the timing verification problem of Chapter 4 — the **max-plus algebra**.

Definition 5.3 *The* **max-plus** *or* $\{\max, +\}$ **algebra** *is the dioid* $(\mathcal{R} \cup \epsilon, \max, +)$ *with identity elements* $\epsilon = -\infty$ *for "*max*" and* $e = 0$ *for "+".*

As we will discover in the next chapter, the max-plus algebra has several useful properties which are not necessarily present in all dioids. Since our solution technique for the max-plus algebra will apply to all algebras of that more restricted class, we will use the $\oplus$ and $\otimes$ operators and ϵ rather than "max", "+", and $-\infty$, respectively. This will then keep the form of our max-plus equations similar to that of the general case. However, we may sometimes use 0 as well as e as the identity for $\otimes$. As is the case for the familiar multiplication, juxtaposition of terms a and b implies the presence of

UBC Equations					
wake	$\leq$	$-35 \otimes$ phone	phone	$\leq$	$45 \otimes$ wake
phone	$\leq$	$-10 \otimes$ leave	leave	$\leq$	$15 \otimes$ phone
leave	$\leq$	$-15 \otimes$ arrive	arrive	$\leq$	$20 \otimes$ leave
phone	$\leq$	$-50 \otimes$ ready	ready	$\leq$	$60 \otimes$ phone
arrive	$\leq$	meet	meet	$\leq$	arrive $\oplus$ ready
ready	$\leq$	meet			
meet	$\leq$	$-30 \otimes$ work	work	$\leq$	$40 \otimes$ meet
wake	$\leq$	$-50 \otimes$ leave	leave	$\leq$	$55 \otimes$ wake

Figure 5.1: Equations of Figure 2.7 in Max-Plus form.

the $\otimes$ operator. Hence, $ab = a \otimes b$. In equations with ambiguous parenthesization, the $\otimes$ operation should be considered to have priority over $\oplus$.

One of these useful additional properties of the max-plus algebra is that the $\otimes$ operation, "+", is invertible for all $a \neq \epsilon$. We may often find it easier to represent the $\otimes$ inverse of a value such as 35 as -35 rather than 35^{-1}. This makes the translation between the max-plus and more usual form of our equations a bit more obvious, and should not cause too much confusion as there is no $\oplus$ inverse here.

We now re-phrase the equations of Figure 2.7 in max-plus form. They are pictured in Figure 5.1. Note that while we have written the UBC

$$\text{wake} \leq \text{phone} - 35 \tag{5.12}$$

as

$$\text{wake} \leq -35 \otimes \text{phone}, \tag{5.13}$$

the dioid ordering relation of Equation 5.10 allows us to express such inequalities as equalities, in this case as

$$-35 \otimes \text{phone} = (-35 \otimes \text{phone}) \oplus \text{wake}. \tag{5.14}$$

5.2 Linear Expressions and Equations over Dioids

The expressions on the right hand side of each UBC in Figure 5.1 are **linear max-plus expressions**, a specific example of **linear dioid expressions.** Linear dioid expressions and linear dioid equations have forms analogous to those of normal linear algebraic expressions and equations. As with regular linear systems, the existence of a set of n variables, $\mathcal{X} = \{x_0, x_1, \ldots, x_{n-1}\}$ is assumed.

Definition 5.4 *A* **linear expression over dioid** $\mathcal{D}$ *is the finite* $\oplus$ *of terms* b_i *and* $a_i \otimes x_i$ *where the* a_i*'s and* b_i*'s are elements of* $\mathcal{D}$ *and the terms* x_i *are drawn from* $\mathcal{X}$*.*

Proposition 5.1 *Any linear expression over dioid* $\mathcal{D}$ *can be written as*

$$a_0x_0 \oplus a_1x_1 \oplus \ldots \oplus a_{n-1}x_{n-1} \oplus b, \tag{5.15}$$

or equivalently as

$$\left[\bigoplus_{i=0}^{i=n-1} a_i \otimes x_i\right] \oplus b \tag{5.16}$$

where b *and all* a_i *are members of* $\mathcal{D}$*.*

Proof: Note that since a_i and b may equal ϵ, any linear dioid expression which does not contain a term b or a term $a_i \otimes x_i$ for a given i still may be expressed as in Equation 5.15, by setting the value of b or a_i to ϵ and applying the absorbing property as given in Equation 5.5. Additionally, any expression containing a finite number, k, of terms a_jx_i for a given i may be written

$$y \oplus a_1x_i \oplus a_2x_i \oplus \cdots \oplus a_kx_i, \tag{5.17}$$

where y includes all terms not including x_i, since $\oplus$ is commutative. We then apply the distributive rule (Equation 5.6), resulting in

$$y \oplus (a_1 \oplus a_2 \oplus \cdots \oplus a_k)x, \tag{5.18}$$

and then consolidate the a_j's into a single term, a', resulting in

$$y \oplus a'x_i. \tag{5.19}$$

The same applies for multiple terms b_i as a special case when $x = e$. $\square$

Definition 5.5 *A* **linear equation over dioid** $\mathcal{D}$ *is the equation of two linear dioid expressions, and thus has form*

$$\left[\bigoplus_{i=0}^{i=n-1} a_i \otimes x_i\right] \oplus b = \left[\bigoplus_{i=0}^{i=n-1} c_i \otimes x_i\right] \oplus d, \tag{5.20}$$

where b*,* d*, and all* a_i *and* c_i *are members of* $\mathcal{D}$*.*

Unlike a linear expression in the more usual algebra, a linear dioid equation must allow for a full complement of terms on both sides of the equality. This is a direct result of the fact that in this algebra there are no inverses to the $\oplus$ operation.

5.2.1 Canonical Form for Linear Max-Plus Expressions and Equations

Although Proposition 5.1 shows that the form given in Equation 5.15 is the canonical form for linear dioid expressions, the linear dioid equation as given in Equation 5.20 is not a canonical form. To see this we need only note that for the max-plus algebra the two equations

$$\begin{aligned} x &= (-2 \otimes x) \oplus 4 \text{ and} && (5.21)\\ x &= (-3 \otimes x) \oplus 4 && (5.22) \end{aligned}$$

are equivalent, and are both solved only when $x = 4$. We begin by noting that in both equations, it must be the case that $x \geq 4$ since the right hand side of both equations is at least as large as 4. Furthermore, for the first equation, we know that since $x \neq \epsilon$, x is strictly greater than $x - 2$, and thus the term $-2 \otimes x$ may safely be dropped from the right hand side of the equation. The same argument applies to the second equation. This process of dropping unnecessary terms is the basis for the following definition of a canonical linear max-plus equation, due to Baccelli et. al. [BCOQ92].

Definition 5.6 *A linear max-plus equation in* **canonical form** *is a linear max-plus equation of the form*

$$\left[\bigoplus_{i=0}^{i=n-1} a_i \otimes x_i\right] \oplus b = \left[\bigoplus_{i=0}^{i=n-1} c_i \otimes x_i\right] \oplus d \tag{5.23}$$

where if $a_i \neq c_i$ *then either* $a_i = \epsilon$ *or* $c_i = \epsilon$*, and similarly if* $b \neq d$ *then either* $b = \epsilon$ *or* $d = \epsilon$*.*

Proposition 5.2 (Baccelli et al [BCOQ92], Definition 3.15) *Any linear max-plus equation may be written in the canonical form of Definition 5.6.*

It is important to note that this definition of a canonical linear max-plus equation does not apply to all dioids. For example in the dioid whose elements are pairs $[a, b]$ with $a, b \in \{\mathcal{R} \cup \epsilon\}$ with the $\oplus$ and $\otimes$ operation being the point-wise max and addition functions, we find that the set of solutions to the equation

$$[0, 1]x = [1, 0]x \tag{5.24}$$

is true precisely for those x of the form $[a, a]$ while the equations

$$\begin{aligned} [0,1]\, x &= [\epsilon, \epsilon] \text{ and} & (5.25) \\ [1,0]\, x &= [\epsilon, \epsilon] & (5.26) \end{aligned}$$

are not. The canonical form of Equation 5.23 is possible due to the fact that the elements of the max-plus dioid are totally ordered.

5.3 Dioid Matrices

The systems we wish to study are generally expressed with a group of several equations over a single set of variables. A group of linear dioid equations over the same variables is called a **linear dioid system**. One convenient way to represent such a system is with matrices whose entries are members of the dioid.

Definition 5.7 *The* **additive operation,** $\oplus$ *for* $m \times n$ *matrices* A *and* B *with entries from* $\mathcal{D}$ *is* $A \oplus B = C$ *where*

$$\forall i, j : \;\; [0 \le i < m \text{ and } 0 \le j < n] \;\; \Rightarrow c_{i,j} = a_{i,j} \oplus b_{i,j}. \tag{5.27}$$

Definition 5.8 *The* **multiplicative operation,** $\otimes$ *for* $m \times n$ *matrix* A *and* $n \times p$ *matrix* B *with entries from* $\mathcal{D}$ *is* $A \otimes B = C$ *where*

$$\forall i, j : \;\; [0 \le i < m \text{ and } 0 \le j < p] \;\; \rightarrow c_{i,j} = \bigoplus_{k=0}^{n-1} a_{i,k} \otimes b_{k,j} \tag{5.28}$$

Whenever $(\mathcal{D}, \oplus, \otimes)$ is a dioid, $\mathcal{D}^{n\times n}$, the set of $n \times n$ matrices with entries in $\mathcal{D}$ is a dioid as well [CMQV89]. The addition and multiplication definitions are included in the definitions above, but we must define the $\oplus$- and $\otimes$-identities for $\mathcal{D}^{n\times n}$.

Definition 5.9 *The* **additive identity** *for matrices of size $n \times n$ with entries from $\mathcal{D}$ is the $n \times n$ matrix with every entry equal to ϵ.*

Definition 5.10 *The* **multiplicative identity** *for matrices of size $n \times n$ is the $n \times n$ matrix E in which $E_{i,i} = e$ for all $0 \leq i < n$ and all other $E_{j,k} = \epsilon$. For a specific n, this matrix may sometimes be referred to here as E_n.*

Note however, that as with our more familiar matrix algebra, the matrix dioid is not in general commutative. A simple example is given below.

Example 5.1 *Below are given two 2×2 matrices whose max-plus products demonstrate that max-plus multiplication is not commutative.*

$$\begin{bmatrix} 0 & \epsilon \\ 1 & \epsilon \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ \epsilon & \epsilon \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix} \tag{5.29}$$

$$\begin{bmatrix} 0 & 1 \\ \epsilon & \epsilon \end{bmatrix} \otimes \begin{bmatrix} 0 & \epsilon \\ 1 & \epsilon \end{bmatrix} = \begin{bmatrix} 2 & \epsilon \\ \epsilon & \epsilon \end{bmatrix}. \tag{5.30}$$

Based on these definitions, we can now completely define a linear dioid system.

Definition 5.11 *A* **linear dioid system** *of size $m \times n$ is a set of m equations, such that the i-th equation is a linear dioid equation (Equation 5.20) of form*

$$\left[\bigoplus_{j=0}^{n-1} a_{i,j} \otimes x_j\right] \oplus b_i = \left[\bigoplus_{j=0}^{n-1} c_{i,j} \otimes x_j\right] \oplus d_i. \tag{5.31}$$

These systems can also be expressed as the equation

$$A \otimes x \oplus b = C \otimes x \oplus d, \tag{5.32}$$

where A and C are $m \times n$ matrices over $\mathcal{D}$, b and d are $m \times 1$ matrices over $\mathcal{D}$, and x is the $n \times 1$ column vector of variables $[x_0, \ldots, x_i, \ldots, x_{n-1}]^T$.

5.4 Dioid Closure

One natural endeavor is to categorize the types of dioid equations which can be easily solved. The notion of **dioid closure** [BCOQ92], also called the **quasi-inverse** [GM84], for single variable equations and for matrix equations, which are both defined below, arises from this effort. As we will see below, max-plus matrix closure corresponds to determining the length of the longest paths in a directed graph. This graph-based model of max-plus closure will form the basis for the solution technique given in Chapter 6.

5.4.1 Closure of a Single Variable

Consider the possible solutions for the variable x in the equation

$$x = (a \otimes x) \oplus e, \tag{5.33}$$

where e is the identity element for $\otimes$. By substituting the right hand side of this equation into itself, we see that

$$x = (a \otimes [(a \otimes x) \oplus e]) \oplus e = a^2 x \oplus a \oplus e, \tag{5.34}$$

where a^2 is defined as immediately below.

Definition 5.12 *For $a \in \mathcal{D}$, a^k is the $\otimes$ of k a terms, the natural extension of multiplication to powering.*

If we repeatedly substitute the right hand side of this equation into itself, and if the dioid is complete, we arrive at the following formula for x:

$$x = e \oplus a \oplus a^2 \oplus a^3 \oplus \cdots. \tag{5.35}$$

It should be clear that for $e \geq a$, $a^i \geq a^{i+1}$, and therefore $x = e$.

Since the max-plus dioid as we have defined it is not complete, we cannot freely use such an infinite object. One can easily make the max-plus dioid complete by adding to it the element ∞, which is absorbing for $\oplus$. In the next Chapter we will not use this augmented max-plus dioid, but will instead only make use of closure when it can be described finitely, as is done in the proof of Theorem 6.1.

Definition 5.13 *For $a \in \mathcal{D}$, the* **closure** *of a denoted a^*, is*

$$e \oplus a \oplus a^2 \oplus a^3 \oplus \cdots = \bigoplus_{k \geq 0} a^k, \tag{5.36}$$

which is the minimum solution to the equation $x = ax \oplus e$.

An example of an equation with more than one solution is

$$x = x \oplus e, \tag{5.37}$$

which is true for all $x \geq e$, but must have minimum solution $x = e$. On the other hand, in the max-plus dioid the equation

$$x = ax \oplus e \tag{5.38}$$

has no minimum solution when $a \geq e$ but $a \neq e$ and so we describe the minimum solution set as ∞. It is important to note that "minimum" may deviate from its common meaning when the operation $\oplus$ changes — for example, in the dioid $(\mathcal{R} \cup \infty, \min, +)$, the "minimum" element is actually ∞. To see this, note that when $\oplus = \min$ the relation "$a \geq b$" of Equation 5.10 would actually indicate that a is **smaller** than b in the familiar sense.

We may also use the closure technique above to solve the more sophisticated equation

$$x = (a \otimes x) \oplus b. \tag{5.39}$$

Instead of Equation 5.34 the expansion of Equation 5.39 becomes

$$x = (a \otimes [(a \otimes x) \oplus b] \oplus b = a^2 x \oplus ab \oplus b. \tag{5.40}$$

This generalizes to

$$x = b \oplus ab \oplus a^2 b \oplus a^3 b \oplus \cdots \tag{5.41}$$

and leads to a final solution of

$$x = a^* b \tag{5.42}$$

for x.

5.4.2 Max-Plus Matrix Closure and Longest Paths

The closure operation on complete matrix dioids $\mathcal{D}^{n\times n}$ can be defined easily by extending the definition of single variable closure to matrices using the corresponding matrix operations and solving the equation

$$X = (A \otimes X) \oplus E_n \tag{5.43}$$

where X, A, and E_n are $n \times n$ matrices and E_n is the matrix identity for $\otimes$ as in Definition 5.10.

In the case of matrices over the max-plus dioid, there is a physical model for matrix closure that will provide useful intuition for the techniques to follow in the next chapter. This model is the **maximum path length** problem over directed graphs described by Gondran and Minoux [GM77]. In this section, we review the max-plus closure operation on matrices in light of this graph model, and then give the derivation for the more general dioid closure of matrices. More thorough treatments may be found in the works of Gondran & Minoux and Baccelli et. al. [GM77, BCOQ92].

Suppose we have an n-vertex graph, G, with weighted, directed edges, and wish to find the maximum total weight of all directed paths between all pairs of vertices in the graph. We may assume that from vertex x_j to vertex x_i there is only one or no directed edge since only the length of the longest such directed edge need be considered in any longest path. Such a graph may be represented by the $n \times n$ matrix, A where $A_{i,j} = w$ if and only if there is a directed edge from x_j to x_i with weight w, and $A_{i,j} = \epsilon$ otherwise.

The ordering of indices here may seem to be the reverse of the natural ordering, but it is a natural consequence of writing our matrix equations as $X = AX \oplus E$ rather than $X = XA \oplus E$. In our later study of linear dioid systems, this will allow us to write the system variables as a column vector, and each system equation more identifiably as a row of the matrix rather than a column.

If $p^h_{i,j}$ is the set of all paths to x_i from x_j traveling along a series of h or fewer directed edges, then $m^h_{i,j}$, the length of the longest total path weight among paths in $p^h_{i,j}$, can then be defined recursively as

$$m^h_{i,j} = \max\left[m^{h-1}_{i,j}, \max_k(A_{i,k} + m^{h-1}_{k,j})\right]. \tag{5.44}$$

The expression

$$\max_k (A_{i,k} + m_{k,j}^{h-1}) \tag{5.45}$$

takes the maximum weight of all paths which can be described as a path of $h-1$ or fewer directed edges to an intermediate vertex, x_k, plus a single directed edge from x_k to x_i. Recall that where there is no edge from x_k to x_i, $A_{i,k} = \epsilon$ and that $\epsilon + a = \epsilon$ for all a, and thus we may take the maximum over all such terms, rather than just those for which there is an edge from x_k to x_i.

Nearly all paths in $p_{i,j}^{h-1}$ are also considered in Expression 5.45, which includes the maximum over all paths that can be expressed as an element of $p_{k,j}^{h-2}$ plus an additional edge from x_k to x_i. The only paths of $h-1$ edges or fewer from x_j to x_i which are not accounted for by Equation 5.45 are null-length paths, and so we can also express the relationship in Equation 5.44 as:

$$m_{i,j}^h = \left[\bigoplus_k A_{i,k} \otimes m_{k,j}^{h-1}\right] \oplus m_{i,j}^0 \tag{5.46}$$

where $m_{i,i}^0 = 0$ and $m_{i,j}^0 = \epsilon$ for $i \neq j$.

Note that the term inside the square brackets in Equation 5.46 corresponds to the definition of the $\otimes$ operator for max-plus matrix systems (Equation 5.28) and so we can represent Equation 5.46 with the matrix equation

$$M^h = \left[A \otimes M^{h-1}\right] \oplus M^0, \tag{5.47}$$

where M^h has $m_{i,j}^h$ as its (i,j)th entry. Then M^∞, representing the longest paths in G, satisfies the following equation:

$$M^\infty = [A \otimes M^\infty] \oplus E_n, \tag{5.48}$$

where $M^0 = E_n$ which is the $\otimes$ identity for max-plus matrix multiplication. This is exactly the matrix analogue of Equation 5.46, and matches the form of Equation 5.33. Thus, if the **minimum** solution for M^∞ is the solution to the maximum path length problem, we can use the max-plus matrix closure of A to solve for M^∞.

Example 5.2 *Intuition that the* **minimum** *solution to Equation 5.48 gives the maximum path lengths can be found by looking at a simple graph with two nodes and two* *directed edges. In this graph there is a directed edge of weight b from x_0 to x_1, and one of weight 0 from x_1 to itself. The corresponding equation,*

$$x_1 \quad = \quad x_1 \oplus b x_0, \tag{5.49}$$

has matrix form

$$M^{\infty} = \begin{bmatrix} \epsilon & \epsilon \\ b & 0 \end{bmatrix} \otimes M^{\infty} \oplus E. \tag{5.50}$$

Although the minimum solution is

$$M^{\infty} = \begin{bmatrix} 0 & \epsilon \\ b & 0 \end{bmatrix}, \tag{5.51}$$

there are infinitely many solutions, all of the form

$$\begin{bmatrix} 0 & \epsilon \\ c & 0 \end{bmatrix}, \tag{5.52}$$

where $c \geq b$, yet clearly, the longest path from x_0 to x_1 has length b.

More formally, we can show that the value of the longest path to x_i from x_0 is never bigger than the value of x_i in the minimum solution to the system. To see this we note the following:

- If the length of the longest path to x_i is ϵ, then the minimum solution to the system cannot be smaller since ϵ is our smallest element.

- The length of the longest path to x_i is infinite if and only if there is a positive cycle in the graph, but in this case $M^{\infty}_{i,0}$ must also have no upper bound since $M^{n}_{i,i} > 0$ for all x_i on that cycle.

- If the length of the longest path to x_i is some $\alpha \in \mathcal{R}$ then we may assume that path is simple, since following a negative or zero weight cycle on the path would make the path to x_i no longer than the same path with the cycle removed, and the existence of a positive weight requires the maximum path weight be infinite. However, the existence of such a path requires that the minimum solution for x_i be at least as large as the path weight.

5.4.3 Closure for Matrix Dioids

Definition 5.14 *As with single variable closure, a matrix A's* **matrix closure** *is commonly denoted A^*.*

Lemma 5.1 Baccelli et. al. ([BCOQ92], Lemma 4.101) *Matrix closure can be calculated using the following recursive formulation.*

$$\begin{bmatrix} W & X \\ Y & Z \end{bmatrix}^* = \begin{bmatrix} W^* \oplus W^*X(YW^*X \oplus Z)^*YW^* & W^*X(YW^*X \oplus Z)^* \\ (YW^*X \oplus Z)^*YW^* & (YW^*X \oplus Z)^* \end{bmatrix}. \quad (5.53)$$

Proposition 5.3 *The max-plus closure can be determined using calculation method given in Lemma 5.1 in time $O(n^3)$ for an $n \times n$ matrix.*

Proof Sketch: We do not give the full proof of Proposition 5.3 here, but note that it is easily proved for matrices in which $n = 2^k$ for some integer k using the formula

$$T(n) = \underbrace{2 \cdot T(\frac{n}{2})}_{\text{closure operations}} + \underbrace{O(n^2)}_{\text{matrix adds}} + \underbrace{O(n^3)}_{\substack{\text{matrix}\\ \text{multiplications}}}, \quad (5.54)$$

giving us

$$T(n) = 2 \cdot T(\frac{n}{2}) + O(n^3). \quad (5.55)$$

Combined with the fact that $T(1) = 1$, this can be easily shown to yield $T(n)$ in $O(n^3)$. □

We can relate Equation 5.53 intuitively to the longest paths formulation above with the following observation. The division of the matrix A into the four parts W, X, Y and Z can be thought of as dividing the graph G's vertices, V, into one of two disjoint sets of vertices, V_1 and V_2 such that sub-matrices W and Z represent all edges within V_1 and V_2 respectively, while X represents all edges into V_1 from V_2 and Y represents all edges into V_2 from V_1. Thus the expression

$$(YW^*X \oplus Z)^* \quad (5.56)$$

describes all paths which originate and end in V_2 as being some number of concatenations (the outer *) of paths which either stay entirely in V_2 (denoted by Z), or travel to V_1 (X), move about in V_1 only (W^*) and then come back to V_2 from V_1 (Y).

Equation 5.53 is easily obtained using the equations

$$\begin{bmatrix} W & X \\ Y & Z \end{bmatrix}^* = \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{bmatrix} = \begin{bmatrix} W & X \\ Y & Z \end{bmatrix} \otimes \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{bmatrix} \oplus \begin{bmatrix} e & \epsilon \\ \epsilon & e \end{bmatrix}, \qquad (5.57)$$

to generate the four equations

$$v_{11} = Wv_{11} \quad \oplus \quad Xv_{21} \oplus e \qquad (5.58)$$
$$v_{12} = Wv_{12} \quad \oplus \quad Xv_{22} \qquad (5.59)$$
$$v_{21} = Yv_{11} \quad \oplus \quad Zv_{21} \qquad (5.60)$$
$$v_{22} = Yv_{12} \quad \oplus \quad Zv_{22} \oplus e. \qquad (5.61)$$

Then using the solution technique of Equations 5.39 and 5.42, Equation 5.58 becomes

$$v_{11} = W^*(Xv_{21} \oplus e) \qquad (5.62)$$

and Equation 5.59 becomes

$$v_{12} = W^*Xv_{22}. \qquad (5.63)$$

These two expressions may then be substituted into Equations 5.60 and 5.61, to obtain values for v_{21} and v_{22} from which the final values for v_{11} and v_{12} may be obtained.

Chapter 6

MAX-PLUS SOLUTION OF UBCS

In this chapter, we will describe and prove correct the `UBCsolv` method for finding the maximum solution to a system of inequalities in which linear max-plus expressions provide upper bounds for single variables. These inequalities are the **upper bound constraints** originally introduced in Chapter 2 and which will be defined again in Section 6.4. When combined with the material in Chapter 7, this chapter will provide the `MPsolv` method for finding the maximum solution to any linear max-plus system. The presentation of `UBCsolv` and `MPsolv` in this chapter and the next expands upon a previously published version of these techniques [WB95]. `MPsolv` consists of three steps:

- translating each linear max-plus equation into a small set of upper bound constraints [Chapter 7],
- choosing a subset of these constraints whose maximum solution is easily calculated using the dioid closure operation [Sections 6.2 and 6.4], and
- using that subset's maximum solution to guide the choice of a new constraint subset with a smaller maximum solution [Section 6.3].

The last two steps above are repeated until either the process converges upon a solution which meets all the initial constraints, or it is found that the system only has solutions that include at least one variable with value ϵ. Such systems are called **inconsistent**.

Definition 6.1 *A system of linear max-plus equations over the variables* $\{x_0, \ldots, x_{n-1}\}$ *is* **consistent** *if it has at least one solution for which no* $x_i = \epsilon$*. If a system of linear max-plus equations is not consistent, it is called* **inconsistent**

The `UBCsolv` solution technique will rely on our ability to distinguish ordered, but possibly identical elements, from ordered but distinct elements. We make such a distinction through use of the "$>$" relation.

Definition 6.2 *For all dioids $(\mathcal{D}, \oplus, \otimes)$ we say a is* **strictly greater than** *b, denoted $a > b$, for $a, b \in \mathcal{D}$ if and only if $a \geq b$ and $a \neq b$. The expressions $a > b$ and $b < a$ are equivalent.*

We may also extend this relationship to vectors of bounds.

Definition 6.3 *Given $l = [l_0, l_1, \ldots, l_{n-1}]^T$ and $l' = [l'_0, l'_1, \ldots, l'_{n-1}]^T$, we say that vector l is* **greater than or equal to** *vector l', written $l \geq l'$ precisely when $l \oplus l' = l$. Vector l is* **strictly greater than** *vector l', written $l > l'$, if $l \geq l'$ and $l \neq l'$.*

While this chapter and the next discuss the solution of linear max-plus systems, the techniques presented here apply to any linear system over a commutative dioid, $\mathcal{D}$, for which the following two additional properties hold:

- the structure $(\mathcal{D} \setminus \{\epsilon\}, \otimes)$, is a **group**, indicating that for all $a \in \mathcal{D} \setminus \{\epsilon\}$, there exists an element, a^{-1} such that $a \otimes a^{-1} = e$, and
- the $\oplus$ operation induces a **total ordering** on the elements of $\mathcal{D}$ — for any a and b in $\mathcal{D}$,

$$a = a \oplus b \quad \text{or} \quad b = a \oplus b. \tag{6.1}$$

Such a dioid has been termed a **pseudoring** by Wagneur [Wag91] and has the following property of which we will make extensive use in this chapter.

Proposition 6.1 *For all totally ordered dioids $(\mathcal{D}, \oplus, \otimes)$ one of the following is true for any pair $a, b \in \mathcal{D}$:*

- *$a > b$,*
- *$a = b$, or*
- *$b > a$.*

Proof: This follows directly since by Equation 6.1 and Definition 6.2 for all $a, b \in \mathcal{D}$ either $a \geq b$ or $b \geq a$ $\square$

6.1 Upper Bound Constraints

We now give the formal definition of a generalized upper bound constraint, as originally introduced in Section 2.3.

Definition 6.4 *An* **upper bound constraint (UBC)** *is a linear inequality of the form*

$$x_k \leq a_0 x_0 \oplus a_1 x_1 \oplus \cdots \oplus a_{n-1} x_{n-1},$$

where the x_i terms are variables and the a_i terms are constants from $\mathcal{D}$. In a system of m UBCs over n variables $\{x_0, x_1, \ldots, x_{n-1}\}$, the ith such UBC, u_i, is written

$$\mathbf{x}_{\tau(\mathbf{i})} \leq \bigoplus_{j=0}^{n-1} a_{i,j} \otimes \mathbf{x}_{\mathbf{j}} \tag{6.2}$$

where $x_{\tau(i)}$ is said to be the **target** *of the equation, and the function $\tau(i)$ indexes the target of the ith such UBC.*

It is important to realize that for interesting linear max-plus systems, we often have more than one UBC targeting a single x_i, thus requiring the indexing function τ.

Several of the proofs in this chapter rely upon expressing the right hand side of a UBC targeting a given x_i independently of x_i. To accomplish this we introduce the following proposition.

Proposition 6.2 *For any given system, $\mathcal{U}$, of m UBCs over n variables either $\mathcal{U}$ is inconsistent or there exists an equivalent system, $\mathcal{U}'$, of m or fewer UBCs over n variables in which every $a_{i,\tau(i)} = \epsilon$.*

Proof: This proposition is a direct consequence of Proposition 5.2, which states that any linear max-plus equation may be written in canonical form as in Equation 5.23. Suppose we have $u_i \in \mathcal{U}$, for which $a_{i,\tau(i)} \neq \epsilon$. Then we may write u_i as

$$\mathbf{x}_{\tau(\mathbf{i})} \leq [a_{i,\tau(i)} \otimes \mathbf{x}_{\tau(\mathbf{i})}] \oplus \left[\bigoplus_{j \neq \tau(i)} a_{i,j} \otimes \mathbf{x}_{\mathbf{j}} \right]. \tag{6.3}$$

Applying the definition of "$\leq$" (Equation 5.10) we get

$$[(0 \oplus a_{i,\tau(i)}) \otimes \mathbf{x}_{\tau(\mathbf{i})}] \oplus \left[\bigoplus_{j \neq \tau(i)} a_{i,j} \otimes \mathbf{x_j} \right] = [a_{i,\tau(i)} \otimes \mathbf{x}_{\tau(\mathbf{i})}] \oplus \left[\bigoplus_{j \neq \tau(i)}^{n-1} a_{i,j} \otimes \mathbf{x_j} \right]. \quad (6.4)$$

We may then put this equation into canonical form and make the following observations: if $0 > a_{i,\tau(i)}$, then we may convert the equation back into UBC form with $a_{i,\tau(i)} = \epsilon$; otherwise $0 \oplus a_{i,\tau(i)} = a_{i,\tau(i)}$ and both sides of the equation are identical. Then u_i may be removed from the system since it is always satisfied for any assignment of values to the system variables. □

Throughout this chapter it will be convenient to speak about UBCs using one of two different representations. The first is the graph-theoretical representation first introduced in Chapter 2. This graph representation will allow us to take advantage of the longest path description of matrix closure given in Section 5.4.2 as an intuition behind the technique that follows. The second is a matrix representation of a system of UBCs, which will be used to calculate the max-plus closure corresponding to the longest paths in the graphs.

6.1.1 Graph-theoretical representation of systems of UBCs

Definition 6.5 *A system of upper bound constraints* **induces** *a graph as follows:*

- *for each variable, x_i, in the system, there is a single node labeled x_i.*
- *for each upper bound constraint, $x_{\tau(i)} \leq \bigoplus_{j=0}^{n-1} a_{i,j} \otimes x_j$ there is a set of arcs, one for each term $a_{i,j} \otimes x_j$ such that $a_{i,j} \neq \epsilon$. Each such arc*
  - *is directed from x_j to $x_{\tau(i)}$, and*
  - *has label $a_{i,j}$.*

  All arcs arising from terms on the right hand side of a single UBC are **bundled** *together by a line crossing through those arcs at the arrow head.*

Figure 6.1 illustrates a set of max-plus upper bound constraints and the graph they induce. Note the use of the arc bundling convention to distinguish the pairs of UBCs targeting x_2 and x_3. Bundling is not required for the constraints targeting x_1 since each of them contains only one non-ϵ a term on its right hand side.

6.1.2 Matrix representation of systems of UBCs

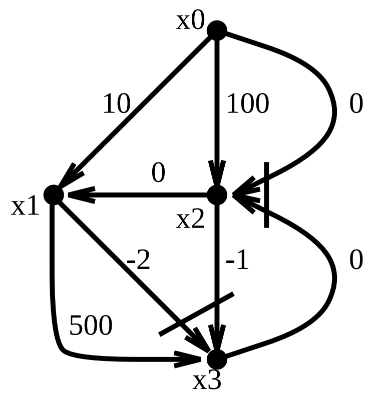
In addition to the graph-theoretical representation shown in Figure 6.1, there is also a matrix representation for the given system of UBCs. This matrix representation is defined below and will be of use both in relating matrix closure to our problem and in the proofs of Section 6.3.

Definition 6.6 *The* **matrix formulation** *of a system of m UBCs over n variables as given in Definition 6.2 is the matrix equation*

$$(J \oplus A) \otimes x = A \otimes x, \tag{6.5}$$

where A is the $m \times n$ matrix of entries $a_{i,j}$, x is the column vector of n variables x_i, and J is the $m \times n$ matrix with entries $j_{i,\tau(i)} = 0$ and all other entries equal to ϵ.

$$\begin{aligned}
x_1 &\leq 10 \otimes x_0 \\
x_1 &\leq x_2 \\
x_2 &\leq 100 \otimes x_0 \\
x_2 &\leq x_0 \oplus x_3 \\
x_3 &\leq 500 \otimes x_1 \\
x_3 &\leq (-2 \otimes x_1) \oplus (-1 \otimes x_2)
\end{aligned}$$



$$\left(\begin{bmatrix} \epsilon & 0 & \epsilon & \epsilon \\ \epsilon & 0 & \epsilon & \epsilon \\ \epsilon & \epsilon & 0 & \epsilon \\ \epsilon & \epsilon & 0 & \epsilon \\ \epsilon & \epsilon & \epsilon & 0 \\ \epsilon & \epsilon & \epsilon & 0 \end{bmatrix} \oplus \begin{bmatrix} 10 & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 0 & \epsilon \\ 100 & \epsilon & \epsilon & \epsilon \\ 0 & \epsilon & \epsilon & 0 \\ \epsilon & 500 & \epsilon & \epsilon \\ \epsilon & -2 & -1 & \epsilon \end{bmatrix} \right) \otimes \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10 & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 0 & \epsilon \\ 100 & \epsilon & \epsilon & \epsilon \\ 0 & \epsilon & \epsilon & 0 \\ \epsilon & 500 & \epsilon & \epsilon \\ \epsilon & -2 & -1 & \epsilon \end{bmatrix} \otimes \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Figure 6.1: A set of constraints and the graph and matrix representations they induce.

Thus, the i-th UBC

$$\mathbf{x}_{\tau(\mathbf{i})} \leq \bigoplus_{j=0}^{n} a_{i,j} \otimes \mathbf{x}_{\mathbf{j}} \tag{6.6}$$

is expressed as

$$\mathbf{x}_{\tau(\mathbf{i})} \oplus \left[\bigoplus_{j=0}^{n} a_{i,j} \otimes \mathbf{x}_{\mathbf{j}} \right] = \bigoplus_{j=0}^{n} a_{i,j} \otimes \mathbf{x}_{\mathbf{j}}. \tag{6.7}$$

6.2 Bounding Systems of UBCs with Closure

In this section we will demonstrate that for any UBC system, $\mathcal{U}$, the vector made up of the maximum possible solutions for each of the variables x_i when $x_0 = 0$ is itself a solution to $\mathcal{U}$. Furthermore, we will show that there exist certain special subsets $\mathcal{S}$ of $\mathcal{U}$ whose maximum solution when $x_0 = 0$ is easily calculated using max-plus closure.

Proposition 6.3 *For a consistent UBC system, $\mathcal{U}$, if l_i is the maximum possible value for a given x_i when $x_0 = 0$, then the vector $l = [l_0, l_1, \ldots, l_{n-1}]^T$ is a solution to $\mathcal{U}$.*

Proof: Clearly there can be no solution to $\mathcal{U}$ which has any variable larger than l_i and so we need only show that l solves $\mathcal{U}$. If l does not satisfy $\mathcal{U}$, then there must be some UBC $u_i \in \mathcal{U}$, written

$$\mathbf{x}_{\tau(\mathbf{i})} \leq \bigoplus_{j=0}^{n-1} \mathbf{x}_{\mathbf{j}} + a_{i,j} \tag{6.8}$$

such that

$$l_{\tau(i)} > \bigoplus_{j=0}^{n-1} l_j + a_{i,j}. \tag{6.9}$$

Since the l_j terms in the right-hand side of Equation 6.9 bound from above their values in any solution, we have violated the given that $l_{\tau(i)}$ is the maximum possible value of $x_{\tau(i)}$ in any solution to $\mathcal{U}$. Thus l must be a solution to $\mathcal{U}$ □

Definition 6.7 *For a consistent UBC system, $\mathcal{U}$ with l_i the maximum possible value for a given x_i, we call the vector $l = [l_0, l_1, \ldots, l_{n-1}]^T$ the* **maximum solution** *to $\mathcal{U}$.*

By the following proposition we know that such a maximum solution to any subset $\mathcal{S}$ of $\mathcal{U}$ must bound the solutions to $\mathcal{U}$ from above.

Proposition 6.4 *If $m = [m_0, \ldots, m_{n-1}]^T$ and $l = [l_0, \ldots, l_{n-1}]^T$ are the maximum solutions to systems $\mathcal{U}$ and $\mathcal{S}$ where $\mathcal{S} \subseteq \mathcal{U}$, then for all x_i, $l_i \geq m_i$.*

Proof: Suppose there exists some i with $l_i < m_i$. If this is the case then m_i is not a solution for x_i in $\mathcal{U}$ because it does not satisfy all of the constraints in $\mathcal{S}$ and therefore cannot satisfy all of the constraints in $\mathcal{U}$. $\square$

Definition 6.8 *Given a system $\mathcal{U}$ of m upper bound constraints over n variables, a* **targeting subset** *of $\mathcal{U}$ is an $n-1$ element UBC subset, $\mathcal{S}$, in which each variable x_i except x_0 is the target of exactly one constraint.*

Example 6.1 *For the set of UBCs in Figure 6.1, the eight targeting subsets are:*

- $x_1 \leq 10x_0, \quad x_2 \leq 100x_0, \quad x_3 \leq 500x_1$
- $x_1 \leq 10x_0, \quad x_2 \leq 100x_0, \quad x_3 \leq -2x_1 \oplus -1x_2$
- $x_1 \leq 10x_0, \quad x_2 \leq x_0 \oplus x_3, \quad x_3 \leq 500x_1$
- $x_1 \leq 10x_0, \quad x_2 \leq x_0 \oplus x_3, \quad x_3 \leq -2x_1 \oplus -1x_2$
- $x_1 \leq x_2, \quad x_2 \leq 100x_0, \quad x_3 \leq 500x_1$
- $x_1 \leq x_2, \quad x_2 \leq 100x_0, \quad x_3 \leq -2x_1 \oplus -1x_2$
- $x_1 \leq x_2, \quad x_2 \leq x_0 \oplus x_3, \quad x_3 \leq 500x_1$
- $x_1 \leq x_2, \quad x_2 \leq x_0 \oplus x_3, \quad x_3 \leq -2x_1 \oplus -1x_2$

Naturally, there may sometimes be UBC systems in which some variable x_i with $i \neq 0$ has no UBC which targets it. If this is the case, then there is no maximum solution for x_i. Section 6.4 discusses how to handle this situation. For the moment we will assume that there are constraints targeting each x_i.

We can describe each of these targeting subsets in matrix form by using a special **selector matrix**, first described for this problem by Burns [Bur94], to pick out from the matrix expression $(J \oplus A) \otimes x = A \otimes x$ only those rows corresponding to UBCs in the targeting subset.

Conceptually, we define a selector matrix, $\mathcal{P}$, to be an $(n-1) \times m$ such that each row of $\mathcal{P}$ contains exactly one 0 entry and $m-1$ ϵ entries.

In order to assure that only one UBC is chosen to target each variable, we must also require that the product $\mathcal{P}J$ has no more than one 0 in each column, since that assures that each x_i is the target of no more than one UBC. In practice, we want the product $\mathcal{P}A$ to be a square matrix so we can perform matrix closure on it and so we get the following slightly altered definition.

Definition 6.9 *Given a system $\mathcal{U}$ of m UBCs in matrix form, a* **targeting selector** *of $\mathcal{U}$ is an $n \times m$ matrix $\mathcal{P}$ such that each row except the 0th of $\mathcal{P}$ contains exactly one 0, all other entries are ϵ, and*

$$\mathcal{P} \otimes J = \hat{E_n}, \tag{6.10}$$

where $\hat{E_n}$ is the $n \times n$ matrix identical to E_n except that $(\hat{E_n})_{0,0} = \epsilon$.

When applied to the the matrix expression $(J \oplus A) \otimes x = A \otimes x$, a 0 in the jth column of the ith row of $\mathcal{P}$ selects the jth UBC of the system $\mathcal{U}$ as the ith UBC of sub-system $\mathcal{S}$. $\mathcal{P}$, can then be used to express the constraints of a targeting subset in matrix form by multiplying both sides of Equation 6.5 on the left. The top row of the matrix $\mathcal{P}$ and its products $\mathcal{P} \otimes J$ and $\mathcal{P} \otimes A$ are always all ϵ-terms. This is because while x_0 is not the target of any constraint in the targeting subset, we will later apply dioid closure to the matrix $\mathcal{P}A$, and this operation will require a square matrix.

Example 6.2 *Given the matrix expression in Figure 6.1, the targeting selector, $\mathcal{P}$, for the last targeting subset of Example 6.1 above is*

$$\mathcal{P} = \begin{bmatrix} \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & 0 & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & 0 & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & 0 \end{bmatrix}, \tag{6.11}$$

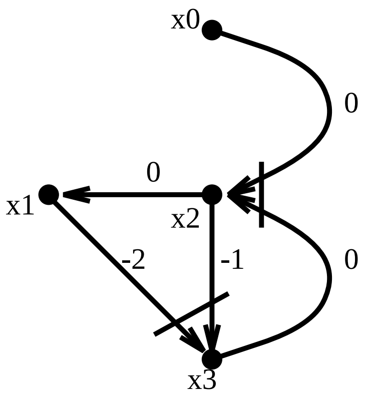
and the sub-system equation resulting from multiplying both left hand sides of the original system equations by $\mathcal{P}$ is

$$\left(\begin{bmatrix} \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & 0 & \epsilon & \epsilon \\ \epsilon & \epsilon & 0 & \epsilon \\ \epsilon & \epsilon & \epsilon & 0 \end{bmatrix} \oplus \begin{bmatrix} \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 0 & \epsilon \\ 0 & \epsilon & \epsilon & 0 \\ \epsilon & -2 & -1 & \epsilon \end{bmatrix} \right) \otimes \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 0 & \epsilon \\ 0 & \epsilon & \epsilon & 0 \\ \epsilon & -2 & -1 & \epsilon \end{bmatrix} \otimes \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{6.12}$$

Definition 6.10 *For a given targeting selector, $\mathcal{P}$ and UBC system $\mathcal{U}$ in matrix form $(J \oplus A) \otimes x = A \otimes x$ as in Definition 6.6 above, the expression*

$$\mathcal{P} \otimes A, \tag{6.13}$$

$$\begin{aligned} x_1 &\le x_2 \\ x_2 &\le x_0 \oplus x_3 \\ x_3 &\le (-2 \otimes x_1) \oplus (-1 \otimes x_2) \end{aligned}$$



$$\begin{bmatrix} \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 0 & \epsilon \\ 0 & \epsilon & \epsilon & 0 \\ \epsilon & -2 & -1 & \epsilon \end{bmatrix}$$

Figure 6.2: A targeting subset, its induced graph, and targeting matrix.

is the **target matrix** *for the targeting subset corresponding to $\mathcal{P}$.*

Example 6.3 *Figure 6.2 gives the equations, graph, and targeting matrix for a single targeting subset of the example in Figure 6.1.*

The targeting matrix $\mathcal{P}A$ above allows us to write a targeting subset, $\mathcal{S}$, of UBC system $\mathcal{U}$ as the $n-1$ UBCs

$$\mathbf{x_i} \le \bigoplus_{j=0}^{n-1} (\mathcal{P}A)_{i,j} \otimes \mathbf{x_j}, \tag{6.14}$$

for $1 \le i \le n-1$. By Proposition 6.4, the maximum solution to the system of Equation 6.14 when $x_0 = 0$ bounds from above the values of all variables in the maximum solution to $\mathcal{U}$ when $x_0 = 0$.

Recall from Section 5.4.3 that for a complete dioid, the closure operation allows us to calculate the minimum solution for X in the equation

$$X = (\mathcal{P}A)X \oplus E_n \tag{6.15}$$

for $n \times n$ matrix X.

By exploiting the definition of max-plus matrix multiply we see that in a complete dioid the 0th column of $(\mathcal{P}A)^*$ must be the minimum solution to the system

$$\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} = (\mathcal{P}A) \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} \oplus \begin{bmatrix} 0 \\ \epsilon \\ \vdots \\ \epsilon \end{bmatrix}. \tag{6.16}$$

Since the 0th row of $(\mathcal{P}A)$ consists of all ϵ entries, Equation 6.16 is equivalent to $x_0 = 0$ plus the $n-1$ equations

$$\mathbf{x_i} = \bigoplus_{j=0}^{n-1} (\mathcal{P}A)_{i,j} \otimes \mathbf{x_j}, \tag{6.17}$$

for $1 \leq i \leq n-1$ when $x_0 = 0$.

We would like to build a solution for the system of Equation 6.14 when $x_0 = 0$ using the solution of Equation 6.17, but there are three obstacles to overcome:

(1) the max-plus dioid as given here is not complete,
(2) the Equation 6.17 is more restrictive than Equation 6.14, and
(3) the closure-derived solution to the system of Equation 6.17 is the *minimum* such solution, and we require the maximum such solution in order to safely bound our solutions from above.

Luckily, it happens that if we choose our targeting selector $\mathcal{P}$ carefully, we can assure that performing closure as a series of substitutions as in Equation 5.34 does not require infinite series of $\oplus$ operations, and that the minimum solution to the system of Equation 6.17 will be the maximum solution to the system of Equation 6.14. We begin this effort with the following definitions.

Definition 6.11 *For any directed path p from x_i to x_j, the* **$\otimes$-generalized path weight from x_i to x_j** *is the $\otimes$ of all arc weights along the path. When the directed path is a simple cycle, we refer to its $\otimes$-generalized simple cycle weight.*

Throughout this chapter whenever path or cycle weights are mentioned, they should be assumed to be the $\otimes$-generalized path and cycle weights.

$$\begin{aligned} x_1 &\le x_2 \\ x_2 &\le x_0 \oplus x_3 \\ x_3 &\le 500 \otimes x_1 \end{aligned}$$

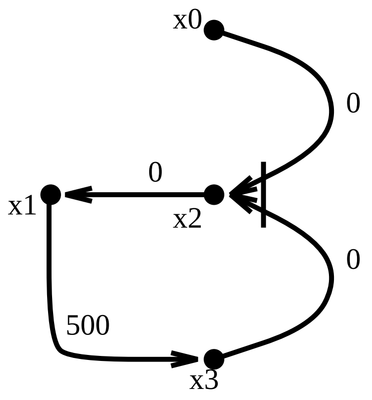


Figure 6.3: An unsafe targeting subset of the constraints in Figure 6.1 and the subgraph they induce.

Definition 6.12 *Given a system $\mathcal{U}$ of upper bound constraints over n variables, a* **safe targeting subset** *of $\mathcal{U}$ is a* **targeting subset** *of $\mathcal{U}$ in which all directed simple cycles in the induced graph have $\otimes$-generalized simple cycle weights strictly less than 0. When this is true, $\mathcal{P}$, is said to be a* **safe targeting selector** *of the system.*

Example 6.4 *For the set of UBCs in Figure 6.1, the only unsafe targeting subset listed in Example 6.1 is*

- $x_1 \le x_2, \quad x_2 \le x_0 \oplus x_3, \quad x_3 \le 500x_1,$

which is pictured in Figure 6.3. All other targeting subsets are safe.

Luckily, the following theorem removes obstacles 1 and 3 above.

Theorem 6.1 Baccelli et. al. ([BCOQ92], Theorem 3.17) *For matrix A with induced graph $\mathcal{G}(A)$, if the cycle weights in $\mathcal{G}(A)$ are all negative then there is a unique solution to the equation $x = Ax \oplus b$, which is given by $x = A^*b$.*

Proof: We give here the somewhat simplified proof with $b = E_n$. Recall that A^*, the closure of A is defined as the minimum solution for x in the equation $x = Ax \oplus E_n$, which expands as in Equation 5.34 to

$$x = A^k x \oplus \left[A^{k-1} \oplus \cdots \oplus A \oplus E_n\right] \tag{6.18}$$

for any $k \geq 1$. We then know that

$$x = A^n x \oplus \bigoplus_{j=0}^{n-1} A^j, \tag{6.19}$$

for the special case when $k = n$ and that if we sum together the left and right hand sides of Equation 6.18 for values of k between 0 and $n-1$ we can obtain

$$x = \bigoplus_{k=0}^{n-1} \left[A^k x \oplus \bigoplus_{j=0}^{k-1} A^j \right]. \tag{6.20}$$

By the idempotent and commutative properties of the matrix $\oplus$ operation, we can rephrase Equation 6.20 as

$$x = \left[\bigoplus_{k=0}^{n-1} A^k \right] x \oplus \left[\bigoplus_{k=0}^{n-1} A^k \right]. \tag{6.21}$$

Together Equations 6.19 and 6.21 then yield

$$[A^n x] \oplus \bigoplus_{k=0}^{n-1} A^k = \left[\bigoplus_{k=0}^{n-1} A^k \right] x \oplus \left[\bigoplus_{k=0}^{n-1} A^k \right]. \tag{6.22}$$

Now, since A^n represents all paths of exactly n edges in $\mathcal{G}(A)$, and there are n nodes in $\mathcal{G}(A)$, these paths all must contain negative cycles. Therefore, for all indices i and j of A,

$$A^n_{i,j} < \bigoplus_{k=0}^{n-1} A^k_{i,j}. \tag{6.23}$$

We may therefore remove the term $A^n x$ from Equation 6.22, leaving us

$$\bigoplus_{k=0}^{n-1} A^k, = \left[\bigoplus_{k=0}^{n-1} A^k \right] x \oplus \left[\bigoplus_{k=0}^{n-1} A^k \right]. \tag{6.24}$$

We then use Equation 6.21 to substitute x for the right hand side of Equation 6.24, obtaining

$$x = \bigoplus_{k=0}^{n-1} A^k. \tag{6.25}$$

This is an exact formula for x, which must therefore admit no other solution and which does not require a complete dioid for calculation. □

Theorem 6.1 then tells us that when $\mathcal{P}$ is safe there is only one solution to Equation 6.17, and therefore it is both minimum and maximum, and we need only show that the solution to the system of Equation 6.17 is the maximum solution to the system of Equation 6.14.

Lemma 6.1 *For any UBC system $\mathcal{U}$ with safe targeting subset $\mathcal{S}$ consisting of the upper bound constraints*

$$\mathbf{x_i} \leq \bigoplus_{j=0}^{n-1} (\mathcal{P}A)_{i,j} \otimes \mathbf{x_j},$$

the vector

$$l = [l_0 = 0, l_1, l_2, \ldots, l_{n-1}]^T$$

where $l_i = (\mathcal{P}A)^*_{i,0}$ is a solution to $\mathcal{S}$ when $x_0 = 0$.

Proof: By the definition of dioid closure, we know that

$$(\mathcal{P}A)^* = (\mathcal{P}A) \otimes (\mathcal{P}A)^* \oplus E_n, \tag{6.26}$$

and therefore

$$l = (\mathcal{P}A) \otimes l \oplus \overline{e_0}, \tag{6.27}$$

where $\overline{e_0}$ is the n element vector

$$\overline{e_0} = [0, \epsilon, \ldots, \epsilon]^T. \tag{6.28}$$

For $i \neq 0$ this gives us

$$l_i = \bigoplus_{j=0}^{n-1} (\mathcal{P}A)_{i,j} \otimes l_j \tag{6.29}$$

which certainly meets the looser constraints

$$\mathbf{x_i} \leq \bigoplus_{j=0}^{n-1} (\mathcal{P}A)_{i,j} \otimes \mathbf{x_j}, \tag{6.30}$$

and we know that $l_0 = 0$ since the top row of $(\mathcal{P}A)$ is constrained to contain only ϵ entries. $\square$

Lemma 6.2 *A maximum solution to any safe targeting subset, $\mathcal{S}$, when $x_0 = 0$ exists and is the solution given in Lemma 6.1.*

Proof: Theorem 6.1 tells us that there is a maximum solution, $m = [m_0, m_1, \ldots, m_{n-1}]$ to the system of Equation 6.17, and Lemma 6.1 tells us that that maximum solution satisfies $\mathcal{S}$. Thus we need only show there cannot exist l, a solution to $\mathcal{S}$ with $x_0 = 0$ such that $l_i > m_i$ for some x_i.

Suppose there is a solution, $l' = [l'_0, \ldots, l'_{n-1}]^T$ with $l_0 = 0$ to $\mathcal{S}$ such that for some $i \neq 0$

$$l'_i < \bigoplus_{j=0}^{n-1} (\mathcal{P}A)_{i,j} \otimes l'_j. \tag{6.31}$$

If this is the case, then we may form a new solution, l'' with $l''_j = l'_j$ for $j \neq i$ and $l''_i = \bigoplus_{j=0}^{n-1} (\mathcal{P}A)_{i,j} \otimes l''_j$. This solution has $l''_i > l'_i$ and therefore l' is not a maximum solution. □

Lemma 6.3 *Given a safe targeting matrix, $(\mathcal{P}A)$, corresponding to a safe targeting subset $\mathcal{S}$ of UBC system $\mathcal{U}$, the maximum solution to $\mathcal{S}$ when $x_0 = 0$ is the vector*

$$\left[(\mathcal{P}A)^*_{0,0}, \ldots, (\mathcal{P}A)^*_{i,0}, \ldots, (\mathcal{P}A)^*_{n-1,0}\right]^T \tag{6.32}$$

where $(\mathcal{P}A)^*$ is the closure of $(\mathcal{P}A)$.

Proof: This follows directly from Definition 6.10, Lemma 6.2, and Definition 5.14 (matrix closure). □

6.3 Generating successively smaller upper bounds

Lemma 6.3 tells us that if we can find a safe targeting subset $\mathcal{S}$ of a system of UBCs $\mathcal{U}$, $\mathcal{S}$'s maximum solution when $x_0 = 0$ is easy to calculate. By Proposition 6.4, the maximum solution to $\mathcal{S}$ when $x_0 = 0$ provides upper bounds for maximum solution to each of the values of the x_i in $\mathcal{U}$ when $x_0 = 0$. We would now like to show how, given a safe targeting subset $\mathcal{S}$ and its associated maximum solution, we can find another safe targeting subset, $\mathcal{S}'$, which generates a tighter bound.

Given a safe targeting subset, $\mathcal{S}$, and its associated bound, l, a new safe targeting subset $\mathcal{S}'$ producing a tighter upper bound, l', can be found by individually evaluating the bounds produced by each UBC in the entire system. For each UBC in the original system, we test the upper bound it gives its target when the variables in that UBC's right hand side take on the values in l. If any UBC produces a bound for its target,

$x_{\tau(i)}$, which is less than the current bound $l_{\tau(i)}$, then it can be exchanged with the UBC currently targeting $x_{\tau(i)}$, and the resulting targeting subset will be safe and produce a tighter bound.

In order to prove this we will need to show both that such an exchange results in a safe targeting subset and that the corresponding bound is lower. To this end, we now introduce the following definition and lemmas.

Definition 6.13 *If $\mathcal{U}$ is the set of all UBCs in a system, then given a UBC, $u_i \in \mathcal{U}$,*

$$u_i : \mathbf{x}_{\tau(\mathbf{i})} \leq \bigoplus_{j=0}^{n-1} a_{i,j} \otimes \mathbf{x}_\mathbf{j}, \tag{6.33}$$

and a vector $l = [l_0, \ldots, l_{n-1}]^T$ of values bounding the variables $\mathcal{X}$ of a system, $u_i(l)$, the **value of u subject to l** *is the value*

$$u_i(l) = \bigoplus_{j=0}^{n-1} a_{i,j} \otimes l_j. \tag{6.34}$$

Lemma 6.4 *Given a safe targeting subset $\mathcal{S}$ of UBC system $\mathcal{U}$, with maximum solution l, UBC $v \in \mathcal{S}$ targeting $x_{\tau(i)}$ and an additional UBC $u \in \mathcal{U}$ also targeting $x_{\tau(i)}$, if $u(l) < l_{\tau(i)}$ then $\mathcal{S}' = \mathcal{S} \cup \{u\} - \{v\}$ is also a safe targeting subset.*

Proof: Clearly $\mathcal{S}'$ is a targeting subset since one UBC targeting $x_{\tau(i)}$ has been exchanged for another. $\mathcal{S}'$ is safe so long as the $\otimes$-generalized weights of all cycles in its induced subgraph are less than 0. Thus, we assume that a cycle of weight 0 or greater does exist in $\mathcal{S}'$ and then show this contradicts the hypothesis that $u(l) < l_{\tau(i)}$.

If the exchange of u for v introduces a cycle of weight $\omega \geq 0$ into the induced graph, it must be the case that that cycle includes one of the arcs induced by constraint u. Let x_j be at the tail end of that arc, and let $a_{i,j}$ be the weight of that arc. Furthermore, let α be the $\otimes$-weight of the path from $x_{\tau(i)}$ to x_j along the corresponding "bad" cycle. All the arcs which are present in α are present in the graphs induced by both $\mathcal{S}$ and $\mathcal{S}'$, since only the edges directed to $x_{\tau(i)}$ have changed. Since the cycle in $\mathcal{S}'$ has weight $\omega \geq 0$, it must be the case that

$$0 \leq \omega = \alpha \otimes a_{i,j}. \tag{6.35}$$

A picture of this situation is given in Figure 6.4.

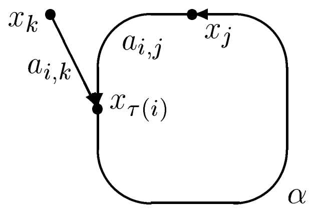


Figure 6.4: Diagram supporting the argument for Lemma 6.4.

Consider now what this means for the values in l. The longest path from $x_{\tau(i)}$ to x_j in the graph induced by $\mathcal{S}$ must have weight $\geq \alpha$ since all directed edges along that α-weight path are in $\mathcal{S}$. Therefore

$$l_j \geq \alpha \otimes l_{\tau(i)}. \tag{6.36}$$

We also know that

$$u(l) \geq a_{i,j} \otimes l_j \tag{6.37}$$

since $u(l)$ was the $\oplus$ of several terms including this one. Together, Equations 6.36 and 6.37 give us

$$u(l) \geq a_j \otimes \alpha \otimes l_{\tau(i)}. \tag{6.38}$$

Now since $\alpha \otimes a_j \geq 0$ this means that $u(l) \geq l_{\tau(i)}$, which directly contradicts the given that $u(l) < l_{\tau(i)}$, and so $\mathcal{S}'$ must be a safe targeting subset of $\mathcal{U}$. □

Lemma 6.5 *Let $\mathcal{S}$ and $\mathcal{S}'$ be safe targeting subsets of UBC system $\mathcal{U}$ such that $\mathcal{S}$ and $\mathcal{S}'$ differ only in that UBC v targets x_i in $\mathcal{S}$ and u targets x_i in $\mathcal{S}'$. Then if $\mathcal{S}$ and $\mathcal{S}'$ have maximum solutions l and l' respectively, with $u(l) < l_i$, it must be the case that $l' < l$.*

Proof: The proof of this lemma relies upon the following observation: when calculating the maximum solutions for $\mathcal{S}$ and $\mathcal{S}'$, the matrix upon which we perform the closure operation differs only in the row corresponding to x_i's target. We may then examine the two resulting closure matrices using a recursive definition as in

Equation 5.53, in which the matrices are unevenly divided so as to make more evident the similarities between the two.

Suppose, without loss of generality, that x_i, the variable targeted by u and v, is the highest numbered variable, x_{n-1}. If not the variables can be reordered to achieve this. In this case, the target matrices $\mathcal{C} = \mathcal{P}A$ and $\mathcal{C}' = \mathcal{P}'A$ corresponding to $\mathcal{S}$ and $\mathcal{S}'$ respectively have form

$$\mathcal{C} = \begin{bmatrix} W & X \\ Y & Z \end{bmatrix}, \text{ and } \mathcal{C}' = \begin{bmatrix} W & X \\ Y' & Z' \end{bmatrix}, \tag{6.39}$$

where W is an $(n-1) \times (n-1)$ matrix, X is an $(n-1) \times 1$ matrix, Y and Y' are $1 \times (n-1)$ matrices, and Z and Z' are 1×1 matrices. In order to show that $l' < l$, we will show that $l'_i \leq l_i$ for all $i \neq n-1$, and that $l'_{n-1} < l_{n-1}$.

We begin with the following two claims which demonstrate the similarity of $\mathcal{C}$ and $\mathcal{C}'$.

Claim:$\underline{Z = \epsilon \text{ and } Z' = \epsilon.}$ This is achieved as a direct result of Proposition 6.2, which tells us that the UBC targeting x_{n-1} does not include a term $a \otimes x_{n-1}$ for $a \neq \epsilon$.

Claim:$\underline{(YW^*X \oplus Z)^* = 0 \text{ and } (Y'W^*X \oplus Z')^* = 0.}$ This follows from the maximum length path description (Section 5.4.2) of max-plus closure. Both expressions measure the longest path from x_{n-1} to itself. We know that the zero edge path from x_{n-1} to itself is 0, giving this expression a lower bound of 0. Any path along one or more edges from x_{n-1} to itself can be broken into a (possibly infinite) concatenation of simple cycles, but since all such cycles have negative weight, the maximum total path weight is less than 0.

We can now see that $\mathcal{C}^*$ and $\mathcal{C}'^*$ are simplified by these observations so that by applying Equation 5.53 we get

$$\mathcal{C}^* = \begin{bmatrix} W^* \oplus W^*XYW^* & W^*X \\ YW^* & 0 \end{bmatrix}, \mathcal{C}'^* = \begin{bmatrix} W^* \oplus W^*XY'W^* & W^*X \\ Y'W^* & 0 \end{bmatrix}, \tag{6.40}$$

We can now show that $l' < l$. To do this we define $\mathcal{F}_n$ and $\mathcal{F}_{n-1}$ to be (respectively) the $n \times 1$ and $(n-1) \times 1$ column vectors with ϵ entries everywhere except that $(\mathcal{F}_n)_0 = 0$ and $(\mathcal{F}_{n-1})_0 = 0$. Thus $C^*\mathcal{F}_n$ extracts from C^* the column vector $\left[C^*_{0,0}, \ldots, C^*_{i,0}, \ldots, C^*_{n-1,0}\right]^T$, which, by Lemma 6.3 is the maximum solution to the UBC subset $\mathcal{S}$ when $x_0 = 0$.

Claim:$l'_{n-1} < l_{n-1}$. First, we note that we have been given

$$u(l) < l_{n-1}. \tag{6.41}$$

By definition

$$u(l) = [Y' \;\; Z] \otimes l = [Y' \;\; Z] \otimes \begin{bmatrix} W^* \oplus W^*XYW^* & W^*X \\ YW^* & 0 \end{bmatrix} \otimes \mathcal{F}_n, \tag{6.42}$$

which is equivalent to

$$u(l) = [Y' \;\; Z] \otimes \begin{bmatrix} W^* \oplus W^*XYW^* \\ YW^* \end{bmatrix} \otimes \mathcal{F}_{n-1}, \tag{6.43}$$

since both $\mathcal{F}_n$ and $\mathcal{F}_{n-1}$ select out only the first column of the matrices to which they are applied. Next, note that since $Z = \epsilon$, Equation 6.43 is equivalent to

$$u(l) = Y' \otimes (W^* \oplus W^*XYW^*) \otimes \mathcal{F}_{n-1}. \tag{6.44}$$

By distributing through the $\mathcal{F}$ and Y' terms on the right and left hand sides of the above equation, we can see that

$$u(l) = Y'W^*\mathcal{F}_{n-1} \oplus Y'W^*XYW^*\mathcal{F}_{n-1}, \tag{6.45}$$

which in turn yields

$$u(l) \geq Y'W^*\mathcal{F}_{n-1}, \tag{6.46}$$

by the definition of "$\geq$". By inspecting the matrix $\mathcal{C}'$ in Equation 6.40 we can verify that

$$l'_{n-1} = Y'W^*\mathcal{F}_{n-1}. \tag{6.47}$$

So together, Equations 6.41, 6.46 and 6.47 yield

$$l'_{n-1} < l_{n-1}. \tag{6.48}$$

Claim:$l' < l$ Since we have shown that $l'_{n-1} < l_{n-1}$, it only remains to show that $l' \leq l$. This amounts to showing that

$$\begin{bmatrix} W^* \oplus W^*XY'W^* & W^*X \\ Y'W^* & 0 \end{bmatrix} \otimes \mathcal{F}_n \leq \begin{bmatrix} W^* \oplus W^*XYW^* & W^*X \\ YW^* & 0 \end{bmatrix} \otimes \mathcal{F}_n, \tag{6.49}$$

which is reduced to showing that

$$\begin{bmatrix} W^* \oplus W^*XY'W^* \\ Y'W^* \end{bmatrix} \otimes \mathcal{F}_{n-1} \leq \begin{bmatrix} W^* \oplus W^*XYW^* \\ YW^* \end{bmatrix} \otimes \mathcal{F}_{n-1}, \tag{6.50}$$

as was done in Equation 6.43. Since we already know that $l'_{n-1} < l_{n-1}$, Equation 6.47, plus the similarly derived

$$l_{n-1} = YW^*\mathcal{F}_{n-1}. \tag{6.51}$$

give us

$$Y'W^*\mathcal{F}_{n-1} < YW^*\mathcal{F}_{n-1}, \tag{6.52}$$

and we need only show that

$$[W^* \oplus W^*XY'W^*] \otimes \mathcal{F}_{n-1} \leq [W^* \oplus W^*XYW^*] \otimes \mathcal{F}_{n-1}. \tag{6.53}$$

However, it is straightforward to obtain Equation 6.53 from Equation 6.52 — we simply $\otimes$ both sides of the inequality with W^*X on their left hand sides, and then perform $\oplus$ with $W^*\mathcal{F}_{n-1}$ to both, yielding Equation 6.53. □

Lemma 6.6 *Given $\mathcal{U}$, a consistent UBC system, $\mathcal{S}$ a safe targeting subset of $\mathcal{U}$, and l, the maximum solution to $\mathcal{S}$ when $x_0 = 0$, if there exists UBC $u_i \in \mathcal{U}$ with $u_i(l) < l_{\tau(i)}$ and $x_{\tau(i)}$ is the target of u_i, then the UBC subset $\mathcal{S}' = \{u_i\} \cup \mathcal{S} \setminus \{v\}$ where v is the constraint targeting $x_{\tau(i)}$ in $\mathcal{S}$ is a safe targeting subset of $\mathcal{U}$ and the maximum solution to $\mathcal{S}'$ when $x_0 = 0$ is strictly less than l.*

Proof: This follows directly from Lemmas 6.4 and 6.5. □

Lemma 6.6 gives us what will be the key feature of the `UBCsolv` technique: given a safe targeting subset, $\mathcal{S}$, we may find a new safe targeting subset by applying each of the UBCs of the system to the maximum bound of $\mathcal{S}$. If a u is found targeting x_i such that $u(l)$ is smaller than l_i we exchange the UBC targeting x_i in $\mathcal{S}$ with u and calculate the new maximum solution. For a consistent UBC system, this continues until a solution is converged upon. In order to make this technique we must do two additional things:

- show how to determine an initial safe targeting subset $\mathcal{S}$, and
- show how to handle inconsistent systems.

6.4 Generating an Initial Safe Targeting Subset

Although we have shown how to obtain one safe targeting subset from another, we must first have an initial safe targeting subset. If we know that $\mathcal{U}$ has a maximum solution in which no variable has value greater than $\mathcal{V}$, we could simply augment $\mathcal{U}$ with $n-1$ constraints

$$x_i \leq x_0 \otimes \mathcal{V}.$$

These additional constraints would not change the solution space of $\mathcal{U}$ and would form a targeting subset containing no cycles.

While we cannot assume that for any UBC system $\mathcal{U}$ there is a maximum solution, we note that for any specific solution to $\mathcal{U}$ with $x_0 = 0$, there must be some largest x_i. If $\mathcal{V}$ is greater than the largest such x_i in a given solution, then augmenting the system with these constraints does not change the validity of that solution.

Unfortunately, $\mathcal{V}$ may vary for different solutions to the system, so we do not know *a priori* a suitable value for it. Instead we treat $\mathcal{V}$ as a finite but very large element of $\mathcal{D}$, obeying both the required properties of dioids and the additional properties listed at the beginning of Chapter 6 except:

- $\mathcal{V} \otimes \mathcal{V}$ is undefined, and
- $\mathcal{V}$ has no inverse for $\otimes$.

In addition, we assume $\mathcal{V} \otimes a > b$ for all $a \neq \epsilon, b \in \mathcal{D}$.

Luckily, the dioid closure operation does not require $\otimes$ inverses, and we never need to calculate $\mathcal{V} \otimes \mathcal{V}$. If we begin with the initial target matrix

$$V = \begin{bmatrix} \epsilon & \epsilon & \cdots & \epsilon \\ \mathcal{V} & \epsilon & \cdots & \epsilon \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{V} & \epsilon & \cdots & \epsilon \end{bmatrix}, \tag{6.54}$$

then the initial closure matrix, V^*, has form

$$V^* = \begin{bmatrix} 0 & \epsilon & \cdots & \epsilon \\ \mathcal{V} & 0 & \cdots & \epsilon \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{V} & \epsilon & \cdots & 0 \end{bmatrix}. \tag{6.55}$$

As we replace equations $x_i \leq \mathcal{V} \otimes x_0$ with equations not containing a $\mathcal{V}$ term, we need to calculate the closure of matrices which can be written

$$\begin{bmatrix} V & \bar{\epsilon} \\ Y & Z \end{bmatrix} \tag{6.56}$$

where if V is a $k \times k$ sub-matrix it has entries $V_{i,0} = \mathcal{V}$ for $1 \leq i < k$, and ϵ everywhere else, while the sub-matrix denoted "$\bar{\epsilon}$" above has all entries equal to ϵ. The closure of this matrix can then be seen to be

$$\begin{bmatrix} V^* & \bar{\epsilon} \\ Z^*YV^* & Z^* \end{bmatrix}, \tag{6.57}$$

where V and V^* are related as in Equations 6.54 and 6.55.

Lemma 6.7 *Should the process converge to a consistent solution, l, while some x_i are still bounded with an expression that includes a $\mathcal{V}$-term then those x_i have no maximum solution.*

Proof: We prove this lemma by demonstrating that there is some $r \in \mathcal{R}$ such that all assignments to $\mathcal{V} \geq r$ solve the system.

Let $r = m \otimes (\alpha^{-1} \oplus 0)$ where m is the maximum non-$\mathcal{V}$-term bound in l and α is the smallest non-ϵ coefficient in the UBC system. We know that when `UBCsolv` converges, for every UBC u_i

$$u_i : \mathbf{x}_{\tau(\mathbf{i})} \leq \bigoplus_{j=0}^{n-1} A_{i,j} \otimes \mathbf{x_j} \tag{6.58}$$

in $\mathcal{U}$, u_i is satisfied if and only if

$$l_{\tau(i)} \leq a_{i,j} \otimes l_j \tag{6.59}$$

for some j. We substitute r for $\mathcal{V}$ and note how this affects Equation 6.59.

By our definition of $\mathcal{V}$, it cannot be the case that $l_{\tau(i)}$ contains a $\mathcal{V}$-term while l_j does not. If either $l_{\tau(i)}$ and l_j both contain a $\mathcal{V}$ term, or neither $l_{\tau(i)}$ nor l_j contains a $\mathcal{V}$-term, then the relative values of $l_{\tau(i)}$ and l_j are unchanged when we substitute r

for $\mathcal{V}$, and Equation 6.59 still holds true. If l_j is $\mathcal{V} \otimes a$ and $l_{\tau(i)}$ contains no $\mathcal{V}$-term then l_j becomes $r \otimes a = m \otimes a \otimes (\alpha^{-1} \oplus 0)$ while $l_{\tau(i)}$ does not change. Suppose

$$l_{\tau(i)} > m \otimes a \otimes (\alpha^{-1} \oplus 0), \tag{6.60}$$

thus violating Equation 6.59. Since $l_{\tau(i)} \leq m$, we then have

$$m > m \otimes a \otimes (\alpha^{-1} \oplus 0), \tag{6.61}$$

which then yields

$$0 > (a \otimes \alpha^{-1}) \oplus a. \tag{6.62}$$

By the definition of α, $a \otimes \alpha \geq 0$ and thus Equation 6.59 must be satisfied. Thus substituting r for $\mathcal{V}$ provides a solution to the system. Now, we need only note that the above also holds for any $r' \geq r$, since, in the only interesting case above, this results in the right hand side of Equation 6.60 increasing while its left-hand side stays constant. $\Box$

Thus, for systems with no maximum solution, we are able to give, through the relative values of the $\mathcal{V}$-bounds, a description of an infinite family of solutions to the system $\mathcal{U}$. Note, however, that there may be many other such families of infinite solutions to the system.

6.5 Identifying Inconsistent Systems

If at any time during the process of choosing successive safe targeting subsets and calculating their induced bounds an upper bound of ϵ is discovered for any variable, the process stops and declares that the set of UBCs is inconsistent.

Lemma 6.8 *If, for a given UBC system, there exists a safe targeting subset in whose induced graph there is no path from x_0 to x_i for $0 \neq i$ then that UBC system is inconsistent.*

Proof: If there is no path from x_0 to x_i in the given safe targeting subset then we may divide the variables into two non-empty subsets: V_1, consisting of those x_j to which there is a path from x_0, and V_2, consisting of those x_j to which there is no path from x_0. Then without loss of generality we may order the variables so that

$$(x_j \in V_1) \wedge (x_k \in V_2) \Rightarrow j < k. \tag{6.63}$$

We may then write the safe targeting matrix as

$$\begin{bmatrix} W & X \\ \bar{\epsilon} & Z \end{bmatrix} \tag{6.64}$$

with the matrix divisions corresponding to V_1 and V_2 as in Equation 5.57, and where $\bar{\epsilon}$ consists of all ϵ entries as defined for Equation 6.56, indicating there are no paths from V_1 to V_2. Using Equation 5.53, we see that the closure of this matrix is

$$\begin{bmatrix} W^* & W^*XZ^* \\ \bar{\epsilon} & Z^* \end{bmatrix}, \tag{6.65}$$

indicating that no variable in V_2 may be given an assignment that satisfies the safe targeting subset. □

In order to make sure that the constraint set is consistent, we must also examine any constraints of the form

$$\mathbf{x_0} \leq \bigoplus_{j=0}^{n-1} (\mathcal{P}A)_{0,j} \otimes \mathbf{x_j} \tag{6.66}$$

to make sure they cannot reduce the bound of $x_0 = 0$ any lower. If they can then we must declare the constraint set inconsistent since this indicates that the maximum solution to the system when $x_0 = 0$ requires $x_0 < 0$.

Lemma 6.9 *If there exists a safe targeting subset $\mathcal{S}$ of $\mathcal{U}$ with corresponding maximum solution $l = [l_0, \ldots, l_{n-1}]^T$ and a UBC*

$$u_i : \mathbf{x_0} \leq \bigoplus_{j=0}^{n-1} (\mathcal{P}A)_{0,j} \otimes \mathbf{x_j} \tag{6.67}$$

such that

$$0 > \bigoplus_{j=0}^{n-1} (\mathcal{P}A)_{0,j} \otimes l_j \tag{6.68}$$

then the UBC system is inconsistent.

Proof: The system must be inconsistent because $\mathcal{S}$'s maximum solution and u_i together require that the maximum solution when $x_0 = 0$ be strictly less than 0. □

6.6 The UBCsolv Algorithm

Definition 6.14 *The* `UBCsolv` *algorithm for a system $\mathcal{U}$ of m UBCs over n variables consists of the following steps:*

- *augment $\mathcal{U}$ with the $n-1$ additional constraints $x_i \leq x_0 + \mathcal{V}$ for every $0 < i < n$ and create an initial safe targeting subset, $\mathcal{S}$ with upper bound $l = [0, \mathcal{V}, \ldots, \mathcal{V}]^T$*
- *Do the following*
  - ◦ *Search through $\mathcal{U}$ to find a constraint u targeting some x_i such that $u(l) < l_i$*
  - ◦ *If u targets x_0 declare $\mathcal{U}$ inconsistent and exit the algorithm*
  - ◦ *Exchange u for the UBC targeting x_i in $\mathcal{S}$*
  - ◦ *Calculate $(\mathcal{P}A)^*$, the closure of $\mathcal{S}$'s target matrix*
  - ◦ *Replace each l_i with $(\mathcal{P}A)^*_{i,0}$*
  - ◦ *If any $l_i = \epsilon$, declare $\mathcal{U}$ inconsistent and exit the algorithm*

  until no UBC is found which can decrease the bound l.
- *Report the values in l as the maximum solution to $\mathcal{U}$. If any l_i has a maximum bound containing a $\mathcal{V}$ term, it reports the bounds in terms of the $\mathcal{V}$-expressions and a value for r, as given in Lemma 6.7 that satisfies them.*

6.6.1 Convergence of UBCsolv Within Finite Time

If at any time, no UBC can be applied to the current variable bounds and reduce them, then all UBCs have been satisfied and the current upper bound is the maximum solution to the entire system of UBCs. We now show that within finite time, either this method will declare the UBC system inconsistent, or will converge upon an answer.

Definition 6.15 *For a system of m UBC's over n variables, t, the number of* **terms** *is the total number of non-ϵ terms in the right hand sides of all UBCs.*

Lemma 6.10 *Given a UBC system $\mathcal{U}$ of finite size, the number of safe targeting subsets encountered during* `UBCsolv` *is finite.*

Proof: The number of safe targeting subsets encountered is bounded from above by the number of possible targeting subsets. For a given system of m UBCs over n variables, each of $n-1$ variables x_i with $i \neq 0$ may be the target of at most any of the m original UBCs plus the additional UBC $x_i \leq x_0 + \mathcal{V}$. This gives us a loose upper bound of

$$(m+1)^{n-1} \tag{6.69}$$

distinct safe targeting subsets appearing during the algorithm. Since by Lemma 6.2 each safe targeting subset has a maximum solution, and by Lemma 6.5 each successive safe targeting subset we encounter has maximum solution smaller than the previous one, we may only encounter each safe targeting subset once. □

Lemma 6.11 *The time required by* `UBCsolv` *to find the maximum solution of a given safe targeting subset and use that solution to find a new safe targeting subset with a smaller maximum solution is $O(t+n^3)$.*

Proof: Finding a new constraint to exchange takes at most as much time as it takes to evaluate each constraint and compare the value to the current value. Assuming that additions and compares are each one step the total work is

$$\underbrace{t}_{\substack{\text{calculating}\\ a_i \otimes x_i}} + \underbrace{(t-m)}_{\substack{\text{comparisons}\\ \text{within a UBC}}} + \underbrace{m}_{\substack{\text{compare UBC to}\\ \text{current value}}} = 2t, \tag{6.70}$$

where the first term represents all additions performed, the second represents all comparisons made in evaluating individual UBCs, and the last represents all comparisons between current values and the calculated values. By Proposition 5.3, the time to calculate a new closure matrix is

$$O(n^3). \tag{6.71}$$

This combines with Equation 6.70 to give a total running time of

$$O(t+n^3). \tag{6.72}$$

□

Burns has suggested [Bur95] that for sparse safe targeting subsets Johnson's $O(t^2 \log t + nt)$ all pairs shortest paths algorithm [Joh77, CLR90] be modified to perform the longest paths calculation instead of using the recursive definition of closure given in Equation 5.53. Similarly, the Bellman-Ford $O(nt)$ algorithm [CLR90] may be used as well. Both algorithms find the shortest paths in a directed graph with no negative-weight cycles. For the max-plus case this corresponds to a targeting subset with no positive-weight cycles.

Lemma 6.12 *Given a UBC system of finite size,* `UBCsolv` *converges or declares the system inconsistent in finite time.*

Proof: This follows directly from the fact that each of the finitely many safe targeting subsets (Lemma 6.10) has its maximum solution calculated only once, and that calculation plus finding the next safe targeting subset takes finite time (Lemma 6.11). $\square$

6.6.2 Correctness of the UBCsolv

Lemma 6.13 *If* `UBCsolv` *converges upon a solution to UBC system $\mathcal{U}$ then either that solution is the maximum solution to $\mathcal{U}$ when $x_0 = 0$, or $\mathcal{U}$ has no maximum solution and* `UBCsolv` *describes an infinite family of solutions to $\mathcal{U}$ with increasing maximum variable value.*

Proof: Since the technique will not converge upon a solution unless all UBCs are satisfied, we know that when it finds a solution is solves the UBC system with $x_0 = 0$. To see that this is the maximum solution we note that by Lemma 6.2 the solution bounds it maintains are always the maximum solution to a safe targeting subset for which they are calculated. Lemma 6.7 covers the case when there is no maximum solution but the system is soluble. $\square$

Lemma 6.14 *If* `UBCsolv` *declares a UBC system inconsistent then it is inconsistent.*

Proof: Since by Lemma 6.13 the technique cannot claim to find a maximum solution if all UBCs are not satisfied, we need only show that it never erroneously declares a UBC subset inconsistent. Lemma 6.8 tells us that any time we declare a system inconsistent because we have determined an upper bound of $x_i = \epsilon$ for some $i \neq 0$, we have made the correct choice. Lemma 6.9 handles the case for $x_0 < 0$. $\square$

Theorem 6.2 *The* `UBCsolv` *algorithm for determining the maximum solution to a UBC system* $\mathcal{U}$ *when* $x_0 = 0$ *always converges within finite time to the correct answer.*

Proof: By Lemma 6.12 we know that `UBCsolv` must terminate in finite time. By Lemmas 6.13 and 6.14 we know that `UBCsolv` correctly determines either the maximum solution or that the system is inconsistent. □

Chapter 7

GENERAL LINEAR MAX-PLUS OPTIMIZATION

In this chapter we show how a simple transformation allows us to use the `UBCsolv` technique of Chapter 6 to solve an arbitrary linear max-plus system, as well as to optimize linear max-plus expressions over such a system. We refer to this general max-plus linear system solution and optimization technique as `MPsolv`.

7.1 UBC-Based Solution of General Linear Max-Plus Systems

The solution technique discussed in Chapter 6 works with UBCs in the form given by Equation 6.2. In contrast, a general max-plus linear system as given in Equation 5.32 can be described as a set of m equations of the form:

$$\left[\bigoplus_{j=0}^{n-1} A_{i,j} \otimes x_j\right] \oplus b_i = \left[\bigoplus_{j=0}^{n-1} C_{i,j} \otimes x_j\right] \oplus d_i. \tag{7.1}$$

We can express these equations with upper bound constraints by taking advantage of the following simple Lemma.

Lemma 7.1 *For a and b in any dioid,*

$$[a \geq b \wedge b \geq a] \quad \Leftrightarrow a = b. \tag{7.2}$$

Proof: This follows directly from Equation 5.10.

□

The transformation to UBCs is as follows:

- We create a new set of variables $\mathcal{X} = \{x_0, \ldots, x_{n-1}\} \cup \{\hat{x}_0, \ldots, \hat{x}_{m-1}\} \cup \{x_\emptyset\}$ where the x_i terms represent each of the original x_i terms in the equations, and there is a $\hat{x}_i$ term for each of the m rows in the matrix A. The value 0 is represented by the variable $x_\emptyset$.

- For each row i of a system as given above in Equation 7.1 we create several upper bound constraints which will force each $\hat{x}_i$ to be equal to both sides of the equation it represents.

  - To bound each $\hat{x}_i$ from above we add the equations:

$$\hat{x}_i \leq \left[\bigoplus_{j=0}^{n-1} A_{i,j} \otimes x_j\right] \oplus [x_\emptyset \otimes b_i] \tag{7.3}$$

$$\hat{x}_i \leq \left[\bigoplus_{j=0}^{n-1} C_{i,j} \otimes x_j\right] \oplus [x_\emptyset \otimes d_i]\,. \tag{7.4}$$

  - We cannot bound each $\hat{x}_i$ from below so easily since the "$\geq$" relation faces the wrong direction. However, by the total ordering on elements of the max-plus algebra we know that the equations

$$\left[\bigoplus_{j=0}^{n-1} A_{i,j} \otimes x_j\right] \oplus [x_\emptyset \otimes b_i] \leq \hat{x}_i \tag{7.5}$$

$$\left[\bigoplus_{j=0}^{n-1} C_{i,j} \otimes x_j\right] \oplus [x_\emptyset \otimes d_i] \leq \hat{x}_i \tag{7.6}$$

$$\tag{7.7}$$

    are equivalent to saying that $\hat{x}_i$ is greater than or equal to each of the terms on the left hand side of the inequalities, and so we use the inequalities

$$\begin{array}{lll} x_j \leq -A_{i,j} \otimes \hat{x}_i, & \text{for all } (i,j) \text{ pairs such that} & A_{i,j} > \epsilon \\ x_j \leq -C_{i,j} \otimes \hat{x}_i, & \text{for all } (i,j) \text{ pairs such that} & C_{i,j} > \epsilon \\ x_\emptyset \leq -b_i \otimes \hat{x}_i & \text{for all } i \text{ such that} & b_i > \epsilon \\ x_\emptyset \leq -d_i \otimes \hat{x}_i & \text{for all } i \text{ such that} & d_i > \epsilon, \end{array}$$

    where a minus sign indicates the familiar additive inverse, which is the $\otimes$ inverse in this scheme.

A solution to the original system can then be found by finding the maximum solution to the new system when $x_\emptyset = 0$.

7.1.1 Time Complexity of UBC-Based General Solution

The time required to transform a linear system into UBCs is polynomial in the size of the original problem. Unfortunately, as noted in Section 6.6.1 the process of solving the resulting UBC system, while requiring finite time, is not known to have a polynomial time bound.

Definition 7.1 *Similar to Definition 6.15, for a linear max-plus system of m equations over n variables the number of* **terms***, t, is the total number of non-ϵ entries in the matrices A, b, C, and d.*

Lemma 7.2 *The process of translating a linear max-plus system of t terms over m equations and n variables results in a UBC system with at most $m + n + 1$ variables, $2m + t$ UBCs, and $2t$ terms among the right hand sides of all UBCs.*

Proof: We begin with n variables and add m more, one for each equation, plus another for $x_\emptyset$, giving us $n + m + 1$ total variables. For each equation in the original system, we have two equations of the form $\hat{x} \leq \bigoplus_{i=0}^{n-1} a_i \otimes x_i$, plus as many equations of the form $x_j \leq -a_{i,j} \otimes \hat{x_i}$ as there are non-ϵ entries in the given equation, giving us $2m + t$ total equations. Each entry in the original equation appears on the right hand side of one UBC, and also forces an equation with a single entry on its right hand side, giving us $2t$ terms. □

7.2 Previous Solutions for Specific Cases

There are two previous solutions for more specialized problems within the domain of linear max-plus solution which are worth mention here.

7.2.1 Zimmermann's Max-Plus Minimization Technique

For the special case of max-plus systems which may be written

$$A \otimes x \geq b \tag{7.8}$$

for $m \times n$ matrix A, and m-entry column vector b with entries in $\mathcal{R} \cup \epsilon$, and n-entry column vector of variables x, we may apply the technique of Zimmermann ([Zim81], Chapter 10) for minimizing the expression

$$c \otimes x, \tag{7.9}$$

where c is an n-element row vector. This is phrased as a general technique which requires only that the operation $\oplus$ induces a total order, and that $(\mathcal{R}, \otimes)$ is a group. In this case, the solution is given as

$$\sup_{i \in M} \left[b_i \otimes \inf_{j \in N} \left(c_j \otimes a_{i,j}^{-1} \right) \right], \tag{7.10}$$

where M is the set of rows and N is the set of columns in Equation 7.8.

7.2.2 Baccelli et.al's Max-Plus Symmetrization Technique

In the special case of max-plus systems when the matrices A and C of Equation 5.32 are square, the Max Plus working group at INRIA [Plu90] and Baccelli et.al. [BCOQ92] give an elegant technique for solving such equations, which is based on Cramer's rule for solving linear systems.

In order to accomplish this, they first introduce a new algebra of **max plus balances** which has elements of three types:

- "positive" elements $a = (a, \epsilon)$,
- "negative" elements $\ominus a = (\epsilon, a)$, and
- "balanced" elements $a^\bullet = (a, a)$,

with the following rules for the operations $\oplus$ and $\otimes$:

$$\begin{aligned} (x_1, x_2) \oplus (y_1, y_2) &= (x_1 \oplus y_1, x_2 \oplus y_2) && (7.11) \\ (x_1, x_2) \otimes (y_1, y_2) &= (x_1 y_1 \oplus x_2 y_2, x_1 y_2 \oplus x_2 y_1), && (7.12) \end{aligned}$$

and identity and zero elements $e = (0, \epsilon)$ and $\epsilon = (\epsilon, \epsilon)$. Two elements, $x = (x_1, x_2)$ and $y = (y_1, y_2)$ are said to **balance** each other, written $x \nabla y$ if $x_1 \oplus y_2 = x_2 \oplus y_1$.

Assuming the matrices in Equation 5.32 are in canonical form, we create new matrices A' and b' such that $A'_{i,j} = (A_{i,j}, C_{i,j})$ and $b'_i = (d_i, b_i)$, and express Equation 5.32 as

$$A' x \nabla b'. \tag{7.13}$$

The determinant of A' is defined to be

$$\bigoplus_{\sigma} \operatorname{sgn}(\sigma) \bigotimes_{i=0}^{n-1} A'_{i,\sigma(i)}, \tag{7.14}$$

where σ ranges over all permutations of the integers $\{0, 1, \ldots, n-1\}$, and $\operatorname{sgn}(\sigma) = e$ when σ is an even permutation, and $\ominus e$ when σ is odd. A solution for x_i can then be found by finding the determinant of A' when the ith column is replaced by b', and "dividing" (i.e. the inverse of the $\otimes$ operation for the balance elements — dividing by (t, ϵ) is equivalent to $\otimes$-ing by $(-t, \epsilon)$) by the determinant of A'. When all x_i have "positive" solutions $x_i = (t_i, \epsilon)$ then the solution with all $x_i = t_i$ is a solution to the original system.

Unfortunately, a non-square system cannot be padded with ϵ terms to apply this method, as this results in an ϵ determinant for A', making A' non-invertible.

7.3 A Computationally Expensive General Solution

In his thesis, Gaubert [Gau92] shows that the max-plus balances technique above may also be applied to solve rectangular systems

$$A \otimes x = C \otimes x \tag{7.15}$$

when the matrices A and C have dimensions $m \times n$ for $m > n$. However, this "brute force" technique is computationally very expensive as it requires calculating the determinants of every one of the $(n-1) \times (n-1)$ sub-matrices of A', where A' is as given in the corresponding max-plus balance equation, $A'x\nabla\epsilon$.

7.4 Optimization of General Linear Max-Plus Systems

In this section we discuss minimizing and maximizing max-plus expressions over max-plus systems. In both cases the problem is solved by adding a single variable and no more than $n + 1$ UBCs containing a total of $2n$ non-ϵ terms, and thus the time complexity is similar to that of determining the maximum solution alone.

7.4.1 Maximum Solutions

Finding the maximum solution to a linear max-plus expression subject to the constraints of a linear max-plus system is straightforward. One need only find the maximum solution to the system, and then calculate the value of the expression subject to those values.

Lemma 7.3 *The maximum value of a linear max-plus expression subject to the constraints of a linear max-plus system occurs at the maximum solution to the system.*

Proof: This follows naturally from the maximum solution since no variable can have a larger value, and all linear max-plus expressions monotonically decrease with a decrease in any of its variables. □

7.4.2 Minimum Solutions

As noted in Chapter 2, we naturally phrase time separation problems as "what is the maximum value of x_j when x_i is 0?" The question of minimum separation is easily answered by finding the maximum time of x_i when x_j is 0 and then flipping its sign. Thus, if we wish to find the minimum value of a given expression, it is quite easy to do so as follows:

- Create a new variable, x_α with additional UBCs setting it equal to the expression to be minimized.
- Find maximum value of $x_\emptyset$ when x_α is 0.
- take the $\otimes$ inverse of the value.

Thus while our normal problem generally maximizes the quantity

$$x_i - x_\emptyset \tag{7.16}$$

for all x_i, we now maximize

$$x_\emptyset - x_\alpha, \tag{7.17}$$

and then change the sign of the result.

7.5 Applying the Max-Plus Technique to other Algebras

As mentioned in Chapter 6 the UBC-based technique will work for a subclass of dioid algebras $(\mathcal{D}, \oplus, \otimes)$ in which the elements of $\mathcal{D}$ are totally ordered with respect to the $\oplus$ operation, and the $\otimes$ operation forms a group on $\mathcal{D} \setminus \{\epsilon\}$ where ϵ is the identity element for the $\oplus$ operation.

The proofs and techniques of Chapters 6 and 7 still apply as long as we note the following changes:

- the notion that two elements of $\mathcal{D}$, d_1 and d_2, satisfy the relationship $d_1 \geq d_2$ may differ from intuition — in the min-times algebra, $2 \geq 3$ is a correct statement, and

- the identity elements ϵ and 0 may differ from algebra to algebra — for example in the min-times algebra, the $\oplus$-identity is $+\infty$ and the $\otimes$-identity is 1.

As a result, in the min-times algorithm safe targeting subsets are not targeting subsets whose cycles negative total edge weight, but instead have cycles whose edge weights when multiplied together (in the usual algebra) result in a value of 1 or more.

In order to use the general form of Proposition 6.2 we must define and prove canonical a more general notion of canonical representation than found in Equation 5.23. This is defined and proved correct in Appendix B

Part IV

Practical Algorithmic Solutions: Timing Verification and on Toward Timing Synthesis

Chapter 8
THE SHORTCIRCUIT ALGORITHM

In this section we present the `ShortCircuit` algorithm for determining the maximum possible time separations between events whose temporal relationships are expressed solely with UBCs. The algorithm overcomes both of the problems noted for McMillan and Dill's algorithm in Section 4.3.1. Figure 8.1 repeats the simplified version of McMillan and Dill's algorithm originally given in Figure 4.3.

Figure 8.2 gives the text of the `ShortCircuit` algorithm [WB93b, WB94a]. The algorithm calculates the maximum value of all variables $\{x_0, \ldots, x_{n-1}\}$ relative to a distinguished variable, x_0, and thus for interface timing verification problems must be run n times to calculate the full n^2 separations between n events.

McMillan & Dill's algorithm in UBC terms

Inputs: a system of m UBCs over n variables x_0 through x_{n-1}
Outputs: maximum possible value for each x_i when $x_0 = 0$

```
$x_0 \leftarrow 0$
**Forall** $x_i$
    **if** $i \neq 0$ **then** $x_i \leftarrow \infty$
**Do:**
    **For** each UBC $u_i : x_{\tau(i)} \leq \max_j(x_j + \alpha_{i,j}, \ldots)$ in sequence **do:**
        **If** $x_{\tau(i)} > \max_j(x_j + \alpha_{i,j}, \ldots)$ **then**
            $x_{\tau(i)} \leftarrow \max_j(x_j + \alpha_{i,j}, \ldots)$
    **If** $x_0 < 0$ **then**
        Report constraints inconsistent and exit algorithm
**Until** no $x_i$ changes
Report values of all $x_i$.
```

Figure 8.1: McMillan and Dill's algorithm modified to determine maximum time separations relative to a single event.

The Short-circuit algorithm	
Inputs:	a system of m UBCs over n variables x_0 through x_{n-1}
Outputs:	maximum possible value for each x_i when $x_0 = 0$

```
Initialization:
    $x_0 \leftarrow 0$
    **Forall** $x_i$ with $i \neq 0$ **do:**
        $x_i \leftarrow \mathcal{V}$
        Associate the UBC $x_i \leq x_0 + \mathcal{V}$ with $x_i$
**Do:**
    Updating Phase:
        **Do** $n$ times:
            **For** each UBC $u_i : x_{\tau(i)} \leq \max_j(x_j + \alpha_{i,j})$ in sequence **do:**
                **If** $x_{\tau(i)} > \max_j(x_j + \alpha_{i,j})$ **then**
                    **If** $\tau(i) == 0$ **then**
                        Report constraints inconsistent and exit algorithm
                    $x_{\tau(i)} \leftarrow \max_j(x_j + \alpha_{i,j})$
                    Mark $x_{\tau(i)}$ as having changed value during this phase
                    Change $x_{\tau(i)}$'s associated UBC to $u_i$
    Short-circuiting Phase:
        Perform strongly connected components analysis on graph edges
            induced by the UBCs associated with those $x_i$ who are marked
            as changing value during the most recent updating phase
        **Foreach** strongly connected component $\mathcal{C}$ **do in parallel:**
            **Foreach** $x_{\tau(i)}$ in $\mathcal{C}$ **do:**
                $y_{\tau(i)} \leftarrow \max_j(\mathcal{F}_{\mathcal{C}}(x_j) + \alpha_{i,j})$ **where**
                    $\mathcal{F}_{\mathcal{C}}(x_j) = -\infty$ if $x_j$ is in $\mathcal{C}$ and
                    $\mathcal{F}_{\mathcal{C}}(x_j) = x_j$ if $x_j$ is not in $\mathcal{C}$
            **If** all $y_i \in \mathcal{C}$ are $-\infty$ **then:**
                Report the constraints inconsistent and exit algorithm
            Among those $x_i \in \mathcal{C}$ with $y_i \neq -\infty$
                **Let** $\delta$ be the minimum value of $x_i - y_i$
            **If** $\delta > 0$ **then**
                **Forall** $x_j$ in $\mathcal{C}$ **do:** $x_j \leftarrow (x_j - \delta)$
**Until** no $x_i$ changes
Report values of all $x_i$.
```

Figure 8.2: The Short-circuit algorithm

The `ShortCircuit` algorithm consists of three different sections of code: the *Initialization* section, in which for each $i \neq 0$, the variable x_i is set to the symbolic value $\mathcal{V}$ discussed in Sections 4.3.2 and 6.4; the *Updating* phase in which each of the UBCs in the system is applied to update variable bounds where possible; and the *Short-circuiting* phase in which the effects of repeated patterns of constraint application are deduced and applied to the variables. The initialization phase is executed once at the start of the algorithm and from then on the updating and short-circuiting phases alternate. Our proofs of correctness for the `ShortCircuit` algorithm will rely upon relating the actions occurring in different executions of the updating and short-circuiting phases and so we make the following definitions.

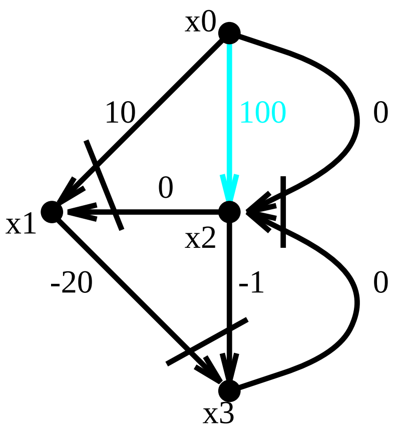
Definition 8.1 *The* **i-th occurrence** *of an updating or short-circuiting phase refers to the i-th time that phase is run. Any particular occurrence of the updating phase is subdivided into n* **rounds**, *one for each iteration within the "*`Do n times`*" loop of the updating phase.*

During the updating and short-circuiting phases, each variable x_i has a single UBC associated with it. The updating phase applies the same constraints as in the `Do`-loop of McMillan and Dill's algorithm, but whenever an variable's maximum value is decreased, the `ShortCircuit` algorithm changes the UBC associated with that variable to reflect the constraint used to decrease it. In addition, the `ShortCircuit` algorithm keeps track of which variables x_i had their values updated during the most recent occurrence of the updating phase.

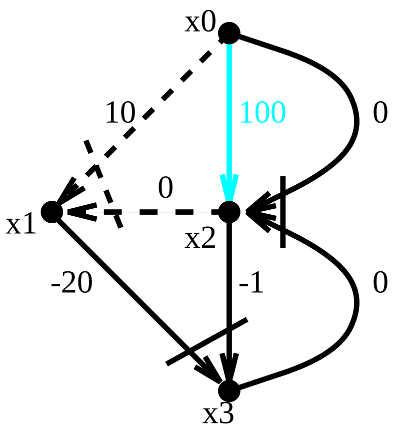
Figure 8.3 gives an example of `ShortCircuit` operating on the UBC system given at the top of the figure. After n trips through the list of UBCs, `ShortCircuit` proceeds on to the short-circuiting phase. At this point the bounds on the variables x_i are as given in the fifth data column of Figure 8.3(d). The graphical representation of the constraint set is pictured in Figure 8.3(b). As was done in Section 4.3.2, black constraint arcs represent those constraints associated with a given node, and gray constraint arcs represent all other constraints. In Figure 8.3(b), all constraints associated with nodes at the end of the first occurrence of the updating phase were used during that occurrence, and so in the first short-circuiting phase we perform the strongly connected component analysis on the black arcs in Figure 8.3(b). We note that x_1 must decrease from 98 to 10 to meet its exterior bound, while x_2 must

$$\begin{aligned}
x_1 &\leq \max(x_0 + 10, x_2) \\
x_2 &\leq \max(x_0, x_3) \\
x_2 &\leq x_0 + 100 \\
x_3 &\leq \max(x_1 - 20, x_2 - 1)
\end{aligned}$$

(a)



(b)



(c)

variable	start	$\mathcal{MD}$ rounds				short-c	$\mathcal{MD}$ rounds				short-c
x_0	0	0	0	0	0	0	0	0	0	0	0
x_1	$\mathcal{V}$	$\mathcal{V}$	100	99	98	10	10	10	10	10	10
x_2	$\mathcal{V}$	100	99	98	97	9	8	7	6	5	0
x_3	$\mathcal{V}$	99	98	97	96	8	7	6	5	4	-1

(d)

Figure 8.3: Two different short-circuits of the same safe targeting subset.

decrease from 97 to 0 to do so. Clearly x_1's decrease is the smaller and so we decrease all of x_1, x_2, and x_3 by 88.

Note that the arcs pictured in black in Figure 8.3(b) are a safe targeting subset (Definition 6.12). As we will show in Section 8.1, during each step of the `ShortCircuit` algorithm, the set of constraints associated with the variables is a safe targeting subset. For the example of Figure 8.3, the result of the first short-circuiting round is not the maximum solution to that safe targeting subset. A second pass through both phases is required even though the set of constraints associated with the variables does not change between the two occurrences of the short-circuiting phase. However, the constraint $x_1 \leq \max(x_0 + 10, x_2)$ is not applied during the second occurrence of the short-circuiting phase. When the constraint associated with a specific x_i is not used during the most recent updating phase, we represent it as in Figure 8.3(c) with dashed black arcs. The strongly connected components analysis is performed on the black arcs only with results as given in the last column of Figure 8.3(d).

8.1 Proof of Correctness for the `ShortCircuit` Algorithm

In this section we show that `ShortCircuit` and `UBCsolv` obtain the same maximum solutions for a set of variables x_i relative to a distinguished x_0. Since Theorem 6.2 has already proved `UBCsolv` correct, this will prove `ShortCircuit` correct as well. To do this we show several things:

- that the set of constraints associated with each x_i always forms a safe targeting subset,
- that the process of short-circuiting components within a safe targeting subset $\mathcal{S}$ always results in variable bounds greater than or equal to the maximum solution to $\mathcal{S}$, and
- that short-circuiting process is applied to a given targeting subset $\mathcal{S}$ at most n times.

Our arguments below depend on the UBCs having the property that for UBC

$$u_i : \mathbf{x}_{\tau(\mathbf{i})} \leq \max_j(\mathbf{x}_\mathbf{j} + a_{i,j}) \tag{8.1}$$

targeting $x_{\tau(i)}$, the value of $a_{i,\tau(i)}$ is $-\infty$. As was shown in Proposition 6.2, any system of UBCs may be changed into an equivalent set of UBCs with this property.

Proposition 8.1 *During every step of the* `ShortCircuit` *algorithm the set of UBCs associated with the variables* $\{x_0, \ldots, x_{n-1}\}$ *is a targeting subset.*

Proof: Recall from Definition 6.8 that a targeting subset of a system of UBCs over n variables is a set of $n-1$ UBCs such that all variables except x_0 are the target of a single constraint. As we begin the `ShortCircuit` algorithm, each x_i with $i \neq 0$ is associated with a single UBC, thus giving us a targeting subset. From that point on, an individual x_i's targeting UBC may change, but each x_i where $i \neq 0$ always has one. Note that we never associate a UBC with x_0 since to do so would require a UBC capable of decreasing the bound on x_0, but this causes the algorithm to terminate immediately indicating that the constraint set is inconsistent. □

We would now like to show that such a targeting subset is always safe. To do so, we first demonstrate that the following local property holds for all constraints in the targeting subsets present during the `ShortCircuit` algorithm. We begin by recalling the method for referring to a targeting subset of UBCs as consisting of $n-1$ equations

$$\mathbf{x_i} \leq \max_j(\mathbf{x_j} + (\mathcal{P}A)_{i,j}) \tag{8.2}$$

where $\mathcal{P}$ is a targeting selector for the UBC system as originally given in Definition 6.9. Throughout the rest of this chapter, we will use the bold term $\mathbf{x_i}$ in equations which describe the UBCs of $\mathcal{U}$ constraining the value of variable x_i, and the italic term x_i to represent its current value.

Lemma 8.1 *Immediately after every change in a variable's bound during the updating phase and at the end of any occurrence of the short-circuiting phase, if*

$$\mathbf{x_i} \leq \max_j(\mathbf{x_j} + (\mathcal{P}A)_{i,j}) \tag{8.3}$$

is the UBC associated with $\mathbf{x_i}$*, then*

$$x_i \geq x_j + (\mathcal{P}A)_{i,j} \tag{8.4}$$

for all j*.*

Proof: It should be obvious that this is the case at the end of the initialization phase since at that point each x_i with $i \neq 0$ is associated with the constraint $\mathbf{x_i} \leq \mathbf{x_0} + \mathcal{V}$ and $x_0 = 0$ while $x_i = \mathcal{V}$ for $i \neq 0$. The remainder of the proof is then an induction argument in which it is argued that if the property holds just before any of the operations which change some x_i's value, it must also hold immediately after the change.

Suppose the algorithm is in the updating phase and at the moment Lemma 8.1 holds, and some variable x_i is about to have its bound decreased. Before that decrease occurs we have

$$x_i \geq x_j + (\mathcal{P}A)_{i,j} \tag{8.5}$$

for all j. When x_i's value changes it becomes the new value x'_i with

$$x'_i < x_i, \tag{8.6}$$

and the targeting selector $\mathcal{P}$ changes to $\mathcal{P}'$ to reflect the possible change in the constraint targeting x_i. Immediately after that update, we have

$$x'_i = \max_j (x_j + (\mathcal{P}'A)_{i,j}), \tag{8.7}$$

which satisfies

$$x'_i \geq x_j + (\mathcal{P}'A)_{i,j}, \tag{8.8}$$

for all j. The question then remains whether the decrease in from x_i to x_i may invalidate the property for some other UBC $\mathbf{x_k} \leq \max_j(\mathbf{x_j} + (\mathcal{P}'A)_{k,j})$ with $k \neq i$. If this is the case, then we must have that

$$x_k \geq x_i + (\mathcal{P}A)_{k,i}, \tag{8.9}$$

while

$$x_k < x'_i + (\mathcal{P}'A)_{k,i}. \tag{8.10}$$

Now, since in the change from $\mathcal{P}$ to $\mathcal{P}'$, the UBC targeting x_k did not change, Equation 8.10 implies that

$$x_k < x'_i + (\mathcal{P}A)_{k,i}. \tag{8.11}$$

Combining Equations 8.9 and 8.11, we get

$$x'_i + (\mathcal{P}A)_{k,i} > x_i + (\mathcal{P}A)_{k,i}, \tag{8.12}$$

which is impossible since it directly contradicts Equation 8.6.

For value changes during short-circuiting we note that since the set of associated UBCs do not change we need show that for every UBC $\mathbf{x_i} \leq \max_j(\mathbf{x_j} + (\mathcal{P}A)_{i,j})$, with

$$x_i \geq x_j + (\mathcal{P}A)_{i,j} \tag{8.13}$$

holding before the short-circuiting steps it follows that

$$x'_i \geq x'_j + (\mathcal{P}A)_{i,j}, \tag{8.14}$$

where x'_i and x'_j are the new values of the variables after short-circuiting. If x_i and x_j are both in the same strongly connected component, then since both decrease by the same amount, Equation 8.13 implies that Equation 8.14 holds. If x_i and x_j are not in the same strongly connected component, then the arc from x_j to x_i is exterior to the component x_i is in, and thus after short-circuiting

$$x'_i \geq x_j + (\mathcal{P}A)_{i,j}, \tag{8.15}$$

since by the way the δ values are calculated by `ShortCircuit`, no node with exterior arc decreases below the value provided by that arc. Since $x'_j < x_j$, Equation 8.15 yields Equation 8.14. □

We are now ready to show that the local property of Lemma 8.1 implies that the targeting subsets present at every step of the `ShortCircuit` algorithm are indeed **safe targeting subsets**.

Lemma 8.2 *At every step of the* `ShortCircuit` *algorithm, the set of UBCs associated with the variables x_i is a safe targeting subset.*

Proof: By Proposition 8.1, we know that such a UBC subset is a targeting subset. Consider any cycle in the graph induced by such a targeting subset. By Lemma 8.1, we know that in this graph, for all arcs from x_j to x_i with arc weight $(\mathcal{P}A)_{i,j}$, it is the case that $x_i \geq x_j + (\mathcal{P}A)_{i,j}$. Assume without loss of generality that the cycle in question consists of the nodes $\{x_1, \ldots, x_k\}$ with arcs of weight α_i from x_i to $x_{(i \bmod k)+1}$. We then have by Lemma 8.1 that

$$x_{(i \bmod k)+1} \geq x_i + \alpha_i \tag{8.16}$$

and that the total cycle weight is

$$\sum_{i=1}^{k} \alpha_i. \tag{8.17}$$

Since the graph induced by the safe targeting subset present at initialization contains no cycles, it must be the case that one arc along this current cycle was the most recently introduced into the graph. Suppose that was the arc from x_k to x_1 and that at that time the value of x_1 was updated from x_1 to x_1'. We know then that

$$x_1' < x_1 \tag{8.18}$$

and that both

$$x_k \geq x_1 + \sum_{i=1}^{k-1} \alpha_i \tag{8.19}$$

and

$$x_1' \geq x_k + \alpha_k. \tag{8.20}$$

By Equations 8.19 and 8.20 we therefore have

$$x_1' \geq x_1 + \sum_{i=1}^{k} \alpha_i. \tag{8.21}$$

Note that the only way Equation 8.21 can be consistent with Equation 8.18 is if

$$0 > \sum_{i=1}^{k} \alpha_i, \tag{8.22}$$

and therefore the weight of any cycle must be negative. □

Lemma 8.3 *If $\mathcal{S}$ is the safe targeting subset associated with a given step of* `ShortCircuit` *then the bounds on the values of the variables x_i during that step are not less than the maximum solution to $\mathcal{S}$ when $x_0 = 0$.*

Proof: By Lemma 8.1 we know that at every step of the `ShortCircuit` algorithm $x_0 = 0$, and the equations

$$x_i \geq \max_j (x_j + (\mathcal{P}A)_{i,j}) \tag{8.23}$$

hold for every variable x_i. In comparison, the maximum solution to $\mathcal{S}$ solves the $n-1$ equations

$$x_i = \max_j (x_j + (\mathcal{P}A)_{i,j}) \tag{8.24}$$

for $x_0 = 0$. Thus we need only prove that in any solution to the system of Equation 8.23, the variable values are no less than those of the solution to Equation 8.24.

Let m_i be the solution for each variable x_i to the system of Equation 8.24, which is unique by Theorem 6.1, and let l_i be a solution for each variable x_i to the system of Equation 8.23. If the lemma is not true then there must be some l_i such that

$$m_i > l_i. \tag{8.25}$$

Now by Equation 8.24, the total ordering property and Proposition 6.2, we know that there exists at least one $j \neq i$ such that

$$m_i = m_j + (\mathcal{P}A)_{i,j}. \tag{8.26}$$

Furthermore, by Equation 8.23 we also know that

$$l_i \geq l_j + (\mathcal{P}A)_{i,j}. \tag{8.27}$$

Together, Equations 8.25, 8.26 and 8.27 give us $m_j + (\mathcal{P}A)_{i,j} > l_j + (\mathcal{P}A)_{i,j}$, and thus

$$m_j > l_j \tag{8.28}$$

We may then apply the same argument to find a k with

$$m_j = m_k + (\mathcal{P}A)_{j,k} \tag{8.29}$$

and

$$m_k > l_k. \tag{8.30}$$

By repeatedly applying this argument we find that one of two things may occur within n applications of this argument: either we discover that

$$m_0 > l_0, \tag{8.31}$$

or we find a series of variables (without loss of generality) $\{x_1, x_2, \ldots, x_k\}$ such that for $1 \leq i \leq k$

$$m_i = m_{1+(i \bmod k)} + (\mathcal{P}A)_{i,[1+(i \bmod k)]} \tag{8.32}$$

and

$$m_i > l_i. \tag{8.33}$$

In the first case, we then have that $l_0 < 0$. However, the ShortCircuit algorithm never decreases the value of x_0, so this cannot happen.

In the second case, we use Equation 8.32 as in Lemma 8.2 above resulting in the equation

$$m_1 = m_1 + \sum_{i=1}^{k} (\mathcal{P}A)_{i,[1+(i \bmod k)]}. \tag{8.34}$$

Since the corresponding safe targeting subset has only negative cycles, this is equivalent to

$$m_1 = m_1 + \omega \tag{8.35}$$

where $\omega < 0$. This forces $m_1 = -\infty$, which is impossible since it violates Equation 8.25. □

We have now shown that ShortCircuit never undercuts the values obtained by UBCsolv for a consistent system, but it remains to answer:

- whether ShortCircuit converges in a finite number of steps,
- whether ShortCircuit and UBCsolv converge to the same bounds, and
- whether ShortCircuit and UBCsolv declare the same systems inconsistent.

We answer the first question by demonstrating that while the safe targeting subset associated with each iteration of short-circuiting may occur more than once, the strongly connected components that are examined are always properly contained within the components short-circuited the last time that particular safe targeting subset was short-circuited. Figure 8.3 includes an example of this phenomenon. The next few lemmas will demonstrate that this is always the case, and thus the same safe targeting subset may be short-circuited at most n times.

Lemma 8.4 *At any point in the* ShortCircuit *algorithm, if all nodes in any strongly connected component, $\mathcal{C}$, contained in the current safe targeting subset, $\mathcal{S}$ satisfy their target UBC,*

$$\mathbf{x_i} \leq \max_j(\mathbf{x_j} + (\mathcal{P}A)_{i,j}) \tag{8.36}$$

then at least one node x_i has value

$$x_i = x_j + (\mathcal{P}A)_{i,j} \tag{8.37}$$

where x_j is not a member of $\mathcal{C}$.

Proof: By Lemma 8.1 we know that for every x_i

$$x_i \geq x_j + (\mathcal{P}A)_{i,j} \tag{8.38}$$

for all j, and therefore Equation 8.36 indicates that for every x_i in $\mathcal{C}$, Equation 8.37 must hold for some x_j which may or may not be in $\mathcal{C}$.

If Equation 8.37 applies for an x_j not in $\mathcal{C}$, we have proved the lemma. Otherwise, Equation 8.37 applies only to pairs x_i and x_j in the same strongly connected component then the arcs corresponding to those pairs must form one or more directed cycles within the component. Applying Equation 8.37 for all nodes on one such cycle gives us an argument similar to that in Lemma 8.3 that

$$x_k = x_k + \omega \tag{8.39}$$

for negative ω as in Equation 8.35. This indicates that $x_k = -\infty$ for some node x_k in the component. However, this satisfies Equation 8.37 since $-\infty \leq x_j + (\mathcal{P}A)_{i,j})$ for all x_j whether it is in the same component or not, and any subgraph of a safe targeting subset includes nodes in at least two different strongly connected components since no arc points towards x_0. □

Definition 8.2 *At the termination of any occurrence of the updating phase, the nodes of the induced graph are one of three types:*

- *nodes whose values are equal to the maximum solution to the safe targeting subset, $\mathcal{S}$, present at the end of this occurrence of the updating phase, called* **valid** *nodes, and denoted in graphs with a □,*

- *nodes that are not* **valid** *nodes and whose values changed during the current iteration of the updating phase, called* **changed** *nodes, and denoted in graphs with a •, and*

- *nodes that are not* **valid** *nodes and whose values have not changed during the current iteration of the updating phase, called* **unchanged** *nodes, and denoted in graphs with a ◦,*

Figure 8.4 gives the diagrams of Figure 8.3 using these node symbols.

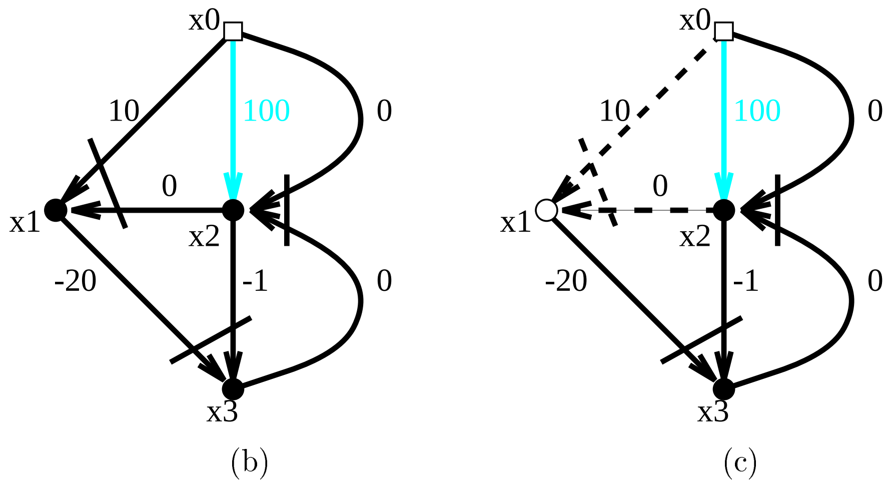


Figure 8.4: The diagrams of Figure 8.3 using the symbols of Definition 8.2.

Lemma 8.5 *If $\mathcal{S}$ is a safe targeting subset present at the end of an updating phase then any variable bound x_i that did not change in that updating phase is at its maximum solution value in $\mathcal{S}$.*

Proof: We consider the graph corresponding to the safe targeting subset at the end of the updating phase, and the types of its nodes according to Definition 8.2, and will prove that unchanged nodes cannot exist in that graph. We examine the placement of unchanged nodes in the graph corresponding to the full safe targeting subset, minus those arcs used to short-circuit at a particular phase. That is, we examine the subgraph made up of all node types but only dashed lines as in Figure 8.4. In such a graph of dashed arcs we find any topologically first strongly connected component. Since x_0 is never an unchanged node, a topologically first component must either be a single node which has no unchanged nodes as parents, or a strongly connected component of size 2 or more unchanged nodes with no unchanged nodes entering it, since otherwise it is not topologically first.

Figure 8.5(a) gives a diagram of the first case. Arcs are shown dashed since they have not been used during the most recent updating phase. Suppose our designated unchanged node is x_i, and has associated UBC

$$\mathbf{x_i} \leq \max_j(\mathbf{x_j} + (\mathcal{P}A)_{i,j}). \tag{8.40}$$

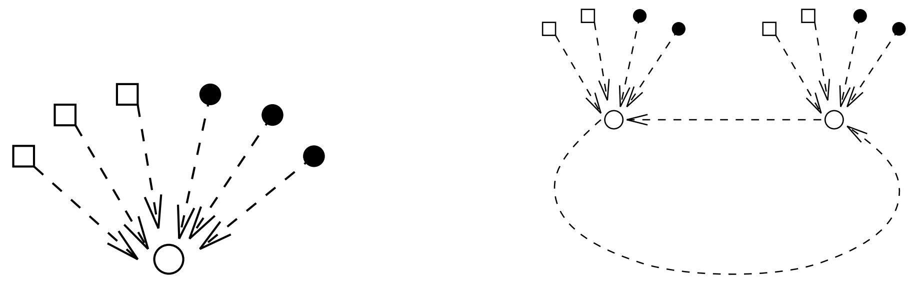


Figure 8.5: The two cases in the proof of Lemma 8.5.

We may re-write this UBC instead as

$$x_i \le \max[\max_p(x_p + (\mathcal{P}A)_{i,p}), \max_q(x_q + (\mathcal{P}A)_{i,q})] \tag{8.41}$$

where among those x_j with terms $(\mathcal{P}A)_{i,j} \neq -\infty$, p ranges over variables x_p which have valid values and q ranges over variables x_q which have changed values. At the end of the updating phase, we claim it must be the case that

$$x_i > \max(\max_p[x_p + (\mathcal{P}A)_{i,p}], \max_q[x_q + (\mathcal{P}A)_{i,q}]). \tag{8.42}$$

To see this, note that

$$x_i > \max_p(x_p + (\mathcal{P}A)_{i,p}), \tag{8.43}$$

else x_i would have to be a node of valid type, and

$$x_i > \max_q(x_q + (\mathcal{P}A)_{i,q}), \tag{8.44}$$

since the values of all such nodes x_q have changed since the last time x_i changed value. Furthermore, at least one of the nodes x_q changed *for the first time* during the nth and final round of the updating phase, since otherwise, applying the constraint in Equation 8.40 would have been updated x_i during updating. Now, applying the same argument, if that same x_q didn't change until the nth round of the current updating phase, then it must have depended on some $x_{q'}$ which changed for the first time no earlier than the $n-1$st round. We continue this argument and find that there must be n distinct nodes, none of them x_i, each of which changed value for the first time in this updating phase no earlier than the k-th round for $1 \leq k \leq n$. That, however, is impossible since that would indicate our system has at least $n+1$ distinct nodes, and we have precisely n.

Figure 8.5(b) gives a diagram of the second case. Here we argue that since the values of the nodes in the strongly connected component cannot all satisfy their associated constraints without meeting the bound of some exterior arc (Lemma 8.4), either one of the nodes in that component has changed during updating or it must have bound equal to that provided its associated set of exterior arcs. By the same argument as above, all such exterior bounds either must provide a valid bound to a node in the component or must change during the first $n-1$ rounds within the current updating phase. If the component has no exterior arcs, it must contain at least one node with value $-\infty$, but `ShortCircuit` never sets any x_i to $-\infty$. Therefore some unchanged arc must change in the nth round or sooner in order to satisfy Lemma 8.4. □

Lemma 8.6 *The* `ShortCircuit` *algorithm may short-circuit using the same safe targeting subset, $\mathcal{S}$, a maximum of n times.*

Proof: Similar to Lemma 8.5, we will now show that in every instance where strongly connected components of a given safe targeting subset, $\mathcal{S}$, are short-circuited, at least one node that was not previously at its correct maximum value for that safe targeting subset achieves that maximum value. Since there are only n nodes in the graph, and since Lemma 8.2 tells us that the values of nodes are never lower than their upper bounds according to the current safe targeting subset this can happen no more than n times per safe targeting subset.

We examine the placement of changed nodes in the graph of the safe targeting subset. This time we find a topologically first changed node or strongly connected component of changed nodes among the strongly connected components examined by `ShortCircuit`.

Clearly the situation in Figure 8.6(a) cannot occur since in the first round of the preceding updating phase, the changed node must have reached its valid value with one application of its target constraint. For the case of Figure 8.6(b) we simply note

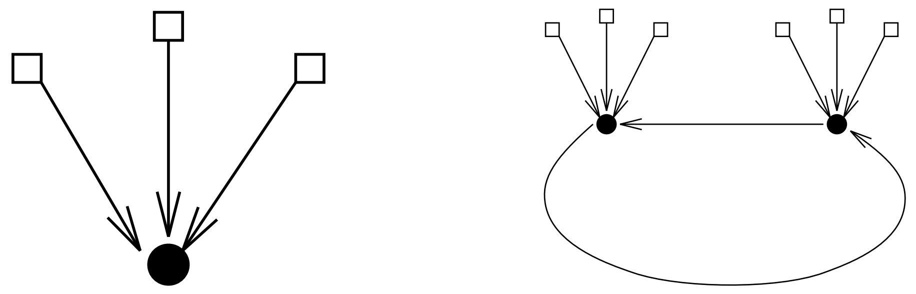

(a) (b)

Figure 8.6: The two cases in the proof of Lemma 8.6.

that the short-circuiting process always brings some node in the component to meet the value provided by its exterior arcs. Whichever node that is, its targeting UBC can be written

$$x_i \leq \max[\max_p(x_p + (\mathcal{P}A)_{i,p}), \max_q(x_q + (\mathcal{P}A)_{i,q})] \tag{8.45}$$

where the x_p range over the nodes with valid values and the x_q range over the nodes with changed values. After short-circuiting, some x_i therefore has value

$$\max_p(x_p + (\mathcal{P}A)_{i,p}) \tag{8.46}$$

which must be the valid value of x_i. Hence, at least one changed node becomes valid for a given safe targeting subset each time that subset appears in the short-circuiting phase. □

Lemma 8.7 *The* `ShortCircuit` *algorithm converges in a finite number of steps through its outer* `Do`*-loop.*

Proof: This is a direct result of the fact that there are only a finite number of safe targeting subsets for a given system of UBCs (Lemma 6.10), a single safe targeting subset is associated with the `ShortCircuit` algorithm each time it enters the short-circuiting phase, and the fact that the `ShortCircuit` algorithm enters the short-circuiting phase at most n times for each such safe targeting subset (Lemma 8.6). □

Combined with the upper bound on the number of possible targeting subsets given in Lemma 6.10 this gives us a bound of

$$O(n \cdot nt \cdot (m+1)^{n-1}) \tag{8.47}$$

on the time for `ShortCircuit` to converge for a system of m UBCs consisting of t terms over n variables. Unlike the bound on McMillan and Dill's algorithm, this bound is expressible independently of the values of the coefficients of the constraint terms. A rather smaller bound for `ShortCircuit` is conjectured at the end of this section.

Lemma 8.8 *The* `ShortCircuit` *and* `UBCsolv` *algorithms declare the same UBC systems inconsistent.*

Proof: By Lemmas 8.7 and 6.10 we know that the `ShortCircuit` algorithm cannot find consistent a set of bounds which `UBCsolv` declares inconsistent. To see this, note that if `ShortCircuit` terminates with a solution it declares consistent then all UBCs in the system are satisfied by that solution and `UBCsolv` cannot find the system inconsistent. Suppose instead that the `ShortCircuit` algorithm finds a UBC system inconsistent while `UBCsolv` does not. `ShortCircuit` algorithm may declare a UBC system inconsistent in one of two ways:

- it finds a UBC which forces the value of x_0 below 0, or
- among the strongly connected components found during the short-circuiting phase, there is at least one which has no recently-used arcs entering it.

In the first case we consider the bounds l_i that the `ShortCircuit` algorithm has for each of the variables x_i just before the constraint reducing x_0's bound is found. By Lemma 8.3, `UBCsolv`'s solutions, m_i, must be no larger, and therefore the constraint which decreases x_0 below when applied to the values l_i in `ShortCircuit` must certainly decrease x_0 when applied to the values m_i by `UBCsolv`.

In the second case, it must be true that in the safe targeting subset containing that component there is no path from x_0 to all x_i in the component. To see this, we note that the components to be short-circuited are determined based on whether the nodes in them have been updated in the most recent updating phase. Among all arcs in the safe targeting subset, all of those pointing to one of those recently updated nodes are included in the components and thus all paths to nodes in the component originate in the component. By Lemma 6.8 this indicates that the UBC system is inconsistent. □

Theorem 8.1 *The* `ShortCircuit` *algorithm converges upon the correct maximum solution to a UBC system $\mathcal{U}$ within a finite number of steps.*

Proof: This is easily proved by showing that the `ShortCircuit` algorithm converges upon the same answer as `UBCsolv` in a finite number of steps. From Lemmas 8.7 and 8.3, we know that `ShortCircuit` converges within a finite amount of time to values which are no smaller than those obtained by `UBCsolv`. However, `ShortCircuit` may only terminate under two conditions: the UBC system is inconsistent, or no UBCs may be applied to reduce any variable's bound. By Lemma 8.8, we know that whenever `ShortCircuit` declares a UBC system inconsistent, `UBCsolv` will do so as well and vice versa. However, if the `ShortCircuit` algorithm cannot apply any more constraints, then the values it converges upon must satisfy the UBC system. None of the variables may have value larger than in `UBCsolv`'s solution, since `UBCsolv` has been proven to find the largest solution to the UBC system (Theorem 6.2). □

Lemma 8.9 *Each round of the* `ShortCircuit` *algorithm requires time at most $O(nt)$ where n is the number of variables in the system and t is the number of terms not equal to $-\infty$ among the right hand sides of the UBCs.*

Proof: During the updating phase, each of the n passes through the constraints requires t additions and t max comparisons for a total of $2nt$ operations. The strongly connected components analysis takes time $O(\max(n, t'))$ ([Baa88], Chapter 4.6) where t' is the maximum number of edges in the graph induced by a targeting subset and is therefore no larger than n^2. □

Conjecture 8.1 *The worst-case number of occurrences of the updating and short-circuiting phases of* `ShortCircuit` *for UBC system of m constraints with t terms not equal to $-\infty$ in the constraints' right hand sides over n variables is between n and t.*

This then would give a worst case time to convergence between $O(n^2t)$ and $O(nt^2)$.

8.2 Practical Advantages of `ShortCircuit` over `UBCsolv`

Examples such as that in Figure 8.3 above in which multiple occurrences of the short-circuiting phase operate on the same safe targeting subsets may initially lead one to believe that `ShortCircuit` is less practical than `UBCsolv`. However there are two ways in which `ShortCircuit` has an advantage over `UBCsolv`.

The first is that the data structures required for `ShortCircuit` are smaller than those for `UBCsolv`. Both require some representation of the individual UBCs, but the closure technique requires that we calculate an $O(n^2)$ size closure matrix, while the `ShortCircuit` algorithm requires a stack of size $O(n)$ and $O(n)$ additional variables for the strongly connected components algorithm, plus for each x_i, variables containing the current value, associated constraint, and a flag indicating if the corresponding variable's bound was updated during the current updating phase.

The second advantage is that the the time to perform one pass through the `ShortCircuit` algorithm is $O(nt)$ as opposed `UBCsolv`'s $O(t + n^3)$. Although it is not difficult to come up with problems in which the `ShortCircuit` algorithm is more time-consuming, these do not correspond to practical timing problems.

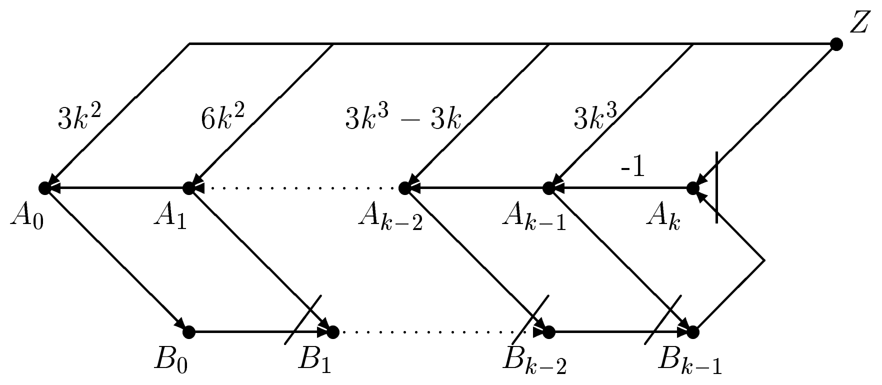
8.3 A Worst-Case Example

Figure 8.7 gives an example system with $O(k)$ UBCs which takes `UBCsolv` $O(k^4)$ and `ShortCircuit` $O(n^3)$ time to solve for all separations relative to the single variable Z.

To analyze `UBCsolv`'s progress, we note that each time a new safe targeting subset $\mathcal{S}'$ is chosen from the previous $\mathcal{S}$, only one UBC targeting a single variable is changed. Thus we may bound `UBCsolv`'s time to convergence by counting the number of times each UBC may be exchanged into or out of the new safe targeting subset.

- the k constraints of the form $A_i \leq 3k^2(i+1) + Z$ may only be switched into the constraint subset once since the value they provide to their target node is relative to the value of Z, which does not change, and thus they contribute at most k rounds to the constraint set updating process;

- similarly, the $k+1$ non-$\mathcal{V}$ constraints targeting A_k and the B_i terms may only be switched into the constraint subset once since each of these nodes has only one



enumerator	max-plus equation			regular equation		
for $i = 0$ to k	A_i	$\leq$	$\mathcal{V} \otimes Z$	A_i	$\leq$	$\mathcal{V} + Z$
for $i = 0$ to $k-1$	B_i	$\leq$	$\mathcal{V} \otimes Z$	B_i	$\leq$	$\mathcal{V} + Z$
for $i = k-2$ down to 0	A_i	$\leq$	A_{i+1}	A_i	$\leq$	A_{i+1}
for $i = k-1$ downto 0	A_i	$\leq$	$(1 \otimes i)^{3k} \otimes Z$	A_i	$\leq$	$3k^2(i+1) + Z$
for $i = k-1$ downto 1	B_i	$\leq$	$B_{i-1} \oplus A_i$	B_i	$\leq$	$\max(B_{i-1}, A_i)$
	B_0	$\leq$	A_0	B_0	$\leq$	A_0
	A_{k-1}	$\leq$	$-1 \otimes A_k$	A_{k-1}	$\leq$	$-1 + A_k$
	A_k	$\leq$	$B_{k-1} \oplus Z$	A_k	$\leq$	$\max(B_{k-1}, Z)$

Figure 8.7: Pathological example of size $O(k)$. Arcs without labels should be considered to have scalar offsets of 0.

non-$\mathcal{V}$ constraint targeting it, and when a $\mathcal{V}$-constraint is swapped out the value of the node it targets becomes less than $\mathcal{V}$, thus preventing the $\mathcal{V}$-constraint from being swapped in again; and

- the $k-1$ constraints of the form $A_i \leq A_{i+1}$ plus the constraint $A_{k-1} \leq A_k - 1$ may only be switched out once by the $A_i \leq 3k(i+1) + Z$ constraints, and then may switch in only one more time, and hence they contribute at most $2k$ rounds to the constraint set updating process.

Thus `UBCsolv` requires at most $k + (k+1) + 2k = 4k + 1$ rounds to converge, and a total time of $O(k^4)$ to solve the system. The constraint ordering given in the table at the bottom of Figure 8.7 results in time $\Theta(k^4)$ to convergence.

For this example, the `ShortCircuit` algorithm is somewhat quicker than `UBCsolv` requiring only time $O(k^3)$ to complete.

For this same example, the time complexity of McMillan and Dill's algorithm ([MD92], in Appendix A) is quite different. Their algorithm begins by filling the entries of an $n \times n$ matrix, S, such that $S_{i,j}$ is the current upper bound on the maximum value of $x_i - x_j$. The algorithm begins by initializing the values of all $S_{i,i}$ to 0 and all other $S_{i,j}$ to ∞. Constraints of the form $x_i \leq x_j + \alpha$ (i.e. those containing no max term) are entered directly into the matrix as $S_{i,j} = \alpha$. The algorithm then proceeds to loop through the matrix entries updating

$$S_{i,j} \leftarrow S_{i,k} + S_{k,j} \tag{8.48}$$

whenever the quantity on the right hand side of the "$\leftarrow$" is smaller than the current value of $S_{i,j}$. Constraints containing a max term cannot be directly entered into the matrix, but their effect on the values $S_{i,j}$ is entered into the matrix at regular intervals.

When the $\mathcal{MD}$ algorithm is applied to the constraints of Figure 8.7, it quickly discovers that the maximum value relative to A_k of each of the other nodes A_i and B_i is -1, and the max constraints targeting the B_i are no longer consulted during the matrix updating phase. However, when this happens, the value of A_k relative to Z is still nearly $3k^3$, and can only be reduced by repeated application of the constraint $A_k \leq \max(Z, B_{k-1})$ to the values of Z and B_{k-1} relative to Z. With each such application, and the resulting updates in the matrix, the value of all nodes (except Z) relative to Z declines by 1, thus requiring a total time of $O(k^6)$ to discover all n^2 time separations in the system — $O(k^3)$ time to update the matrix times $O(k^3)$ matrix updates during which the value of A_k relative to Z declines by only 1. Note, however that any increase in the coefficient of the constraint $A_{k-1} \leq 3k^3 + Z$ results in an increase of the running time, and as discussed in Section 4.3.1, were the constraints $A_i \leq 3k^2(i+1) + Z$ removed, their algorithm would erroneously state that the maximum time separation of all other nodes relative to Z was ∞.

8.3.1 *Comparison of* `ShortCircuit` *with* `MaxSeparation`

As first mentioned in Section 4.3.3, since the introduction of `ShortCircuit`, Yen et. al. have proposed the `MaxSeparation` algorithm. As with `ShortCircuit`, `MaxSeparation` is an iterative technique consisting of repeated relaxation of constraints. The time to perform one relaxation round of `MaxSeparation` is

$$O(E + V \log V), \tag{8.49}$$

when run on an acyclic graph with V nodes and E constraints. The authors conjecture a total running time of

$$O(VE + V^2 \log V), \tag{8.50}$$

based on a maximum of $V + 1$ required relaxation rounds. The time for one `ShortCircuit`, iteration, on the other hand is

$$O(nt), \tag{8.51}$$

with a conjectured total run time falling between

$$O(n^2 t) \tag{8.52}$$

and

$$O(nt^2) \tag{8.53}$$

for a system of UBCs containing t non-ϵ terms over n variables. `MaxSeparation`'s constraint form only allows one **max** type constraint to bound each variable from above and thus there are UBC systems in which each of n variables is targeted by $O(n)$ constraints with $O(1)$ constraint terms each for a total of n nodes and $O(n^2)$ terms while the `MaxSeparation` algorithm must have the problem expressed with a total of $O(n^2)$ edges and $O(n^2)$ nodes — each UBC with more than one non-ϵ term is given its own node and all such nodes corresponding to constraints with the same original target are set equal with a set of $O(n)$ linear constraints with bounds $[0, 0]$. However, for practical problems, the values of n and V and t and E for the two should be comparable. Should a proof of correctness for `MaxSeparation` be produced, it may very well become the technique of choice for interface timing verification.

8.4 Run-time Results

We have implemented the algorithm and run both practical examples [Mye93, MD92] and randomly generated larger examples built to look like practical examples. In all cases no more than three short circuiting phases were required to find maximum skews relative to a single event. Running times were on the order of 20 seconds on a DEC station 5000 to find all n^2 maximum skews for a dense constraint graph with 80 nodes.

Chapter 9

TOWARDS DEVELOPING PRACTICAL TIMING SYNTHESIS PROCEDURES

With the exception of the examples of timing synthesis problems given in Section 2.5, this work has concentrated on problems of timing verification rather than timing synthesis. It is natural, however, that whenever there is a need for the verification of the correct temporal behavior of a system, there must first have been the construction of that system. At the time a system is constructed, whether built from scratch or assembled from modules, decisions are made about the functional and temporal dependencies of its sub-parts. Once functional dependencies are assigned there may still be some freedom in how quickly each sub-system completes its given task. Allotting time to each of these sub-systems is timing synthesis. When the system under construction has several input and output events with timing constraints on their relative occurrence times, timing synthesis may not be trivial.

With the exception of the review of Borriello's work found in the next section, the rest of this chapter will concern itself with timing synthesis problems for which a suitable **synthesis skeleton**, or partial ordering of activities has been provided. This skeleton removes all information about the functional structure of the system and relates events through simple timing relationships. Figures 2.12 and 2.13 give different skeletons for the same problem, one with synthesis variables and one with delays already chosen. In any skeleton, the events which will be related to each other are already determined, but the coefficient terms of those constraints are not yet set. In general, the problem of determining a suitable synthesis skeleton is difficult and requires knowledge of the functional relationships among events as well as their temporal relationships. We note that while we require a skeleton for our problem, the process of creating synthesis skeletons might very well utilize a skeleton-dependent timing synthesis tool to determine which skeletons and sub-skeletons are feasible.

9.1 Borriello's Interface Transducer

In this section we review the `Janus` interface transducer tool of Borriello's Ph.D. thesis [Bor88]. Of key interest here is the `Suture` algorithm which given a timing diagram representing an interface produces the logic that realizes it. `Suture` performs a breadth-first search of the **event graph** — the graph-theoretic realization of the **formalized timing diagram** introduced in Section 4.1 — using a template-matching procedure to determine the logic necessary to generate all required interface events. In this case, the logic generated determines the synthesis skeleton as the timing synthesis decisions are made. In order to assure that all minimum timing constraints from one event to a succeeding event are met, the algorithm may synthesize the logic for the following event to "wait" for an appropriately delayed signal from the preceding event. Following this restriction, events are generated as soon as possible to encourage the construction of an interface capable of handling higher bandwidth. However, generating an event at its earliest allowable time relative to its predecessors may cause a maximum constraint to a succeeding event to be violated. The priority queue based all pairs shortest paths timing verification algorithm [LW83, BN86] first mentioned in Section 4.1 is used to determine more conservative minimum time separations for all pairs of events. This method applies because the timing constraints present are all treated as **guarantees** and **requirements**. As we will see in Section 9.5, this approach is not tractable for all systems that include **delay** relationships among their events.

9.2 A Taxonomy for Interface Synthesis

Although delays, guarantees, and requirements as given in Chapter 3 represent the timing properties of interface hardware, in practice, the correct temporal synthesis of interface logic is better served by a more refined set of primitives. We partition the delays, guarantees, and requirements encountered during the synthesis procedure into the following three orthogonal categories:

Environment vs. Synthesis Object In general, we divide the problem into the environment — those aspects of timing behavior which are provided to us — and the synthesis object, which we are designing.

SPECIFICATION SEMANTICS	CONSTRAINABLE	UNCONSTRAINABLE
PROPAGATION DELAYS:	*incompletely synthesized activities*	*fully synthesized activities*
GUARANTEES:	*modifyable environment*	*behavior of environment*
REQUIREMENTS:	*all performance constraints*	*do not exist*

Figure 9.1: Taxonomy of interface constraint types.

Propagation Delays vs. Timing Constraints: Both circuit and environment may include structural timing information in the form of propagation delays. The environment may include timing requirements which indicate the allowable time separations of inputs to the environment and timing guarantees which summarize its temporal behavior. Section 3.2 discusses these divisions more thoroughly.

Constrainable vs. Unconstrainable ranges: Constrainable delays and performance measures indicate time ranges which may be further constricted, as needed, to create a consistent circuit-environment combination; unconstrainable ranges represent elements for which the circuit must function correctly for arbitrary delay behavior anywhere within the given range.

Figure 9.1 gives a summary of these categories, which are here more completely described from top to bottom.

- **Constrainable Propagation Delays** represent propagation delays which may be adjusted. This may indicate a component which is not completely synthesized, as in the case of interface circuitry, or an environment whose timing behavior can be modified, whether it is done through an adjustment of the part or by selecting one from among a family of parts.

- **Unconstrainable Propagation Delays** represent the unalterable causal structure of a device.

- **Constrainable Timing Guarantees** are similar to constrainable propagation delays, but in this case no explicit structural information is provided.

- **Unconstrainable Timing Guarantees** represent components whose timing behavior cannot be adjusted and for which no explicit structural information is provided.

- **Constrainable Timing Requirements** represent all timing requirements since performance requirements may always be over-met.
- **Unconstrainable Timing Requirements** do not exist.

Under this taxonomy, the verification problem consists of checking that a fully specified system (no portions still represented with constrainable propagation delays or constrainable guarantees) meets all of its performance requirements, and the synthesis problem consists of constraining all constrainable propagation delays and constrainable guarantees until they, in conjunction with their unconstrainable counterparts meet all of the timing requirements.

9.3 Max-Plus Synthesis Problems

The key feature of any timing synthesis problem is that the timing behavior of the synthesized system must fall within a given set of constraints. We can demonstrate that this is the case by showing that the space of the solution is contained within the space of the constraints. The definition of containment below follows the notation of Gahlinger [Gah90].

Definition 9.1 *Let $\mathcal{C}$ be a set of UBCs as given in Definition 6.4, then $\mathcal{S}(\mathcal{C})$ is the* **space of** *$\mathcal{C}$, and is defined as all vectors x over $\mathcal{X}$ such that x satisfies all constraints in $\mathcal{C}$ simultaneously. For any such x, we say that $x \in \mathcal{S}(\mathcal{C})$, and that*

$$[x \in \mathcal{S}(\mathcal{C}) \rightarrow x \in \mathcal{S}(\mathcal{C}')] \quad \Leftrightarrow \quad \mathcal{S}(\mathcal{C}) \subseteq \mathcal{S}(\mathcal{C}') \tag{9.1}$$

Using this notation, we can define the generic max-plus synthesis problems as follows.

Definition 9.2 *Given two sets of max plus constraints, the given $\mathcal{G}$, and the target, $\mathcal{T}$, the* **generic max-plus synthesis problem** *asks if it is possible to find a new set of max-plus equations, $\mathcal{C}$ such that*

$$\emptyset \neq \mathcal{G} \cap \mathcal{C} \subseteq \mathcal{T} \tag{9.2}$$

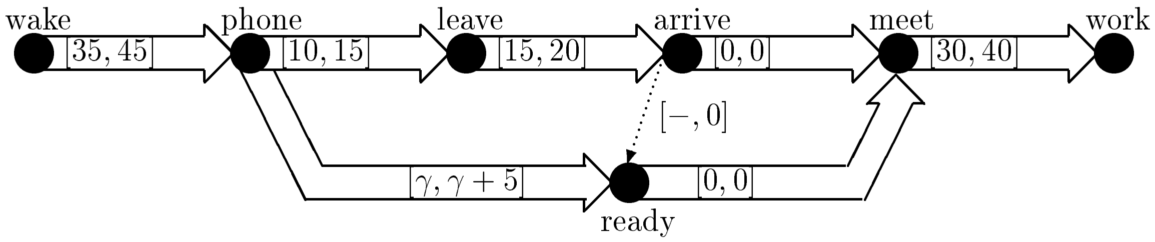


Figure 9.2: IBDS-synthesis diagram for the case in which Ernie must be ready when Bert arrives. The dotted arc indicates a constraint which should be satisfied by the assignment of a value to γ.

Naturally, if $\mathcal{G} \cap \mathcal{T} \neq \emptyset$, then the easiest answer to this question is $\mathcal{C} = \mathcal{T}$. However, problems based on practical situations generally have some tight restrictions on the form of $\mathcal{C}$, and sometimes on $\mathcal{G}$ and $\mathcal{T}$ as well. A few such problems will be given in the following sections.

9.4 I-BDS Synthesis Problems

Recall from Chapter 2 the problem of choosing a revised carpooling arrangement so that the parties involved would be at work by 9:00 in the morning. Suppose that Bert has determined that the best solution to the problem is to drive from his home to Ernie's home, but never to wait if Ernie is not ready when he arrives. If we know that Ernie may under-estimate how quickly he will be ready by as much as 5 minutes, then we must find a value of γ for the system pictured in Figure 9.2 such that

$$\max(\text{ready} - \text{arrive}) \leq 0. \tag{9.3}$$

Determining how much time Ernie may take to get ready is "synthesizing" Ernie's temporal behavior to meet Bert's requirements.

Figure 9.2 is one instance of an I-BDS synthesis problem: the system behavior is expressed as an I-BDS diagram in which some time bounds may be variables, plus some additional bounds on how far apart events ought to occur.

Table 9.1: Time Separations for the Diagram of Figure 9.2

from \ to	wake	phone	leave	arrive	ready	meet	work
wake	0	45	60	80	50γ	$80 \oplus 50\gamma$	$120 \oplus 90\gamma$
phone	-35	0	15	35	5γ	$35 \oplus 5\gamma$	$75 \oplus 45\gamma$
leave	-45	-10	0	20	$(-5)\gamma$	$20 \oplus (-5)\gamma$	$60 \oplus 35\gamma$
arrive	-60	-25	-15	0	$(-20)\gamma$	$0 \oplus (-20)\gamma$	$40 \oplus 20\gamma$
ready	$(-35)(-\gamma)$	$-\gamma$	$15(-\gamma)$	$35(-\gamma)$	0	$35(-\gamma) \oplus 0$	$75(-\gamma) \oplus 40$
meet	$(-35)(-\gamma)$, -60	$-\gamma$, -25	$15(-\gamma)$, -15	$35(-\gamma)$, 0	0, $(-20)\gamma$	0	40
work	$-65(-\gamma)$, -90	$-30(-\gamma)$, -55	$-15(-\gamma)$, -45	$5(-\gamma)$, -30	-30, $(-50)\gamma$	-30	0

Definition 9.3 *An* **I-BDS synthesis problem** *consists of a set of desired time separations, $\mathcal{T}$, and an I-BDS system, $\mathcal{I}$, which meets all of the requirements for an I-BDS system(Section 2.1), except that for a given activity, a, the lower and upper bounds $\delta(a)$ and $\Delta(a)$ on a's duration may be parameterized with variables as $\delta(a) = \gamma_a$, and $\Delta(a) = \gamma_a + x$ for non-negative x.*

As in the verification problem, we can calculate the maximum separations between two events by carrying the variables γ into our calculations as necessary. The separations corresponding to Figure 9.2 are given in Table 9.1. Note that sometimes there may be more than one possible bound on the separation, since we cannot always tell which of two bounds is tightest when one or more of them contains a variable.

In this example, there are two possible separations from the events `work` and `meet` to events which precede `meet` in the graph because there are two distinct backwards paths through the node `meet`, one including a γ term and one not, between all such pairs of events. For those separations which include two possible terms, the expressions at the top of a given table entry correspond to the case in which Ernie is not ready before Bert arrives and the bottom corresponds to the case in which Ernie is ready.

In order to assure that Ernie is ready before Bert arrives, we must make sure that the separation from Bert's arrival to Ernie being ready is less than or equal to 0. We

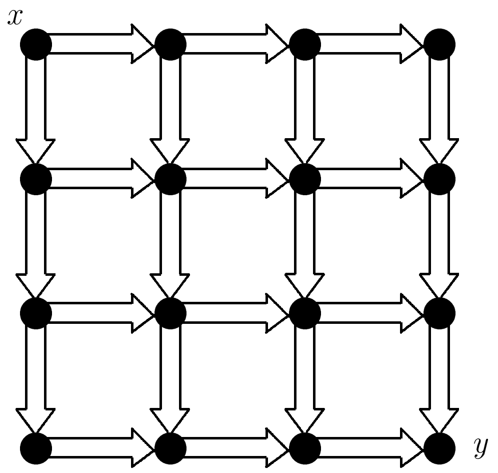


Figure 9.3: The size $n = 4$ instance of the $\Theta(n^2)$ size I-BDS for which there are $\Theta\left(\frac{2^{2n}}{\sqrt{n}}\right)$ paths from y to x.

examine the table and determine that this separation has value $(-20)\gamma$. Thus as long as γ is no more than 20, the constraint will always be met. If we substitute 20 for γ throughout Table 9.1, we see that the second item in each table entry is clearly the tightest bound on the time separation for that pair, and corresponds to the "Ernie is ready on time" possibility.

When there are few unbound variables and few sets of such alternate paths, this approach of calculating the separations for all possible path bounds may be a reasonable solution. Unfortunately, there are very simple examples for which the total number of such expressions grows too quickly to make this method be of practical use. Figure 9.3 gives the $n = 4$ instantiation of the example in which there are $\Theta(n^2)$ events and activities, but $\Theta(\frac{2^{2n}}{\sqrt{n}})$ possible maximum time separations of x relative to y.

To see this, we note that each maximum separation of x relative to y corresponds to a single "backwards" path from y to x. If all $(n-1)^2$ activity bounds are to be synthesized, we then have $\Theta(n^2)$ variables. Each path from y to x has $n-1$ arcs to the left, and $n-1$ arcs directed upwards. These arcs can occur in any order, thus giving us $\binom{2n-2}{n-1}$ different paths. Using Stirling's approximation [Sti30, Knu73] of

$$n! \approx \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n \tag{9.4}$$

we get a total number of paths which is

$$\Theta\left(\frac{2^{2n}}{\sqrt{n}}\right). \tag{9.5}$$

Similarly, any expression for the maximum time of y relative to x must contain as many terms if it is factored out into standard linear form.

It is important to note that this does not guarantee that the IBDS-synthesis problem cannot be solved in polynomial time, but rather that it cannot be quickly solved in the manner outlined above.

9.5 D-BDS Synthesis Problems

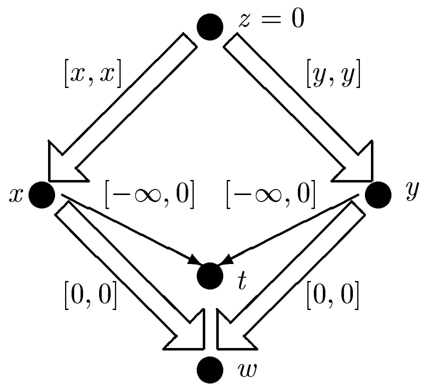
While the method in the above section is useful for systems in which one knows the causal structure of all activities in the system, it is not adequate for the task of interface timing synthesis. Recall that, in general, a databook entry describing a device's timing behavior does not provide us with a full causal description of the device. Instead, causal constraints which match the form of I-BDS relations may be augmented with simple timing guarantees of the form

$$x_j + \delta_{i,j} \leq x_i \leq x_j + \Delta_{i,j}, \tag{9.6}$$

where $\delta_{i,j} \leq \Delta_{i,j}$, and value assignments $\delta_{i,j} = -\infty$ and $\Delta_{i,j} = \infty$ are used to indicate the absence of the lower and upper bound constraints respectively on x_i. Synthesis problems over such systems are called **D-BDS synthesis problems**.

Definition 9.4 *A* **D-BDS synthesis problem** *is an I-BDS synthesis problem to which there may be added events and timing guarantees of the form of Equation 9.6.*

D-BDS synthesis problems are depicted as I-BDS synthesis problems, except that these additional guarantees are represented with thin arcs. Figure 9.4 gives a representative system.



Desired System Behavior				
2	$\leq$	$w - z$	$\leq$	3
0	$\leq$	$t - z$	$\leq$	1

Time Separations					
from | to	z	x	y	t	w
z	0	x	y	x, y	$x \oplus y$
x	$-x$	0	$y - x$	0, $y - x$	$0 \oplus (y - x)$
y	$-y$	$x - y$	0	$x - y$, 0	$0 \oplus (x - y)$
t	∞	∞	∞	0	∞
w	$-y$ $-x$	$x - y$ 0	0 $y - x$	$x - y$, $y - x$	0

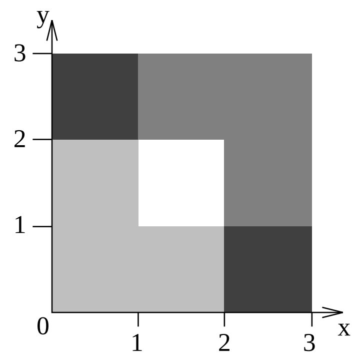


Figure 9.4: A D-BDS synthesis problem which produces disjoint solutions.

If we apply the timing verification algorithm to the constraints pictured in the diagram on the top left hand side of Figure 9.4, we get the time separations as pictured in bottom left of Figure 9.4. As was the case with the timing separations pictured in Table 9.1, when the algorithm is applied we come up with time separations which may be expressed with one of two different formulae. In the table, those separations for which there are two entries may be grouped into two categories — the top and the bottom entries, each of which bounds a consistent set of time separations. The top and bottom separations correspond to the two synthesis possibilities, $x \leq y$ and $x \geq y$, respectively.

The diagram at the bottom right of Figure 9.4 represents the effect of the desired system behavior on the values of x and y as the intersection of two regions. The lightest gray region corresponds to those values of x and y for which the equation

$$0 \leq t - z \leq 1 \tag{9.7}$$

is guaranteed to be satisfied, and the region of middle gray color corresponds to the the values of x and y for which the equation

$$2 \leq w - z \leq 3 \tag{9.8}$$

is satisfied. The area where the two regions overlap is denoted with the darkest shade of gray. These two non-overlapping regions correspond to the solutions for $x \geq y$ and $x \leq y$, and are disjoint.

Note that if our set of desired system behaviors were expressed by the four equations

$$\begin{aligned} 0 \leq \quad & x - z & \leq 1 \qquad (9.9)\\ 0 \leq \quad & y - z & \leq 1 \qquad (9.10)\\ 1 \leq \quad & w - z & \qquad (9.11)\\ & t - z & \leq 0, \qquad (9.12) \end{aligned}$$

the only two possible solutions would be $x = 0, y = 1$ and $x = 1, y = 0$. This allows us to create a small synthesis sub-problem in which x and y are the boolean complement of each other. We could then use McMillan and Dill's NP-completeness reduction (Figure 4.2) to show that D-BDS synthesis is NP-hard.

9.6 Valid Synthesis Problems

Recall the nonsensical synthesis problem presented in Figure 2.14, in which a guarantee was made about the time relationships across events whose timing relationships have not yet been synthesized. A more basic version of this problem is depicted at the left hand side of Figure 9.5. We are asked to choose a value for variable α, but yet we are promised that no matter how far apart the events x and y occur, z will happen within 15 to 30 time units after x. Clearly, if we chose $\alpha > 20$, there is no consistent assignment of event times for the system. Without a explicitly specified

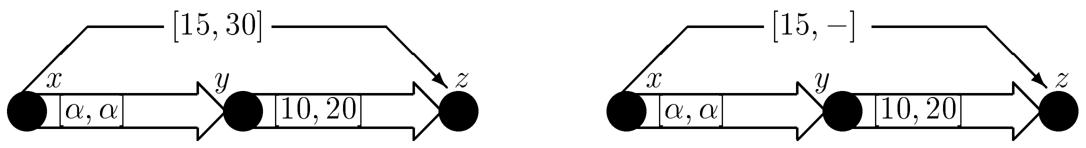


Figure 9.5: A nonsensical synthesis problem.

requirement that we choose a value of α which is less than 20, we must assume that the system has been incorrectly specified.

The interpretation of the diagram at the left of Figure 9.5 is somewhat more difficult. If we were to choose a value of 0 for α, we would expect z to happen as early as 10 time units after x. The guarantee of a minimum separation of 15 is open to several interpretations including:

- the diagram given is a simplification of the actual system and in reality time separations within the bound of $[10, 20]$ from y to z do not occur randomly, but are somehow relative to the time of event x, and
- the bound of $[10, 20]$ from y to z is not tight, but should instead be $[15, 20]$ or tighter.

When we consider the effects that these two different interpretations might have when propagated throughout a larger system, it becomes clear that we must agree upon a consistent interpretation.

9.7 Summary

In this Chapter we have discussed timing synthesis problems both in general and as they apply to interface hardware synthesis. We introduced a taxonomy for practical interface hardware synthesis and showed that the natural timing synthesis problems for I-BDS and D-BDS systems are respectively likely and certainly NP-hard.

Part V

Conclusions and Related Work

Chapter 10
CONCLUSIONS

10.1 Contributions

In this work we have brought together and solved problems in two different domains: hardware interface timing and the max-plus algebra.

10.1.1 Hardware Interface Timing

The first area is concerned with assuring that interconnected hardware modules will communicate successfully. Each hardware module has a protocol describing the way it may communicate with its environment, and that protocol has temporal **requirements** it makes of the other modules with which it communicates. These requirements are generally constraints of the form

$$t_j + \delta \leq t_i \leq t_j + \Delta \tag{10.1}$$

where t_i and t_j are times of the events — or signal level changes — in the protocol and $[\delta, \Delta]$ with $\delta \leq \Delta$ is a time interval. In turn that same hardware module protocol makes promises on the time ranges within which it will respond to changes on its inputs. These promises come in two forms, **guarantees**

$$t_j + \delta \leq t_i \leq t_j + \Delta \tag{10.2}$$

which relate absolute bounds between events in the interface and **delay** relationships

$$\max_{j \in \text{pred}(e_i)} t_j + \delta_j \leq t_i \max_{j \in \text{pred}(e_i)} t_j + \Delta_j \tag{10.3}$$

where event e_i is dependent on the events in $\text{pred}(e_i)$, and happens only after every such predecessor event e_j plus some bounded delay has occurred. We considered two different types of timing problems that occur in this domain, and which ask the following questions about a hardware interface:

- **timing verification problems:** Does a given system respect a set of restrictions made upon the relative times of its events, and
- **timing synthesis problems:** Can such restrictions be met by altering the ordering or duration of specific activities represented by the system?

Contributions of this work in the area of hardware interface logic timing include:

- a detailed exploration of the timing verification and timing synthesis problems and the differences between them;
- the `ShortCircuit` algorithm, the first correct algorithm for finding the maximum time separations between pairs of events subject to a system of equations of the form given in Equations 10.2 and 10.3; and
- a preliminary discussion of the requirements of an automated method to perform timing synthesis for hardware interfaces.

10.1.2 Linear Max-Plus Systems

Equations 10.3 and 10.2 above are specific examples of primitives expression **synchronization** and **bounded delay** relations, which can be used to model the timing behavior of a wide variety of systems ranging from from the planning and management of large projects, as done with PERT charts [SPO58, HM61], to the design and verification of computer hardware interfaces, as mentioned above.

The **max-plus algebra** [GM77, CMQV89, BCOQ92], which is the subject of much research in control theory is naturally suited to representing the more general representations of the synchronization and bounded delay primitives respectively as

$$t_i = \bigoplus_j (t_j \otimes \alpha_{i,j}) \tag{10.4}$$

and

$$t_j \otimes (-\delta_j) \leq t_i \leq t_j \otimes \Delta_j \tag{10.5}$$

where $\oplus$ is the operation “max”, $\otimes$ is the operation “+”, each $\alpha_{i,j}$ is drawn from $\mathcal{R} \cup -\infty$.

Due to their ability to express many problems including the interface timing verification problem above, **linear max-plus systems** are a popular subject of study. The consist of a set of m equations over n variables $\{x_0, \ldots, x_{n-1}\}$ where the ith such equation has form

$$a_i \oplus \left[\bigoplus_{j=0}^{n-1} (C_{i,j} \otimes x_j) \right] = b_i \oplus \left[\bigoplus_{j=0}^{n-1} (D_{i,j} \otimes x_j) \right]. \tag{10.6}$$

A fundamental deficiency in the study of linear max-plus which was remedied by this work was the lack of a reasonable method for solving linear max-plus systems. A previous brute-force method due to Gaubert [Gau92] was extremely expensive computationally.

Contributions of this work to the study of the max-plus and related algebras include the following:

- the `MPsolv` algorithm for solving any linear max-plus system of equations;
- the extension of `MPsolv` to maximizing and minimizing linear max-plus expressions subject to an arbitrary max-plus system; and
- that the `MPsolv` technique applies to any dioid $(\mathcal{D}, \oplus, \otimes)$ in which the $\oplus$ operation induces a total order and $(\mathcal{D} \setminus \{\epsilon\}, \otimes)$ is a group.

The `ShortCircuit` and `MPsolv` methods and the problems they solve are shown to be equivalent, with both methods binding from above the maximum solution to the system when some variable $x_0 = 0$. Both methods rely upon phrasing the system equations with the **upper bound constraint (UBC)** also introduced here.

10.2 Open Problems

While several questions have been answered by this work, still others have been left unanswered or have been raised. They include the following open problems.

Open Problem 10.1 *What is the worst-case time complexity of the* `ShortCircuit` *and* `MPsolv` *algorithms?*

Because of its use in practical applications, we would like to know what the worst case time for the `ShortCircuit` algorithm is so we may compare it with other techniques.

Should both `ShortCircuit` and Yen et. al.'s `MaxSeparation` algorithms be shown to have non-polynomial worst case performance, it still remains to be determined what the worst case complexity is for the solution of linear max-plus systems.

Open Problem 10.2 *What is the worst-case time complexity for finding the maximum solution to an arbitrary linear max-plus system of m equations over n variables?*

Because of its similarity to regular linear programming we make the following conjecture.

Conjecture 10.1 *The worst-case time complexity for finding the maximum solution to an arbitrary linear max-plus system of m equations over n variables, and therefore the worst case time complexity for minimizing or maximizing a linear max-plus expression subject to a linear max-plus system, is polynomial in m and n.*

While we have shown that the `MPsolv` technique can solve systems of equations in a pseudoring, there are many dioids which are not pseudorings. This leads to the following open problem.

Open Problem 10.3 *Does* `MPsolv` *give any insight into the techniques that may be developed to solve arbitrary linear dioid systems?*

Recall from Chapter 2 that for any I-BDS the time $d(a)$ required for activity a was within the interval $[\delta_a, \Delta_a]$, but that the activity times were independent of each other. For many real-world problems, this assumption is incorrect. The ranges given for the delays of interface hardware are often correlated. There may be, for example, variations in delay times that are dependent on properties inherent in the physical process that produces the chip. This may result in there being a range of behaviors for different hardware modules fabricated at different times while individual modules will have delay values occurring at one end of the specified range or the other throughout the chip. We call this phenomenon **delay tracking**.

Open Problem 10.4 *Are there practical methods for solving the interface timing verification problem when guarantees and delays may be built from bounds* $[\delta \cdot \lambda, \Delta \cdot \lambda]$ *where the* δ *and* Δ *values are scalar coefficients as before but the* λ*'s are bounded variables used to express the fact that time bounds for parts within a device may track with each other?*

During the process of timing synthesis, the max-plus constraints of the incompletely synthesized skeleton describe a space of possible temporal behaviors for the system. The process of choosing delay values for the synthesis variables restricts that space, and our goal is to restrict the space until it lies within some given desirable temporal space. One can easily conceive of a slight variation of this problem in which the desired timing behaviors are already met, but one wishes instead to choose separations which optimize some other function such as the area required by interface hardware or the cost of the components. These functions will not necessarily be expressible as linear max-plus functions but instead are more likely to be linear or quadratic in our familiar linear algebra.

Open Problem 10.5 *Can the optimization technique as performed with* `MPsolv` *be extended to optimize more sophisticated expressions subject to a linear max-plus system?*

As we demonstrated in Chapter 9, the obvious timing synthesis problem for hardware interface logic is NP-hard. However, it is not clear that practical interface timing synthesis problems are truly that difficult.

Open Problem 10.6 *Are there models of timing synthesis which are both useful for interface timing synthesis and tractable?*

Chapter 11

OTHER RELATED WORK

Although not directly concerned with interface timing problems or the solution of linear max-plus systems, there are several problem domains which are related and may be of interest. Several, but by no means all, are presented below.

11.1 Related Artificial Intelligence Problems

While much artificial intelligence research can be said to fall in the area of determining the temporal separations among a set of events, in practice most such research is concerned with rather more difficult problems than the I-BDS and D-BDS problems presented in Chapter 2. Examples include Allen's work [All83] in which relationships among time intervals are expressed qualitatively with primitives such as "interval a is before interval b" and "interval c overlaps d", and the goal is to deduce similar relationships between intervals not directly related, and the work of Dean and Boddy [DB88] in which establishing possible relative orderings of events which are subject to constraints dependent on non-temporal state variables is the goal.

Dechter et. al. [DMP91] discuss **temporal constraint satisfaction** problems. In the simplest version of this problem, the times of pairs of events are related with intervals so that the interval $[\delta, \Delta]$ from event a to b indicates that

$$\delta \leq t_b - t_a \leq \Delta \tag{11.1}$$

which is easily seen to be solvable as an all pairs shortest paths problem as discussed in Section 9.1. As shown by Gondran and Minoux [GM77], this can also be solved using the min-plus closure of the matrix M whose (i, j)th entry is the upper bound on $t_i - t_j$. Dechter et. al. then extend the problem to apply to pairs of events (a, b) related by a set of time intervals with the understanding that any one of the intervals may apply to the events' relationship. They state the problem algebraically, so that if $T_{i,j}$ is the set of intervals expressing the possible time separations between events

i and j, it should be the case that

$$T_{i,j} = T_{i,j} \oplus (T_{i,k} \otimes T_{k,j}), \tag{11.2}$$

where $\oplus$ takes the intersection of possible values within two sets of intervals and $\otimes$ is the composition of intervals, with value r allowed in $T_1 \otimes T_2$ only if there exist r_1 in T_1 and r_2 in T_2 such that $r = r_1 + r_2$.

11.2 Timed Petri Nets

While we have so far concerned ourselves with relating the times of a finite set of events, there are many max-plus and related algebra problems that concern themselves with the temporal relationships among possibly infinitely repeating sequences of events. Such systems of events may be modeled with timed event graphs such as the one pictured in Figure 11.1. Timed event graphs are a subclass of Petri nets [Pet62, Mur89], of which there are many timed variations [Ram74, Mer74, Chr85, TPN85, BRR87, And91].

The Petri nets discussed here have the following properties:

- the larger vertical bars, called **transitions**, and labeled here with one of $\{u_1, u_2, x_1, x_2, x_3, y\}$ represent each of the events that occurs in the Petri net.

- the larger circles, called **places**, and labeled P_{t_1,t_2} for certain pairs of transitions t_1 and t_2 in Figure 11.1, represent various conditions that may hold during an execution of the Petri net. For a given place P_i, we say that any transition T_j with a directed edge to P_i is an **input transition** of P_i and any transition T_k with a directed edge from P_i is an **output transition** of P_i. **Input places** and **output places** are defined similarly for any transition T_i.

- A solid circle inside a place, called a **token**, indicates that the associated condition currently holds. Transition T_i is **enabled**, or allowed to occur, only when each of its input places contain a token. When such a transition occurs, we say it **fires**. The result is that a token is added to each of the output places of the transition and removed from each input place of the transition.

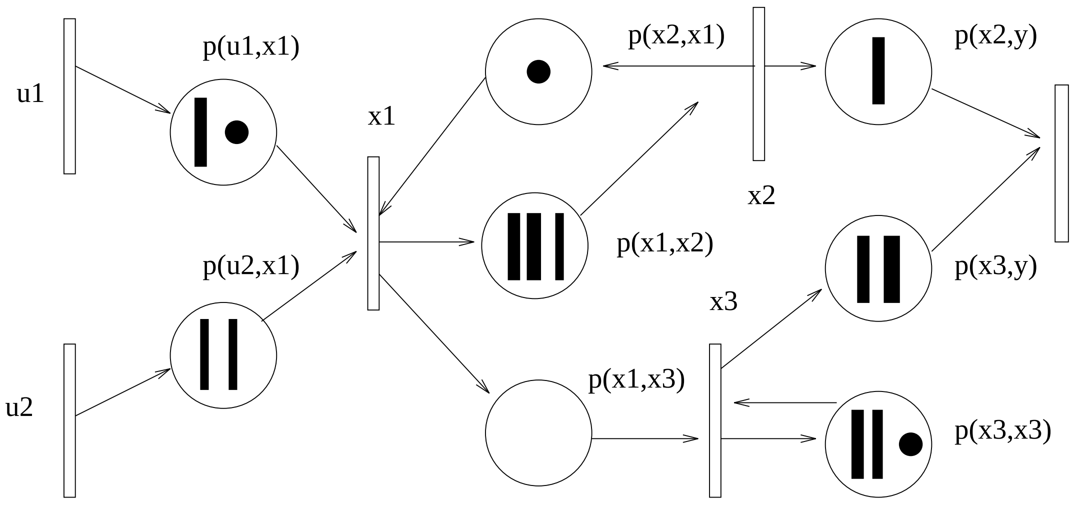


Figure 11.1: A timed event graph with deterministic durations for places. Small dark circles represent tokens in the original marking; hollow bars represent the number of delay units per place.

It is possible to construct a net such that more than one transition has the same place P_i as an input. In such cases, we say that the net has **choice**. If both transitions are simultaneously enabled, then they are in **conflict** with each other since only one can remove the enabling tokens. Conflict may be resolved arbitrarily, by assigning relative probabilities to the given choices, or by assigning priorities to choices. When states are changed due to a transition firing, transitions not in conflict with the firing transition continue to remain enabled.

A **marked graph** or **event graph** is a Petri net in which each place has exactly one input transition and one output transition and therefore no choice. Figure 11.1 is a **timed event graph**. Each place is assigned a duration, here represented by zero to three bars in each place, indicating that a token in that place enables its output transition only after it has been in the place for as many time units as there are bars in the place. Transitions are assumed to fire instantaneously as soon as they are enabled.

Cohen et. al. [CMQV89], demonstrate that a somewhat more sophisticated dioid than the max-plus dioid can be used to completely represent the input/output and transition behavior of such a timed event graph. For the event graph in Figure 11.1, with initial marking as shown, we note that the time of the n-th occurrence of transition x_3, denoted as $(x_3)_n$ is related to the n-th transition x_1 and the next earliest occurrence of x_3 as:

$$(x_3)_n = \max((x_1)_n, (x_3)_{n-1} + 2) \tag{11.3}$$

or, in max-plus notation

$$(x_3)_n = (x_1)_n \oplus 2(x_3)_{n-1}. \tag{11.4}$$

Note that these equations relate the times of occurrences of event x as a function of n, the event occurrence number. In order to invert these relations, one must also be able to express the event number of x_3 at time t, or $[x_3]_t$ as a function of t, this time using the min-plus dioid equation

$$[x_3]_t = [x_1]_t \oplus' 1[x_3]_{t-2} \tag{11.5}$$

where "$\oplus'$" represents the binary minimum operation. This means that the most recent event number for x_3 at time t is the minimum of the most recent event number for x_1 at time t and one plus the most recent event number of x_3 at time $t-2$. Although it is true that $\min(x, y) = -max(-x, -y)$, this is not a linear transformation in either dioid and we cannot readily express both equation types in either dioid. This is because terms of the form $-\delta$ are multiplicative inverses in these dioids.

Cohen et. al.'s solution is to instead create a system variables $\gamma^n\delta^t$ such that the assignment $x = \gamma^n\delta^t$ has the meaning "occurrence number n of event x happens no earlier than time t". They then demonstrate that these $\gamma^n\delta^t$'s can be thought of as both the piece of information "event number n happens no earlier than t", about a given event x_i, and the **shift operator** relating two events x_i and x_j. Applying a shift operator $\gamma^n\delta^t$ from x_i to x_j, to the information $x_i = \gamma^{n'}\delta^{t'}$ results in the information $x_j = \gamma^{n'+n}\delta^{t'+t}$ for x_j. In order to combine both meanings, all variable assignments $x = \gamma^n\delta^t$ are considered to be instances of the single variable $x = \gamma^0\delta^0$ operated upon by the shift operator $\gamma^n\delta^t$. The $\otimes$ operator is used to apply shift operators and the $\oplus$ is used to take the union of the resulting information.

These observations result in the dioid $(MinMax\langle\langle\gamma, \delta\rangle\rangle, \oplus, \otimes)$, with identities $\gamma^{+\infty}\delta^{-\infty}$ for $\oplus$ and $\gamma^0\delta^0$ for $\otimes$ such that

- Elements of $MinMax\langle\langle\gamma, \delta\rangle\rangle$ are (possibly infinite) unions of terms $\gamma^n\delta^t$ with this union expressed by $\oplus$

- $\gamma^i\delta^j \otimes \gamma^{i'}\delta^{j'} = \gamma^{i+i'}\delta^{j+j'}$ and $\otimes$ distributes over infinite sequences of $\oplus$
- Additionally, $\gamma^n\delta^t \oplus \gamma^n\delta^{t'} = \gamma^n\delta^{max(t,t')}$ and $\gamma^n\delta^t \oplus \gamma^{n'}\delta^t = \gamma^{min(n,n')}\delta^t$. The intuition for this is that if we have two bounds, t and t' on the earliest time for the n-th occurrence of a given event, then the larger of $\{t, t'\}$ includes the other, and similarly for the smaller of two different event occurrence numbers with the same time value.

The effect of equations 11.4 and 11.5 above can then be represented in the single equation

$$x_3 = \left(x_1 \otimes \gamma^0\delta^0\right) \oplus \left(x_3 \otimes \gamma^1\delta^2\right). \tag{11.6}$$

This expresses that the the n-th occurrence of x_3 can happen no earlier than t only if either the n-th occurrence of x_1 can happen no earlier than t or the $(n-1)$-th occurrence of x_3 can happen no earlier than $t-2$.

In the case of figure 11.1, we can use this dioid to represent the input, output, and transition behavior of the timed event graph as

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \underline{0} & \gamma & \underline{0} \\ \delta^3 & \underline{0} & \underline{0} \\ \underline{1} & \underline{0} & \gamma\delta^2 \end{bmatrix} \otimes \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \oplus \begin{bmatrix} \gamma\delta & \delta^2 \\ \underline{0} & \underline{0} \\ \underline{0} & \underline{0} \end{bmatrix} \otimes \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{11.7}$$

$$\begin{bmatrix} y \end{bmatrix} = \begin{bmatrix} \underline{0} & \delta & \delta^2 \end{bmatrix} \otimes \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \tag{11.8}$$

where $\underline{0}$ is the $\oplus$ identity, $\gamma^{+\infty}\delta^{-\infty}$ and $\underline{1}$ is the $\otimes$ identity, $\gamma^0\delta^0$.

In general, for a timed event graph with internal transitions X, inputs U and outputs Y, we wish to solve the equations

$$X = AX \oplus BU \tag{11.9}$$
$$Y = CX \tag{11.10}$$

where U,Y and X are column vectors of dimension $i \times 1$, $j \times 1$,and $k \times 1$ respectively representing the inputs, outputs and internal transitions of the system. The $k \times k$ matrix A represents the shift operators between all internal transitions with an entry $A_{i,j} = \gamma^k\delta^t$ indicating that in the original net there is a place containing k tokens and having delay t between transitions i and j. The $k \times i$ matrix B and $j \times k$ matrix

C represent similar relationships between the internal transitions and the inputs and outputs, respectively.

It is shown that solutions to the above systems of equations are of the form

$$X = AX \oplus BU = X(AX \oplus BU) \oplus BU = \cdots = A^* \otimes B \otimes U \tag{11.11}$$

$$Y = CX = C \otimes A^* \otimes B \otimes U. \tag{11.12}$$

Note that if we wish to represent all behavior of the system over time, the elements of X,U and Y must represent the *infinite* behavior of each of their respective transitions.

The solution to the system in Figure 11.1 is

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \gamma\delta\left(\gamma\delta^3\right)^* & \delta^2\left(\gamma\delta^3\right)^* \\ \gamma\delta^4\left(\gamma\delta^3\right)^* & \delta^5\left(\gamma\delta^3\right)^* \\ \gamma\delta\left(\gamma\delta^3\right)^* & \delta^2\left(\gamma\delta^3\right)^* \end{bmatrix} \otimes \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{11.13}$$

$$\begin{bmatrix} y \end{bmatrix} = \begin{bmatrix} \gamma\delta^5\left(\gamma\delta^3\right)^* & \delta^6\left(\gamma\delta^3\right)^* \end{bmatrix} \otimes \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{11.14}$$

Using values of $\gamma^*\delta^0 = \gamma^0\delta^0 = \underline{1}$ for the inputs u_1 and u_2 to represent infinitely many tokens becoming ready at the inputs at time 0, we get a final solution of

$$y = \left(\gamma\delta^5 \oplus \gamma^6\right) \otimes (\gamma\delta^3)^* = \delta^6(\gamma\delta^3)^* \tag{11.15}$$

which is interpreted as "the 0^{th} token output arrives at time 6, and thereafter one token arrives every 3 time units." In general, a solution of the form

$$\gamma^i\delta^j(\gamma^p\delta^q)^* \tag{11.16}$$

indicates that the i^{th} token arrives at time j and thereafter p tokens arrive every q time units. A net can be simulated on any input that can be expressed in the form

$$\bigoplus_{0 \leq k} \gamma^{i_k}\delta^{j_k}. \tag{11.17}$$

11.2.1 Time Separations Bounds for Concurrent Systems

Similar to the problem above is the task of determining the maximum possible time separations [HB+95a] between events in a **process graph**, which is similar to the timed event graph in the definition above except that there are no input or output transitions and the time a token spends at a given place before it enables its output transitions is expressed with a bounded interval. This problem asks "what is the maximum possible time difference

$$(x_i)_{n+\delta} - (x_j)_n \tag{11.18}$$

for fixed events x_i and x_j over all occurrences $n+\delta$ and n separated by a fixed δ. The technique presented by Hulgaard et. al. [HB$^+$95a] relies upon mathematically "unfolding" the process graph until it can represent all occurrences of $(x_i)_{n+\delta}$ and $(x_j)_n$ for any n. There are infinitely many such n, but for any finite n, the graph may be represented as the concatenation RS^kT of graphs where R represents the initial portion of the unfolded graph, T represents the final portion of the unfolded process graph, and S^k is a repeating sub-graph within the unfolded process graph. These subgraphs can be represented using a matrix form similar to that in Section 5.4.2. Then S^*, the closure of the matrix S can be used to represent all of the possible repeating portions of the graph. A refinement of the technique which handles certain types of choice in the original process graph has also been developed [HB95b].

In a similar vein, work on a second-order theory of linear max-plus and min-plus systems [Plu91, CGQ93] concentrates on determining the maximum number of tokens which may be present in a place of a timed event graph and how long they may reside there.

11.3 Min-Max-Plus Systems

Recall from Section 4.2 that if $\mathcal{P} \neq \mathcal{NP}$, there can be no polynomial-time algorithm for determining maximum time separations for systems of events related with guarantees, delays, and min-type delays, it is still sometimes desirable to do so. Research into the solution of many different types of systems which include min, max, and addition operations [CG79, CGM80, Ols91, Ols94] is also quite active and may lead to new solutions in this area.

Bibliography

[All83] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, Nov 1983.

[And91] Charles Andre. Delays in synchronized elementary net systems. In G. Rozenberg, editor, *Advances in Petri Nets 1991*, number 524 in Lecture Notes in Computer Science. Springer–Verlag, 1991.

[Baa88] Sara Baase. *Computer Algorithms: Introduction to Design and Analysis.* Addison-Wesley, 1988.

[BCGZ84] R.E. Burkard, R.A. Cuninghame-Green, and U. Zimmermann, editors. *Algebraic and Combinatorial Methods in Operations Research.* Number 19 in Annals of Discrete Mathematics. North-Holland, 1984.

[BCOQ92] Francois Louis Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. *Synchronization and Linearity: an algebra for discrete event systems.* John Wiley & Sons, 1992.

[BGM91] J.A. Brzozowski, Tony Gahlinger, and F. Mavaddat. Coherence and satisfiability of waveform timing specifications. *Networks*, 21(1):91–107, Jan 1991.

[BN86] J. Burns and A. Newton. SPARCS: a new constraint-based ic symbolic layout spacer. In *Proceedings of the Custom Integrated Circuits Conference*, 1986.

[Bor88] Gaetano Borriello. *A New Interface Specification Methodology and its Application to Transducer Synthesis.* PhD thesis, University of California, May 1988. Report No. UCB/CSD 88/430.

[Bor92] Gaetano Borriello. Formalized timing diagrams. In *Proceedings of the European Design Automation Conference*, March 1992.

[BRR87] W. Brauer, W. Reisig, and G. Rozenberg, editors. *Advances in Petri Nets 1986*. Number 254 in Lecture Notes in Computer Science. Springer–Verlag, 1987. Part I: *Petri Nets: Central Models and their Properties* Part II: *Petri Nets: Applications and Relationships to Other Models of Concurrency*.

[Bur94] Steven M. Burns. August 1994. Personal communication.

[Bur95] Steven M. Burns. September 1995. Personal communication.

[CG79] R. A. Cuninghame-Green. *Minimax Algebra*. Number 166 in Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1979.

[CGM80] R. A. Cuninghame-Green and P. F. J. Meijer. An algebra for piecewise-linear minimax problems. *Discrete Applied Mathematics*, (2):267–294, 1980.

[CGQ93] G. Cohen, S. Gaubert, and J.P. Quadrat. From first to second-order theory of linear discrete event systems. In *Proceedings of the IFAC 12th Triennial World Congress*, volume 1, pages 679–682, 1993.

[Chr85] Philippe Chretienne. Timed event graphs: A complete study of their controlled excecutions. In *[TPN85]*, pages 47–54, 1985.

[Cin93] Viktor Cingel. A graph-based method for timing diagrams representation and verification. Number 683 in Lecture Notes in Computer Science, pages 1–14. Springer-Verlag, May 1993.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. M.I.T. Press, 1990.

[CMQV89] Guy Cohen, Pierre Moller, Jean-Pierre Quadrat, and Michel Viot. Algebraic tools for the performance evaluation of discrete event systems. *Proceedings of the IEEE*, 77(1):39–57, January 1989.

[CPMS91] Alberto Coen-Porisini, Angelo Morzenti, and Donatella Sciuto. Specification and verification of hardware systems using the temporal logic language TRIO. In Dominique Borrione and Ronald Waxman, editors, *Computer Hardware Description Languages and their Applications*, pages 43–62. North-Holland, April 1991. Proceedings of the IFIP WG 10.2 Tenth International Symposium on Computer Hardware Description Languages and their Applications.

[CWB94] Pai Chou, Elizabeth A. Walkup, and Gaetano Borriello. Scheduling for reactive real-time systems. *IEEE Micro*, July 1994.

[DB88] Thomas Dean and Mark Boddy. Reasoning about partially ordered events. *Artificial Intelligence*, 36:375–399, Dec 1988.

[DMP91] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[Gah90] Tony Gahlinger. *Coherence and Satisfiability of Waveform Timing Specifications.* PhD thesis, University of Waterloo, 1990. Research Report CS-90-11.

[Gau92] Stéphane Gaubert. *Théorie des Systèmes Linéaires dans les Dioïdes.* PhD thesis, L'École Nationale Supérieure des Mines de Paris, July 1992.

[Gla93] Bruce Gladstone. Specification of timing in a digital system. *ASIC and EDA*, pages 46–52, August 1993.

[GM77] Michel Gondran and Michel Minoux. *Graphs and Algorithms.* John Wiley and Sons, 1977. English Edition 1984, translated by Steven Vajda.

[GM84] Michel Gondran and Michel Minoux. Linear algebra in dioids. In R.E. Burkard, R.A. Cuninghame-Green, and U. Zimmermann, editors, *Algebraic and Combinatorial Methods in Operations Research*, number 19 in Annals of Discrete Mathematics. North-Holland, 1984.

[HB$^+$95a] Henrik Hulgaard, Steven M. Burns, , Tod Amon, and Gaetano Borriello. An algorithm for exact bounds on the time separation of events in concurrent systems. *IEEE Transactions on Computers*, Nov 1995. to appear.

[HB95b] Henrik Hulgaard and Steven M. Burns. Efficient timing analysis of a class of petri nets. In *Computer Aided Verification*, 1995.

[HM61] L.P. Hartung and J.E. Morgan. PERT/PEP — a dynamic project control method. Technical Report 61-816-2005, IBM Space Guidance Center, 1961.

[HP89] Sally Hayati and Alice Parker. Automatic production of controller secifications from control and timing behavioral descriptions. In *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pages 75–80, june 1989.

[Joh77] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24(1):1–13, 1977.

[KDC91] Karim Khordoc, Mario Dufrense, and Eduard Cerny. A stimulus/response system based on hierarchical timing diagrams. In *1991 IEEE International Conference on Computer-Aided Design*, pages 358–361. IEEE Computer Society Press, November 1991.

[KDC$^+$92] Karim Khordoc, Mario Dufrense, Eduard Cerny, Philippe-Andre Babkine, and Allan Silburt. Integrating behavior and timing in executable specifications. Technical report, Universite de Montreal, Dept d'informatique et recherche operationelle, October 1992.

[KDC$^+$93] Karim Khordoc, Mario Dufrense, Eduard Cerny, Philippe-Andre Babkine, and Allan Silburt. Integrating behavior and timing in executable specifications. In *11th IFIP WG10.2 International Conference on Computer Hardware Description Languages and their Applications - CHDL'93*, volume A-32, pages 399–416, April 1993.

[Knu73] Donald E. Knuth. *Fundamental Algorithms*, volume 1. Addison-Wesley, 2 edition, 1973.

[KRK88] Atsushi Kara, Ravi Rastogi, and Kazuhiko Kawamura. An expert system to automate timing design. *IEEE Design and Test*, 5(4):28–40, 1988.

[LW83] Y. Z. Liao and C. K. Wong. An algorithm to compact a vlsi symbolic layout with mixed constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2(2):62–69, 1983.

[MD92] Kenneth McMillan and David Dill. Algorithms for interface timing verification. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 1992.

[Mer74] P. Merlin. *A study of the recoverability of computer systems.* PhD thesis, University of California, 1974.

[MLC90] Alan R. Martello, Steven P. Levitan, and Donald M. Chiarulli. Timing verification using hdtv. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 118–123, june 1990.

[Mur89] T. Murata. Petri nets: properties, analysis, and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[Mye93] Chris Myers. Synthesis of timed asynchronous circuits. *IEEE Transactions on VLSI Systems*, June 1993.

[NT86] J. Nestor and D. Thomas. Behavioral synthesis with interfaces. In *Proceedings of the International Conference on Computer-Aided Design*, November 1986.

[Ols91] Geert Jan Olsder. Eigenvalues of dynamic max-min systems. *Discrete Event Dynamic Systems: Theory and Applications*, 1:177–207, 1991.

[Ols94] Geert Jan Olsder. On structural properties of min-max systems. In *Proceedings of the 11th international conference on the analysis and optimization of systems*, pages 237–246. Springer-Verlag, 1994. Also appears as TU Delft Faculty of Technical Mathematics and Informatics technical report 93-95.

[Org91] Don Organ. The enVision timing resolver. In *1991 IEEE International Test Conference*, 1991.

[Pet62] C. A. Petri. Kommunikation mit automaten. Technical report, Institut fur Instrumentelle Mathematik, 1962. Schriften des IIM Nr. 3.

[Plu90] Max Plus. Linear systems in (max,+) algebra. In *Proceedings of the 29th IEEE Conference on Decision and Control*, Dec 1990.

[Plu91] Max Plus. Second order theory of min-linear systems and its application to discrete event systems. In *Proceedings of the 30th IEEE Conference on Decision and Control*, Dec 1991.

[PP87] Alice C. Parker and Nohbyung Park. Interface and I/0 protocol descriptions. In R. W. Hartenstein, editor, *Hardware Description Languages*, volume 7, chapter 3.3, pages 111 – 136. Elsevier Science Publishers, B.V. (North-Holland), 1987.

[PW81] Alice Parker and J. Wallace. SLIDE: An I/O hardware description language. *IEEE Transactions on Computers*, C-30(6), june 1981.

[Ram74] C. Ramchandani. Analysis of asynchronous concurrent systems by Petri nets. Technical Report Project MAC TR-120, M.I.T., Cambridge, MA, 1974.

[SPO58] Bureau of Naval Weapons Special Projects Office. PERT summary phase report. Technical report, Department of the Navy, July 1958.

[Sti30] James Stirling. *Methodus Differentialis*. 1730. page 137.

[Tie91] Wolf-Dieter Tiedemann. Bus protocol conversion: from timing diagrams to state machines. In F. Pichler and R. Moreno Díaz, editors, *Computer Aides Systems Theory – EUROCAST'91*. Springer-Verlag, April 1991.

[TPN85] *International Workshop on Timed Petri Nets*. IEEE Computer Society Press, July 1985.

[Van93] Peter Vanbekbergen. *Synthesis of Asynchronous Controllers from Graph-Theoretic Specifications*. PhD thesis, Katholieke Universiteit Leuven, September 1993.

[VGD92] Peter Vanbekbergen, Gert Goossens, and Hugo De Man. Specification and analysis of timing constraints in signal transition graphs. In *Proceedings of the European Design Automation Conference*, March 1992.

[Vis76] C. Vissers. Interface, a dispersed architecture. In *Proceedings of the Third Annual Symposium on Computer Architecture*, pages 98–104, 1976.

[Wag91] Edouard Wagneur. Moduloids and pseudomodules: 1. dimension theory. *Discrete Mathematics*, 98:57–73, 1991.

[WB93a] Elizabeth Walkup and Gaetano Borriello. Automatic synthesis of device drivers for hardware/software co-design. In *Proceedings of the International Workshop on Hardware-Software Co-design*, October 1993.

[WB93b] Elizabeth Walkup and Gaetano Borriello. Interface timing verification with combined max and linear constraints. In *Proceedings of the ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, September 1993.

[WB94a] Elizabeth Walkup and Gaetano Borriello. Interface timing verification with application to synthesis. In *Proceedings of the 31st Design Automation Conference*, June 1994.

[WB94b] Elizabeth A. Walkup and Gaetano Borriello. Interface timing verification with combined max and linear constraints. Technical Report 94-03-04, University of Washington Department of Computer Science, March 1994.

[WB95] Elizabeth A. Walkup and Gaetano Borriello. A general linear max-plus solution technique. In *Proceedings of the BRIMS Workshop on Idempotency*. Cambridge University Press, 1995. To appear.

[YICW94] Ti-Yen Yen, Alex Ishiii, Al Casavant, and Wayne Wolf. Efficient algorithms for interface timing verification. In *Proceedings of the 1994 Euro-DAC*, pages 34–39. IEEE Computer Society Press, september 1994.

[Zim81] Uwe Zimmermann. *Linear Combinatorial Optimization in Ordered Algebraic Structures.* Number 10 in Annals of Discrete Mathematics. North-Holland, 1981.

Appendix A

Mc Millan and Dill's Algorithms

Appendix A.1
MAX-ONLY CONSTRAINTS

Pictured in Figure A.1.1 is McMillan and Dill's algorithm [MD92] for finding the maximum time difference between any pair of events in an I-BDS system. Their algorithm works for an equivalent, but distinct definition of an I-BDS system (Definition 2.1) for which the following information is required:

- for a given event, j, event i is a **predecessor** of j if there exists an activity, a such that activity a is a predecessor of j and i is the initiator of a (i.e. $a \in P(j)$ and $I(a) = i$),

- for a given event, i, event j is a **successor** of i if and only if i is a predecessor of j, and

- for event i with successor j, $\delta_{i,j}$ and $\Delta_{i,j}$ are the upper and lower bounds for the activity a initiated by i and which is a predecessor of j.

We speak of i being an **ancestor** of j if i is a predecessor of j or i is a predecessor of an ancestor of j. If i is an ancestor of j then j is a **decendant** of i.

While their algorithm is structured to cache intermediate values and only examine those time separations necessary to calculate a desired time separation between a particular pair of events, i and j, it is easier to understand the method if we examine it from the point of view of calculating all n separations of events j relative to a single i. The algorithm consists of two procedures, P, and S. Since P does not call S and does not examine the bounds $s_{i,j}$ calculated by S, we can think of P as first calculating bounds on the separations between all i and j and S then using those bounds.

$P(i, j)$ determines an upper bound, $p_{i,k}$ on the time separation from i to j as the negative of the minimum time from j to i for j a predecessor of i. We can think of $P(i, j)$ as sweeping back through the I-BDS graph by following each of its arcs backwards. If $P(i, k)$ has been calculated and there is an activity with bounds

McMillan and Dill's I-BDS algorithm

```
$P(i,j)$ *subroutine:*
    **if** $p_{i,j}$ has not already been defined **then**
        **if** $i = j$ **then** $p_{i,i} = 0$
        **else** $p_{i,j} = \min_{k \in \text{succs}(j)}(P(i,k) - \delta_{j,k})$
    **return** $p_{i,j}$
```

```
$S(i,j)$
    **if** $s_{i,j}$ has not already been defined **then**
        **if** $i = j$ **then** $s_{i,i} = 0$
        **else** $s_{i,j} = \min(\max_{k \in \text{preds}(j)}(S(i,k) + \Delta_{k,j}), P(i,j))$
    **return** $s_{i,j}$
```

Figure A.1.1: McMillan and Dill's Maximum Time Separations Algorithm for I-BDS Systems.

$[\delta_{j,k}, \Delta_{j,k}]$ from j to i then $P(i,j)$ is no more than $P(i,k) - \delta_{j,k}$. This characterizes the minimum time from j to i as being at least as large as $\delta_{j,k}$ plus the minimum time from k to i.

While $P(i,j)$ is an upper bound on the maximum time separation from i to j, it is not necessarily a tight bound. To obtain the tight bounds, S begins with bounds $s_{i,j} = p_{i,j}$ and then sweeps forward in the graph, tightening the bound to a given j as the maximum of $s_{i,k} + \Delta_{k,j}$ for all predecessors k of j.

Appendix A.2
GENERALIZED MAX-ONLY CONSTRAINTS

Figure A.2.1 gives McMillan and Dill's algorithm [MD92] for "generalized max-only" constraint sets which consist of constraints corresponding to both the delays and guarantees of Section 3.2. The algorithm as given here differs from the original only in that both constraint types are represented with UBCs.

McMillan & Dill's original algorithm in UBC terms	
Inputs:	a system of m UBCs over n variables x_0 through x_{n-1}
Outputs:	$s_{i,j}$ contains maximum possible value for x_j when $x_i = 0$

```
for $i := 0$ to $n-1$ do:
    for $j := 0$ to $n-1$ do:
        $s_{i,j} = \infty$
    $s_{i,i} = 0$
for every UBC of the form $x_j \leq x_i + \alpha$ do:
    if $s_{i,j} > \alpha$ then:
        $s_{i,j} \leftarrow \alpha$
Do:
    for $i := 0$ to $n-1$ do:
        for $j := 0$ to $n-1$ do:
            for $k := 0$ to $n-1$ do:
                if $s_{i,k} + s_{k,j} < s_{i,j}$ then: $s_{i,j} \leftarrow s_{i,k} + s_{k,j}$
            if $\exists$ UBC $x_j \leq \max_p(x_{p_1} + \alpha_{p_1}, x_{p_2} + \alpha_{p_2}, \ldots)$ then:
                if $x_{i,j} > \max_p(x_{i,p_1} + \alpha_{p_1}, x_{i,p_2} + \alpha_{i,p_2}, \ldots)$ then:
                    $x_{i,j} \leftarrow \max_p(x_{i,p_1} + \alpha_{p_1}, x_{i,p_2} + \alpha_{i,p_2}, \ldots)$
        If $s_{i,i} < 0$ then
            Report constraints inconsistent
            Exit algorithm
Until no $s_{i,j}$ changes
```

Figure A.2.1: McMillan and Dill's Maximum Time Separations Algorithm for D-BDS Systems.

Appendix B

Generalization of Max-Plus Properties

Appendix B.3
PSEUDORING PROPERTIES

In Chapter 5, we introduced the dioid, and showed that the max-plus algebra is such an object. However, as discussed in Chapter 6 the max-plus algebra, $(\mathcal{R} \cup \epsilon, \oplus, \otimes)$, is totally ordered and $(\mathcal{R}, \otimes)$ is a group. These properties make it easier to solve systems of max-plus equations and allow the `UBCsolv` and `MPsolv` techniques to be applied to any algebraic structure with the same properties.

Definition B.3.1 [Wagneur, ([Wag91], page 60)] *A* **pseudoring***, abbreviated* **pseudoring***, is a dioid, $(\mathcal{D}, \oplus, \otimes)$, with the following additional properties:*

- *$\mathcal{D}$ is a* **commutative dioid** *— $\otimes$ is commutative;*
- *$\mathcal{D}$ is* **totally ordered** *— for any a and b in $\mathcal{D}$,*

$$a = a \oplus b \quad \text{or} \quad b = a \oplus b; \tag{B.3.1}$$

- *the structure $(\mathcal{D} \setminus \{\epsilon\}, \otimes)$***, is a group***, indicating that for all $a \in \mathcal{D} \setminus \{\epsilon\}$, there exists an element, a^{-1} such that $a \otimes a^{-1} = e$.*

We may also describe a pseudoring as being "almost" a field. The difference is that pseudoring exchange the total order property of the $\oplus$ operator for the required $\oplus$-inverses of a field. There are several natural consequences of the definitions of pseudoring and $>$ which will be very useful for deducing the relative orderings of expressions, and which were used implicitly in Chapters 6 and 7. The proof of this next lemma makes extensive use of Proposition 6.1, which stated that for a and b in a pseudoring, one of the three relationships $a > b$, $a = b$, or $a < b$ must hold.

Lemma B.3.1 *The following properties hold for all $a, b, c, x \in \mathcal{D}$, where $\mathcal{D}$ is a pseudoring.*

1. *$a \geq b \Rightarrow ax \geq bx$,*

2. *$[ax = bx$ and $x \neq \epsilon] \Rightarrow a = b$,*

3. *$[a > b$ and $ax = bx] \Rightarrow x = \epsilon$,*

4. *$[a > b$ and $x \neq \epsilon] \Rightarrow ax > bx$,*

5. *$[a > b$ and $b \geq c] \Rightarrow a > c$, and $[a \geq b$ and $b > c] \Rightarrow a > c$,*

6. *$[a > b$ and $a \oplus x = b \oplus x] \Rightarrow x \geq a$,*

7. *$[a > b$ and $a \oplus x = b \oplus x] \Rightarrow x > b$,*

8. *$e > a \Rightarrow a^* = e$,*

Proof:

1. This is true for all dioids and follows directly from the definition of $\geq$ (Equation 5.10).

2. Since $x \neq \epsilon$, we know it has an inverse and therefore we get

$$\begin{aligned} ax &= bx && \text{(B.3.2)} \\ axx^{-1} &= bxx^{-1} && \text{(B.3.3)} \\ a &= b && \text{(B.3.4)} \end{aligned}$$

3. This follows directly from Lemma B.3.1(2) since by definition, $a > b \Rightarrow a \neq b$.

4. Lemma B.3.1(3) tells us that if $a > b$ and $x \neq \epsilon$, it must be the case that $ax \geq bx$. It also indicates that $ax \neq bx$ and we therefore know that $ax > bx$.

5. If we assume $c \geq a$ in either case, we get

$$a \geq b \geq c \geq a, \tag{B.3.5}$$

which yields

$$a = b = c, \tag{B.3.6}$$

in contradiction to both formula's antecedents.

6. Suppose that the antecedent holds, but that $a > x$. Then by definition of ">", we have
$$a = a \oplus x. \tag{B.3.7}$$
If we combine this with the antecedent $a \oplus x = b \oplus x$, we get
$$a = b \oplus x, \tag{B.3.8}$$
which by the total order property indicates either $a = b$ or $a = x$. The first contradicts our given and the second contradicts our assumption that $a > x$, and therefore $x \geq a$.

7. This follows immediately from Lemma B.3.1(6).

8. If $a = \epsilon$, then by the definition of closure, we have that a^* solves the equation $x = \epsilon \oplus e$ and therefore $a^* = e$. Otherwise, since $e > a$, by Lemma B.3.1(4) we have
$$a^i > a^{i+1} \tag{B.3.9}$$
for any finite $i \geq 0$. Thus we have
$$x > ax \tag{B.3.10}$$
for any $x \neq \epsilon$. Now a^* is the solution to $x = ax \oplus e$, but $x = \epsilon$ is impossible since by the definition of $\geq$, $x \geq e$. Thus Equation B.3.10 yields
$$x = e. \tag{B.3.11}$$

□

Definition B.3.2 Linear pseudoring expressions, **linear pseudoring equations**, *and* **linear pseudoring systems** *are defined to be, respectively linear dioid expressions (Equation 5.15), linear dioid equations (Equation 5.20), linear dioid systems (Equations 5.31 and 5.32), with entries taken from a pseudoring. A linear pseudoring equation in* **canonical form** *is a pseudoring equation of the form*
$$\left[\bigoplus_{i=0}^{i=n-1} a_i \otimes x_i\right] \oplus b = \left[\bigoplus_{i=0}^{i=n-1} c_i \otimes x_i\right] \oplus d \tag{B.3.12}$$

where if $a_i \neq c_i$ then either $a_i = \epsilon$ or $c_i = \epsilon$, and similarly if $b \neq d$ then either $b = \epsilon$ or $d = \epsilon$.

That this form is achievable for the max-plus algebra is given in [BCOQ92]. It is proved below for the more general pseudoring case.

Proposition B.3.1 *Any linear pseudoring equation can be represented in the canonical form of Equation B.3.12.*

This proposition is proved by the following lemmas.

Lemma B.3.2 *For a, b, c, and x in pseudoring $\mathcal{D}$*

$$ax \oplus c = c \quad \Rightarrow \quad ax \oplus bx \oplus c = bx \oplus c. \tag{B.3.13}$$

Proof: This follows directly by $\oplus$-ing a term bx to either side of the original equation. □

Lemma B.3.3 *For a, b, c, and x in pseudoring $\mathcal{D}$*

$$[ax \oplus bx \oplus c = bx \oplus c \text{ and } a > b] \quad \Rightarrow \quad ax \oplus c = c. \tag{B.3.14}$$

Proof: Since the proof is trivial when $x = \epsilon$, we need only prove it for $x \neq \epsilon$. To do this, we assume that $ax \oplus c \neq c$ and show that this leads to a contradiction. For $ax \oplus c \neq c$, we must either have

- $ax \oplus c < c$ which is impossible since it violates $ax \oplus c = c$, or
- $ax \oplus c > c$. We then note that the antecedent may be written

  $$(ax \oplus c) \oplus bx = c \oplus bx. \tag{B.3.15}$$

  We apply Lemma B.3.1(6), yielding

  $$bx \geq ax \oplus c, \tag{B.3.16}$$

  and therefore

  $$bx \geq ax. \tag{B.3.17}$$

  However, since $a > b$, Lemma B.3.1(1) requires that $ax \geq bx$, indicating that $ax = bx$, in direct opposition to the operation of Lemma B.3.1(3), the fact that $a > b$, and our assumption that $x \neq \epsilon$. Thus $ax \oplus c$ must equal c

□

Lemma B.3.4 *Any equation $ax \oplus c = bx \oplus c$ for which $a > b$ is equivalent to the equation $ax \oplus c = c$.*

Proof: This is a direct result of Lemmas B.3.2 and B.3.3. □

Vita

Elizabeth A. Walkup was born in Seattle, Washington. She attended the University of California, San Diego from 1984 through 1989, and graduated with Bachelor of Arts degrees in Computer Science and Theatre. She received a Master of Science degree in Computer Science from the University of Washington in 1993.