MULTIPROCESSOR SYSTEM DESIGN VALIDATION USING A TEMPORAL DEDUCTIVE SYSTEM.

by

Patric Michael Xavier Yurkowski

A thesis
presented to the University of Manitoba
in partial fulfillment of the
requirements for the degree of
Master of Science
in
Computer Science

Winnipeg, Manitoba

ISBN 0-315-47926-4

MULTIPROCESSOR SYSTEM DESIGN VALIDATION USING A TEMPORAL DEDUCTIVE SYSTEM

BY

PATRIC MICHAEL XAVIER YURKOWSKI

A thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

Copyright Notice

MULTIPROCESSOR SYSTEM DESIGN VALIDATION USING A TEMPORAL DEDUCTIVE SYSTEM.

by

Patric Michael Xavier Yurkowski

A dissertation submitted to the Faculty of Graduate Studies at the University of Manitoba in partial fulfillment of the requirements of

Master of Science

ABSTRACT

Multiprocessor computer systems pose complex design problems. A system design methodology to specify and validate abstract multiprocessor system designs at the architectural level is presented. A subclass of the general Petri net definition is used to specify functionally equivalent validation models of the multiprocessor systems' concurrent nature. A temporal analysis technique is described that converts the validation models into relation structures conforming to the linear, discrete, and absolute view of time. Consequently, Propositional Linear Temporal Logic can be used to state safety and liveness correctness criteria. Validation consists of showing that the PLTL formulae stating the correctness criteria are tautologies with respect to the relation structures. The proofs of logical consistency and completeness ensure that the validation method is sound. This methodology provides the theoretical basis of a computer assisted design system for multiprocessors.

ACKNOWLEDGEMENTS

I would like to acknowledge the assistance of the following people.

My parents encouraged me to strive for higher academic achievements. They continued to have faith in me and provided financial support, even after all those years.

My thesis advisor was Dr. C. M. Laucht; he was very patient throughout the many years it took to me write this dissertation; he also was very generous in providing text-editing facilities, even at the expense of his own access to the machines. Thank you to the thesis examiners Dr. W. L. Kocay and Dr. D. Cowan. Thank you to Dr. D. H. Scuse for giving me the desire to write correctly and for his academic advice.

Thank you to Andrew Ostrander for letting me cry on his shoulder now and again. Finally, I would like to thank all my friends, especially Greg Boettcher, who bought beer for me when I couldn't afford it. Hi Jane, Kelly, Lisa, Lynnette, and Roxy.

Zen Buddhism is time,

and modern physics is time.

Marian Mountain - The Zen Environment

TABLE OF CONTENTS

CHAPTER 1 : INTRODUCTION

Section 1-1 : Thesis.

Multiprocessor computer systems pose complex design problems. They are increasingly used in critical applications where failures can have disastrous consequences. Some multiprocessor systems interact with the real world, making decisions in real-time. With *real-time* systems, their inputs can never be predicted, and their timing is dictated by the real world and not by the system designer. Nevertheless, systems must be *reliable*, *safe*, and *execute* without an unacceptable level of risk [BL]. Hence, techniques to design safe and reliable systems are necessary. This dissertation describes a *system design methodology* to *specify* and *validate* abstract multiprocessor *system designs* at the architectural [Tan] level. The methodology is based on the notions of specifying functionally equivalent models of system designs and of providing a means to validate that certain time-related reliability and safety criteria hold for these *validation models*. The premise is that if the criteria hold for the validation models, it can be inferred that the designs are *valid*.

The two ways of expressing system reliability and safety criteria determine the type of system design methodology required. *Performance criteria* require that systems execute functions reliably, safely, and efficiently within measured real-time intervals. The system design methodology must provide a means of predicting system performance by constructing stochastic models of system designs to serve as a basis for automated *simulation*. The simulations provide an understanding of system performance in different situations before the system design is implemented. *Correctness criteria* require that only desirable situations result from system execution. Real-time measurements are not

necessary, since correctness criteria are expressed in terms of states and sequences of states, with no concern for the duration of time. The system design methodology must provide modelling techniques to show that correctness criteria hold for the system design. A correct implementation of such a system design results in a system honouring the correctness criteria. Although performance issues are a major consideration in selecting the best system design, performance studies are valuable only if applied to a correct system design [Petri77]; therefore, this dissertation concentrates on system design validity in terms of correctness.

Rigorous specification and validation techniques assist the process of *deriving* abstract, valid system designs, i.e. error-free descriptions of design problem solutions that honour the correctness criteria. Valid system designs are an important step towards implementing systems honouring the criteria, because decisions made while specifying system designs affect the success of correctly implementing the designs. However, techniques of ensuring correct implementation of system designs are not considered here.

The methodology presented here guides and assists the process of deriving valid system designs and is intended to be used in a heuristic manner. Systems designs are derived using an indeterminate process of *composition*, by combining validated sub-designs into larger sub-designs, which are in turn validated. Although selection of designs for validation is indeterminate, the validation technique is a proof method applied to new prospective designs in an iterative manner until an acceptable design is achieved. The validation techniques essentially prove that the criteria hold for validation models of prospective designs. Hence, the inference that if the criteria hold for the validation models the criteria hold for the system designs depends on whether the validation models are accurate representations of the system design.

The methodology is used to *validate* that correctness criteria hold for abstract multiprocessor system designs. The multiprocessor system designs and their concurrent

nature are modelled using a subclass of *Petri nets*. Through *reachability analysis* the Petri net models are reduced to *state-based transition models*. The state based transition models are interpreted in terms of an abstract view of time so that correctness criteria may be expressed using *Propositional Linear Temporal Logic*, PLTL, formulae. By showing that the correctness criteria formulae are tautologies with respect to the models, the system designs are deemed *correct*, i.e. valid with respect to correctness.

Section 1-2 : Multiprocessor Systems.

This dissertation considers the problem of designing *multiprocessor systems*. There are two general multiprocessor system models [GP], illustrated in Figure 1-1.

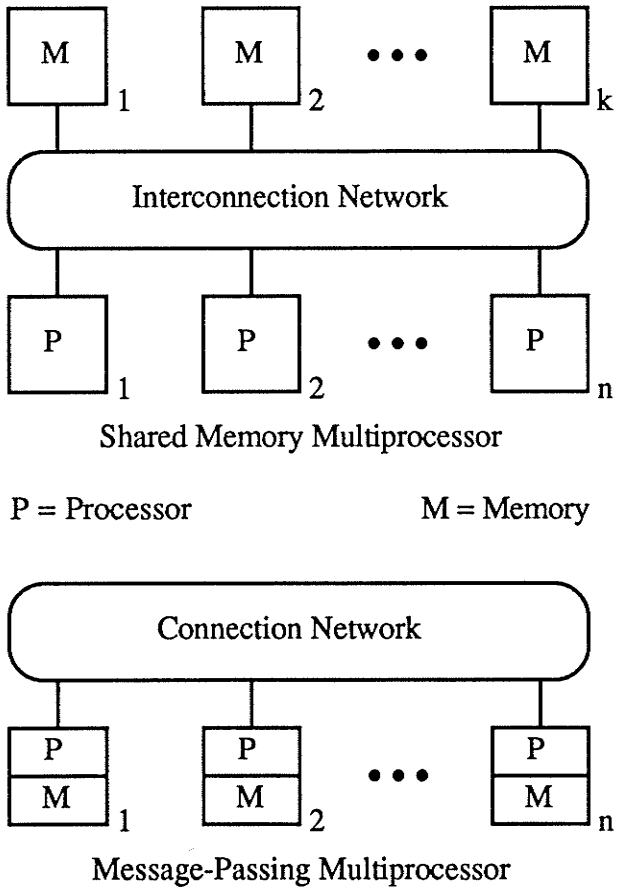


Figure 1-1 : Multiprocessor Models

A *shared memory* multiprocessor consists of numerous distributed processors that communicate using an interconnection network and common memory for data storage. A *message-passing* multiprocessor consists of numerous distributed processors, each with their own local memory, that communicate using messages passed through a connection

network. By maximizing system throughput, these systems are a cost-effective means of increasing computing power and speed. The methodology presented here is applicable to either multiprocessor model.

The problems of designing correct multiprocessor systems are complex. In general, multiprocessor systems consist of numerous *concurrently* executing *system components* that interact. Accordingly, the closely related design issues of *communication*, *data coherence*, and the *ordering of events* among system components must be considered [DSB]. Communication is the exchange of data among the system components using functions provided by *communication protocols* [PZ]. *Synchronization* is a subclass of communication where exchanged data are control information to enforce data coherence and ordering of events. Communication can be used to implement synchronization and vice versa. Data coherence is the software system's ability to correctly access data. For example, it may be incorrect for one component to read data while another component is writing the data. The ordering of events refers to the proper sequencing of data accesses and ordering of synchronization and communication primitives. *Strongly ordered* systems have all components continually execute in a globally specified order. *Weakly ordered* systems do not impose a global execution ordering, except for data accesses and when issuing synchronization and communication primitives.

Section 1-3 : Related Past Work.

Although this dissertation develops a modelling technique strictly for validation purposes, many system and system design modelling techniques have been used in the past for different purposes. The techniques are presented in terms of system and system components, but could be extended to their abstract designs. The techniques surveyed here are those capable of modelling concurrency in multiprocessor systems, with and without respect to time. Some of the techniques incorporating time are extensions of techniques that do not incorporate time, thus the latter are discussed first.

The first group of techniques use the language of mathematics, in particular *set theory* [Hal] and *graph theory* [Ber]. Such models are essentially mathematical structures with techniques to analyse the models [Pra].

Techniques that specify system models based on the notion of a *state* are called *state-based transition* models. Each system component model begins execution holding a *local state*, where some function has not been provided. Furthermore, each system component model can never hold more than one state simultaneously. Through interaction defined by *transitions*, the system component model changes to a state where the function has been completed. A transition has an *enabling state*, that must hold before the transition may proceed, and a *resulting state*, that results from the procession of the transition. It may be necessary for a series of transitions to occur, each resulting in an intermediate state, before the function is completed. Since a system model is a collection of system component models, the system component models' local states are combined into a *global state* of the system model. Like the system component models, multiple global states can never be held simultaneously and a sequence of global states may be held by the system model. Thus, changes to state-based transition models are expressed through changes to the global states.

The state-based transition models provide the method of *state exploration verification* [Wes]. Verification begins with determining which system characteristics are to be demonstrated. Verification requires that the state-based transition models contain enough information to describe the execution of the system. Some arbitrary global state is chosen as the *initial state* of the model and, beginning with that initial state, the set of state sequences is generated by a procession of transitions. All the states *reachable* by these state sequences are arranged into a *reachability state graph.* Each individual reachable state is analysed to determine if it represents an error condition such as deadlock or an incomplete design. Other types of errors, such as unwanted infinite loops, are identified by analysing state sequences. Verification is achieved by comparing the (un)desirable characteristics with the results of the state exploration.

As West points out, some problems of performing state exploration verification on state-based transition models must be solved. First, given that communicating system components execute concurrently, the global state phenomena exhibited by the system must be reflected in the global state specified by the model. Modelling a global state may not be a trivial task. Second, correctness criteria must be defined and tested by the verification technique. The state-based models defined do not necessarily give useful means of solving this problem; furthermore, this verification technique does not yet have a notion of time to reflect the temporal characteristics of the systems. Third, the efficiency of the verification procedure must be considered. To enumerate state sequences and then to determine if the correctness criteria are honoured by the sequences can be impractical in terms of the time and space computing requirements; moreover, there is the added complexity of infinite state sequences. Complex system models have a large number of states; this difficulty is known as the *state space explosion.*

The main causes of state space explosion in system models and their possible solutions are as follows. Communication among components may have many possible

permutations. The analysis of all these state sequences requires a lot of time and space, since there are usually many more state sequences than individual states. An investigation of all possible communication sequences may be unnecessary, but the minimal set of communication sequences that need to be investigated must be determined. The size of communication sequences can be unbounded. This problem is solved by setting a bound on the size of communication sequences. Similarly, a system may reference sequence numbers and variables having a large number of possible values, e.g. a *buffer* and a *window* [Tan]; again, assigning bounds to the domain of the sequence numbers and variables can solve this problem. A common means of solving a problem is to *decompose* it into smaller more manageable sub-problems. Problem decomposition can reduce the number of states and state sequences to be considered at one time. Moreover, there are natural points that partition models into *phases* [CGL]. Each phase can be verified separately, but does not allow the detection of errors which cross over phase boundaries. Given that a large state-space is unavoidable, it is desirable that each state within a sequence be analysed only once. This can be achieved by re-forming the reachability state graph into a *tree* structure[1], and employing a depth-first or breadth-first analysis of the tree.

There are many state-based transition models. A common state-based transition model is a *finite state automaton* [HU][RS]. Many techniques employing finite state automata have been suggested, including a theory of infinite sequences of states generated by finite state automata [Cho][McN][Per][Lau]. A number of other techniques are available to specify concurrent processing [Peter81]. They include *finite state machines* and their extensions, *Petri nets* and their extensions, *marked graphs*, *computation graphs*, *P / V systems*, *message systems*, *UCLA graphs*, *vector addition*

[1] Or a *lattice* [Pra]; a tree is defined below.

systems, and *vector replacement systems*. A detailed discussion of these techniques can be found in the references cited in [Peter81].

Some institutions have expended considerable resources to employ *extended finite state machines* as the basis for formal communication protocol specification techniques [Rudin]. *Comité Consultatif Internationale de Télégraphique et Téléphonique*, CCITT, has developed the *Specification and Description Language*, SDL, to specify *X.21* and *X.25* definitions [Tan]. Subgroups of *International Standards Organization*, ISO are developing *Formal Description Techniques*, FDTs, for communication protocol definitions in their *Open Systems Interconnect*, OSI, architecture. Subgroup A is concerned with the OSI architectural concepts relevant to the FDTs. Subgroup B is developing *Extended State Transition Language*, ESTL, based on Pascal, to assist the definition of finite state machines. IBM is using *Format and Protocol Language*, FAPL, for defining protocols in *Systems Network Architecture*, SNA; this language is based on PL / I and contains constructs for finite state machines.

The second group of techniques use hybrid languages to specify designs. The notation of *Communicating Sequential Processes*, CSP, by Hoare [Hoa78], extends his earlier work on the axiomatic basis for computer programming [Hoa69]. This notation uses a Pascal-like syntax, which includes the commands to specify parallel executing system components and input and output through communication channels. Although CSP is sufficiently abstract to serve as a design tool, it also serves as a foundation of some techniques described below. The *Calculus of Communicating Systems*, CCS, [Mil] is based on the notion of describing a concurrent system with the behaviour experienced by an external observer. The CCS algebra, with the binary operation of *concurrent composition*, describes the behaviour dynamics with *derivations*. The behaviour *derivatives* are based on the premise that observation and communication are

equivalent. Distinct from the mathematical notations above, CSP and CCS use a high-level notation for system models.

Numerous proposals to incorporate time with modelling techniques have been suggested [CPW]; detailed discussions are found in the references cited there. Without time, the theoretical basis for correctness analysis is restricted to investigation of individual model states for errors such as deadlock. One advantage of incorporating time is a justification for the meaning of state sequences, that is, the execution of a system over time. Furthermore, explicit models of time-dependent multiprocessor details, such as time intervals, data transfer delays, and timeouts, are possible.

It seems reasonable to interpret a model based on *global time*, by envisioning a global clock that all communicating system components can read. Unfortunately, co-ordinating the components is impractical, since each component would have a different view of time because of variable delays when reading the clock. Another approach is to establish a local clock for each component. Since no two clocks are identical and each clock will operate at minutely different rates, synchronization of the clocks is mandatory. Physical synchronization of the clocks can be expensive and may not be perfect. To solve this problem, Lamport [Lam78] suggests *logical time*. Logical time is based on the notion that, for an event A to influence an event B, event A must happen before event B. This concept is the basis for an algorithm to synchronize local clocks with respect to global time. Morgan [CPW] showed that it is possible to specify a system design with global time, but implement the design in terms of logical time.

State-based transition models incorporate the concept of logical time in the following manner. Logical time is modelled by associating time variables with each communicating component; when a transition proceeds, each component's time variable is incremented by a particular time value associated with the transition. But before a transition can proceed, the transition enabling state must hold for a minimum time period

T_{min}. Furthermore, an enabling state can only hold for a maximum time period T_{max} before the enabled transition must proceed. Each local time must be kept close to a single global time, within certain defined tolerances. If it is necessary to ensure that certain transitions occur within a certain time period, then incrementing the time variables must be strictly controlled.

Some examples of state-based modelling techniques incorporating logical time are given here. Finite state automata can be defined in terms of logical time. The *coupled time graph*[2] technique uses finite state machines to specify a graph model of a system component design, for each component and the [inter]connection network. The vertices of the graphs model the component states, while the arcs of the graph model the transitions among the states. A time variable is associated with each component graph model, so that logical time may be represented. Interaction among communicating components is represented by coupling selected transitions from each graph model that must proceed together. Analysis is achieved by network flow analysis of graph theory. *Timed Petri nets* are Petri net extensions that adopt logical time by associating T_{min} and T_{max} with their transitions. These techniques have sufficient power to specify system models, but for large models, the state space explosion makes analysis impractical.

ISO Subgroup C is developing *Language Of Temporal Ordering Specifications*, LOTOS, as an FDT for OSI. LOTOS is an executable specification language comprised of CCS and an abstract data type technique ACT-ONE [BB]. A major advantage of LOTOS is its means of step-wise refinement of a design problem. Although primarily designed for protocols, LOTOS can be used for multiprocessor systems in general.

Two other miscellaneous approaches are *Real-Time Attribute Grammars* and *Concurrent State Deltas*. The former technique provides quick specification and testing of

[2] A formal definition of a graph is presented below.

system design prototypes. It is a grammar that allows for top down definitions of system models, and that has the expressive power to specify time. A transition represents the sending and receiving of communication data, represented by a terminal symbol in the grammar. A design specified with the grammar is tested by a parser which determines the semantics of the system design. The weakness of real-time attribute grammars is that they lack analytical power and have no means to formally verify the system models. The latter approach was developed strictly for shared memory multiprocessors based on the concept of communication through *shared variables*. A segment of concurrent execution is specified by a enabling condition, a set of shared variables that can be read, a set of shared variables that can be written on, a time it takes for the computation to complete, and a resulting condition. Thus, if an enabling condition is held within the computation time, a system will end up in a state where the resulting condition holds. This modelling technique has limitations in specifying certain concepts of asynchronous processes, and are not well suited to message-passing multiprocessors.

Temporal logic [HC][RS][McA] is a form of modal logic orginally developed by philosophers that can express temporal relations. Temporal logics are extensions of classical *First Order Logic* [Smu] by the addition of modal operators with temporal meanings. The definition of time is implicit in the usage of temporal logic formulae. There are two types of temporal logic, *branching* temporal logic based on a tree structure of time and *linear* temporal logic based on a determinate structure of time. An interesting debate about whether the linear view of time is better for expressing system design correctness criteria than the branching view of time is described in [BMP][EH82][EH83][EL]. Although each view of time has its advantages and disadvantages, both forms of temporal logic have equivalent expressive powers. The bulk of temporal logic research in multiprocessor system design, especially protocols, has involved the linear form. Linear temporal logic is a language that can express correctness

criteria based on *logical models* [3] consisting of possibly infinite sequences of states; hence, a system model must be reduced to form sequences of global states generated from each communicating system component. By synchronizing the communicating components in terms of some *concurrent execution discipline* [LPS], the allowable sequences are constrained.

Temporal logics are a convenient language to express certain time-dependant correctness criteria [Hai][HO][Lam83iii][MP79][MP81i][MP81ii] [MP83][SM-S]. *Safety* criteria refer to properties which are invariant over the execution of the system, and once proved ensure that nothing bad ever happens. *Liveness* criteria refer to properties that must eventually hold and usually state that something good will happen. Of course, these properties are stated in more precise application orientated statements. Safety and liveness criteria can be stated without explicit *real-time* measurements, thus, because of their implicit time definition, temporal logics are well suited for expression of these criteria. To show that these criteria hold for a system model, it is necessary to show that the correctness criteria formulae are *tautologies* with respect to the models.

Pnueli introduced temporal logic as a logical system for use with Hoare's program axioms, CSP, and *monitor* concept [Hoa74][LP][MP79][MP81i][MP81ii] [MP83][Pnu]. For example, the authors of [Hai][HO] apply temporal logic and Hoare's axiomatic system to concurrent processing. Other groups have pursued research in temporal logic [CE][EH82][EH83][EL][Eme][ES][GPSS][GS] [Lam80][Lam83iii][LPS][SC][Wol][SM-S]. Because programming language models can be reduced to state-based models [MP81i], some research has

[3] A model as in model theory [Sac].

used Temporal logic with state-based transition models [AEIKOT][CES][DGU] [Sis]; this research owes much to the fundamental result achieved by Rabin [Rab].

Section 1-4 : The Work Presented Here.

This dissertation defines a technique for validating that safety and liveness correctness criteria hold for abstract multiprocessor system designs. The technique consists of specifying a validation model of a system design with a Petri net, interpreting the validation model with respect to an abstract concept of time, and using linear temporal logic to state safety and liveness criteria; showing that safety and liveness criteria formulae are tautologies with respect to the validation models validates the system design. It is intended that this validation technique be used as part of a heuristic approach for multiprocessor system design.

A state-based transition model is used to specify the concurrent nature of the multiprocessor system design. Each system component is modelled with a finite state machine. A *coupled graph* [CGL] is used to combine the system component models into a model of a complete system design, however network flow analysis of graph theory is not used. Coupled graphs are a subclass of Petri nets. Using Petri net *reachability analysis* [Peter81], the system design model is reduced to a *reachability graph*, a finite state machine. The reachability graph serves as the foundation for the validation technique.

Because temporal logics are appropriate for stating safety and liveness criteria, *Propositional Linear Temporal Logic*, PLTL, is used. The PLTL definitions are based on the work by Pnueli and his associates [Pnu][MP79][MP81i][MP81ii][GPSS]. A *semantic tableaux procedure* based on [Kri59][Kri63][RU] determines if a *PLTL formula* is true for a given reachability graph. A *PLTL formal deductive system* is defined here, based on the concepts of model theory [Bri][Sac], *modal normal form formulae* [Fine], and Henkin Proofs [HC] to show *consistency* and *completeness*. The deductive system allows tautologies to be derived in a syntactic manner.

Hence, because temporal logic and Petri nets are used together, this approach is similar to that of [AEIKOT][DGU]. However, the authors of [AEIKOT] describe the Petri nets and their transition firings with temporal logic. Instead, this approach attaches propositions to reachability graph vertices and interprets the PLTL formulae with respect to sequences of reachability graph vertices. Like this dissertation, [DGU] develops a semantic method, an axiomatic approach, and defines safety and liveness criteria. However, [DGU] uses first order logic as the basis for the temporal logic, whereas the approach used here takes *propositional logic* as a basis. The axiomatic basis of [DGU] is different from the one used here.

Section 1-5 : Outline.

Chapter 2 defines the basic notations and symbols, gives some preliminary definitions, and defines the concept of a language.

Chapter 3 defines a subclass of General Petri nets, to specify *validation models*. The validation models are composed of *system component models* concurrently holding a local state. Each system component model is a finite state machine expressed as a directed graph, whose vertices represent the local states and whose arcs represent the relation among the local states. The validation model holds a global state composed of all concurrent system component model local states. Interaction among the system component models determines the relation among global states; reachability analysis provides a means to determine this relation. The *reachability graph algorithm* converts the validation model Petri net into a strongly connected directed reachability graph, whose vertices represent global states and whose arcs represent the global state relation. Propositions are mapped to the global states to more precisely define the meaning of a state.

Chapter 4 defines a temporal analysis technique, that converts the strongly connected, directed reachability graph to a structure conforming to the *linear, discrete, and absolute view of time* required for PLTL. The execution of a validation model is represented by constructing the set of distinct finite and infinite walks that begin with a particular vertex in the reachability graph, where each walk represents one possible sequence of global states over time. Standards of conduct define the *α-relation*, which select a subset of all countably infinite walks. This set of infinite walks is condensed into a collection of finite walks, that contains sufficient information to specify the maximal strongly connected reachability graph component specified by the α-relation. The information can be expressed as a family of predicates, whose sentences correspond to the

finite walks. The relation structure is defined from the predicates and forms a structure of finite sequences of propositions providing the basis for the PLTL definitions.

Chapter 5 defines the PLTL language. The syntax selects words from the PLTL language which can be assigned meaning. The temporal semantics determine whether a word asserts something true in a given sequence of propositions. The semantic tableaux procedure tests if a word is valid in the relation structure. A proof is supplied to show that the semantic tableaux procedure does what is claimed.

Chapter 6 defines the *PLTL deductive system.* The *terms* of the deductive system correspond to the validation model states. A set of axiom schemata define the *theses* of the deductive system. The relation structure is expressed with a set of *hypotheses.* Together, the terms, theses, and hypotheses define the axiomatic basis for the PLTL deductive system. The process of deduction derives a set of theorems from the axiomatic basis; the theorems are said to be consistent. The *maximal consistent set* of all theorems is defined. A proof is given that the deductive system is complete if, and only if, deductive system does not derive inconsistent formulae.

Chapter 7 ends this dissertation with some conclusions.

CHAPTER 2 : PRELIMINARIES

Section 2-1 : Basic Notation and Symbols.

The following is a summary of the mathematical and logical symbols used in this dissertation.

$\mathbb{N}$	the natural numbers 1, 2, 3, ...	$\mathbb{Z}^+$	$\mathbb{Z}^+ =_{\mathbf{df}} \mathbb{N} \cup \{ 0 \}$
$\aleph$	cardinal numbers	$\aleph_0$	smallest transfinite cardinal number
ω	infinitely many	$\exists^\infty$	there are infinitely many
$\ni$	such that	:	satisfying the condition that
$\| X \|$	cardinality of set X	$=_{\mathbf{df}}$	is defined as
$=$	a comparison of equality	$:=$	assignment
$\Leftrightarrow$	if, and only if ; abbr. iff	$\Rightarrow$	implies
$\times$	Cartesian product	$\rightarrow$	a mapping between two sets
$\cup$	union	$\cap$	intersection
$\in$	is included in	$\notin$	is not included in
$\subset$	proper containment	$\subseteq$	containment
$\geq$	greater than or equal to	$\leq$	less than or equal to
$>$	greater than	$<$	less than
$\varnothing$	empty set	$\neq$	not equal to
$\exists$	there exists	$\forall$	for every
$\wedge$	conjunction	$\vee$	disjunction
$\neg$	negation	$\| \sigma \|$	length of a sequence σ

It is assumed that the reader is familiar with basic concepts from set theory and First Order Logic, that can be found in references such as [Bri][Hal][Smu].

Definitions are associations of terms with meanings; they are made by listing some term notation, the symbol **=df**, and an explanation of the term. The term notation and explanation can be any form, prose, set theory, First Order Logic, etc. For example, a set X is defined using the notation X **=df** { x ∋ *condition of membership* }, read as X is defined as the set of x such that the condition of membership holds. The dissertation adopts the convention that sets are denoted with upper case letters, while the elements of the set are denoted with lower case letters. Sometimes COM(x) is used as an arbitrary condition of membership. Some structures are defined as tuples of distinct sub-structures; for example, S **=df** < A, B, C > defines a structure S composed of three distinct sub-structures A, B, and C.

This dissertation employs the sets of *natural* $\mathbb{N}$, *ordinal* ORD, and *cardinal* $\aleph$ numbers [Hal]. The set of natural numbers is denoted $\mathbb{N}$ **=df** { 1, 2, ... , i, ... }; a finite subset $I \subseteq \mathbb{N}$ is denoted I **=df** { 1, 2, ... , i }. Greek letters α, β, γ, ... are used to denote ordinal numbers. The symbol ω is the smallest transfinite ordinal number; the symbol ω is read as "infinitely many". A set X is *countably finite* if X is equivalent to a subset of $\mathbb{N}$; X is *countably infinite* if X is equivalent to $\mathbb{N}$. A finite cardinal number is denoted with the equivalent natural number, the smallest transfinite cardinal number is denoted $\aleph_0$, viz. $\aleph_0$ **=df** $|\mathbb{N}|$, and other transfinite cardinals are denoted $\aleph'$ with the fact $\aleph' > \aleph_0$ explicitly stated. To define a set with cardinality of $\aleph_0$, condition of membership statements of the form $\exists_\omega(x)$: COM(x) are used; the statement is read "there exists infinitely many x satisfying the condition that COM(x) is true"; i.e., $|\{ x \ni COM(x) \}| = \aleph_0$ [CV]. In the few cases when *zero* is required, $\mathbb{Z}^+$ is used.

Functions may be *injective*, *surjective*, and *bijective* [Pra]. An arbitrary function is written A : B → C, which is interpreted as "function A is a mapping from set

B to set C". X^Y denotes a mapping from the set Y to the set X. The notation 2^P describes a mapping from the set ***P*** to the *field* denoted by 2, viz. { 0, 1 }, and denotes the power set.

A number of definitions are in the form of Cartesian products and the tuples they define. A function $A : B \rightarrow C$ is a subset of the Cartesian product $B \times C$. An index set I of a Cartesian product is some subset of $\mathbb{N}$. In general, given the finite collection of sets $S_1, S_2, \ldots, S_n$, indexed by the set I, the Cartesian product $S_1 \times S_2 \times \ldots \times S_n =$**df** $\{ (s_1, s_2, \ldots, s_n) \ni s_i \in S_i \text{ for } i \in I \subset \mathbb{N} \}$. It is possible to define an infinite Cartesian product, $S_1 \times S_2 \times \ldots \times S_n \times \ldots =$**df** $\{ (s_1, s_2, \ldots, s_n, \ldots) \ni s_i \in S_i \text{ for } i \in \mathbb{N} \}$. When an n-tuple represents a vector or a Cartesian product X^n, it is denoted $(x_1, x_2, \ldots, x_n)$.

Relations and *predicates* are used in this dissertation. Relations are defined over a set X of elements. Given $x \in X$, the *degree* of the relation **R** is $| \{ x' \ni x \mathbf{R} x' \} |$. One type of relation in particular is required.

Definition 2-1-1 : Equivalence Relation

A relation R is an *equivalence relation* over a set X iff, for elements $x, y, z \in X$:

i) $x R x$ *reflexitivity*;

ii) $x R y \Rightarrow y R x$ *symmetry*;

iii) $x R y$ and $y R z \Rightarrow x R z$ *transitivity*.

Predicates are n-ary relations and are denoted $R(x_1, x_2, \ldots, x_n)$, $x_i \in X$, $1 \leq i \leq n$. $x_1, x_2, \ldots, x_n$ are called *variables* of the predicate. A *sentence* of a predicate $R(x_1, x_2, \ldots, x_n)$ is an n-tuple $(y_1, y_2, \ldots, y_n)$.

Section 2-2 : Preliminary Definitions.

Some preliminary definitions are required [CV]. X is used to denote some arbitrary set. A fundamental mathematical structure used in this dissertation is a *sequence*.

Definition 2-2-1 : Sequence

For the *index set* $I = \{ 0, 1, \ldots, \alpha - 1 \}$, $\alpha \in$ ORD, a *sequence* in a set X is a surjective mapping $v : I \rightarrow X$, where :

i) The *length* of v, denoted $| v |$, is the cardinality of the index set I; i.e., $| v | =$**df** $| I |$;

ii) There may be countably *finite sequences* $v_\circ$, where $| v_\circ | < \aleph_0$, countably *infinite sequences* v_ω, where $| v_\omega | = \aleph_0$, and *transfinite sequences* v_∞, where $| v_\infty | = \aleph'$, $\aleph' > \aleph_0$;

iii) Furthermore, the set of all countably finite sequences $v_\circ$ in X is denoted X^*, the set of all countably infinite sequences v_ω in X is denoted X^ω, and the set of all transfinite sequences v_∞ is denoted X^∞;

iv) The notation $v_\alpha =$**df** $(x_0, x_1, \ldots, x_{\alpha - 1})$, $\alpha \in$ ORD, explicitly describes the *components* of a sequence of length $| v_\alpha |$ in X;

v) The *β-th component* of a sequence v_α, $\beta < \alpha$, is denoted $v_\alpha(\beta) =$**df** x_β for the sequence $v_\alpha =$**df** $(x_0, x_1, \ldots, x_\beta, \ldots, x_{\alpha - 1})$;

vi) For a sequence v_α, $v_\alpha[\beta]$ denotes the *prefix* of length β, $\beta < \alpha$, a restriction of the domain of sequence v_α, $v_\alpha[\beta] : \{ 0, \ldots, \beta - 1 \} \rightarrow X$, defining the tuple $v_\alpha[\beta] =$**df** $(x_0, x_1, \ldots, x_{\beta - 1})$, where for every $\gamma < \beta$, $v_\alpha[\beta](\gamma) = v_\alpha(\gamma)$;

vii) The *empty prefix* $\lambda =$**df** $v[0]$;

viii) The *portion* of length γ of a sequence v_α, $\beta + \gamma < \alpha$, is a sequence denoted $v_\alpha[\beta, \gamma] =$**df** $(x_\beta, x_{\beta + 1}, \ldots, x_{\beta + \gamma})$;

ix) A sequence is a structure $< v_\alpha, < >$ where the relation $<$ is a well order over the components of the sequence $(x_0, x_1, \ldots , x_{\alpha - 1})$ defined by the ordinal indices; i.e., $v_\alpha(\beta) < v_\alpha(\gamma) \Leftrightarrow \gamma < \beta$.

This dissertation adopts the convention of using natural numbers for the index sets of countable sequences. A countably finite sequence $v_\circ$ indexed with natural numbers $\{ 1, 2, \ldots , i \}$, where $| v_\circ | =$**df** $I \subseteq \mathbb{N}$ is denoted $(x_1, x_2, \ldots , x_i)$. Likewise, a countably infinite sequence $v_\omega =$**df** $(x_1, x_2, \ldots , x_i, \ldots)$. The finite *portion* of length k of a countable sequence v is denoted $v[i, j]$, $1 \leq i \leq j \in \mathbb{N} \ni | v | \in \mathbb{N} \Rightarrow j \leq | v |$, where $k = j - i + 1$, and is the mapping $v[i, j] : \{ i, i + 1, i + 2, \ldots , j - 2, j - 1, j \} \rightarrow X$ defining the tuple $v[i, j] =$**df** $(x_i, x_{i + 1}, \ldots x_{j - 1}, x_j)$. A prefix is a special case where $v[j] =$**df** $v[1, j]$.

Definition 2-2-2 : Infinity Set

The *infinity set* for an countably infinite sequence $v_\omega \in X^\omega$ is

$INF(v_\omega) =$**df** $\{ x \in X \ni \exists_\omega(i) : v_\omega(i) = x \text{ and } i \in \mathbb{N} \}$.

The infinity set contains those elements of X that appear infinitely often in an infinite sequence in X.

The following definition is crucial for the development of linear temporal logic [Bri].

Definition 2-2-3 : Collections of Countable Sequences

Given an arbitrary non-empty collection S of sequences, that may include finite $v_\circ$ and infinite v_ω sequences in a set X :

i) the i-th components of the sequences in S form a set denoted S_i; $\{ S_i \ni i \in I \}$ denotes the collection of S_i indexed by the set $I = \{ 1, 2, \ldots , n \}$;

ii) the *union* of $\{ S_i \ni i \in I \}$, denoted $\cup_{i \in I} S_i$, is $\{ s \ni s \in S_i \text{ for some } i \in I \}$;

iii) the *intersection* of $\{ S_i \ni i \in I \}$, denoted $\cap_{i \in I} S_i$, is

{ $s \ni s \in S_i$ for all $i \in I$ };

iv) the *product* of { $S_i \ni i \in I$ }, denoted $\times_{i \in I} S_i$, is the set of *finite choice functions* { $f_\circ \ni f_\circ : I \rightarrow \cup_{i \in I} S_i$, where $f_\circ(i) \in S_i$ for all $i \in I$ }; for $I = \{ 1, \ldots, n \}$, $n \in \mathbb{N}$, the n-tuple $(f_\circ(1), \ldots, f_\circ(n)) \in S_1 \times \ldots \times S_n$ is a well ordering of the range of some finite choice function $f_\circ : I \rightarrow \cup_{i \in 1, \ldots, n} S_i$;

v) The *product* of { $S_i \ni i \in \mathbb{N}$ }, denoted $\times_{i \in \mathbb{N}} S_i$, is the set of *infinite choice functions* { $f_\omega \ni f_\omega : \mathbb{N} \rightarrow \cup_{i \in \mathbb{N}} S_i$, where $f_\omega(i) \in S_i$ for all $i \in \mathbb{N}$ }; for $I = \mathbb{N}$, the infinite tuple $(f_\omega(1), \ldots, f_\omega(n), \ldots) \in S_1 \times \ldots \times S_n \times \ldots$ is a well ordering of the range of some infinite choice function $f_\omega : I \rightarrow \cup_{i \in 1, \ldots, n, \ldots} S_i$.

The essence of this definition is that a predicate $S_i(v_1, v_2, \ldots, v_i)$, whose variables $v_1, v_2, \ldots, v_i$ are bound by components of the sequences; in fact, by the *Axiom of Choice* [Hal], the function $f_\circ$ (f_ω) selects a distinct sequence in $\cup_{i \in I} S_i$ ($\cup_{i \in \mathbb{N}} S_i$) from $\times_{i \in I} S_i$ ($\times_{i \in \mathbb{N}} S_i$), and defines a sentence of the predicate.

The next mathematical structure is found throughout this dissertation. The definitions and nomenclature are based upon those presented in [Ber].

Definition 2-2-4 : Graph

A *graph* [Ber] is a tuple $G =_{\mathbf{df}} < V, A >$ where :

i) V is a set { $v_1, v_2, \ldots, v_m$ } of *vertices*; where $v_i = v_j$ iff v_i and v_j are the same vertex and $v_i \neq v_j$ iff v_i and v_j are not the same vertex, for $1 \leq i \leq m$ and $1 \leq j \leq m$;

ii) The *order* of the graph is $| V |$;

iii) A is a set { $a_1, a_2, \ldots, a_n$ } of *arcs*, where $a_i \in V \times V$, $1 \leq i \leq n$;

iv) Given an arc $(v_i, v_j) \in A$, v_i and v_j are the *endpoints* of the arc, v_i is the

initial endpoint and the *predecessor* of v_j, and v_j is the *terminal endpoint* and the *successor* of v_i;[1]

v) A *loop* is an arc where the terminal and successor endpoints are the same vertex;

vi) Two arcs are *adjacent* if they have an endpoint in common; two adjacent arcs are *parallel* if their initial and terminal endpoints are the same; two adjacent arcs are *in series* if the predecessor endpoint of one arc is equal to the successor endpoint of the other arc;

vii) The *multiplicity* of a pair of vertices is the number of parallel arcs defined with the vertices; this dissertation is restricted to graphs where the multiplicity of a pair of vertices is not greater than one, called *1-graphs*;

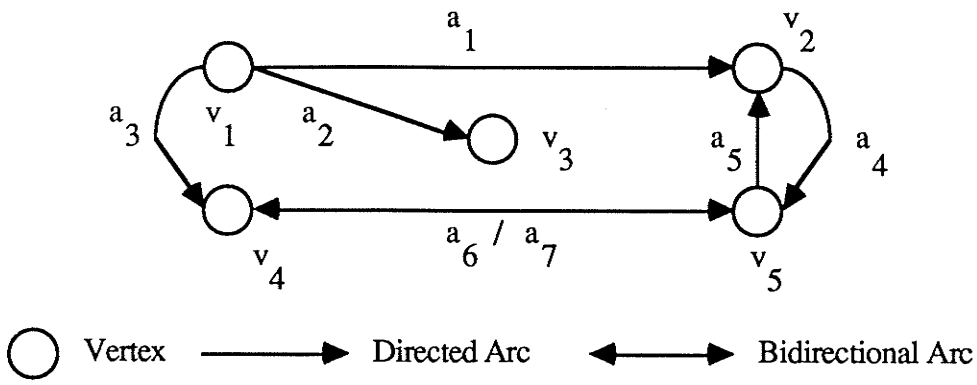
viii) The *degree* of vertex v_i, $1 \leq i \leq m$, is the number of arcs with v_i as an endpoint;

ix) A graph is *regular* if each vertex has the same degree;

x) A *labelled graph* is a 3-tuple $G =\mathbf{df} < V, A, Y >$, where Y is a function $Y : V \rightarrow X$ which associates a vertice with an element of some arbitrary set X.

Figure 2-1 illustrates a 1-graph, labelled with elements of the power set of $P =\mathbf{df} \{ p_1, p_2, p_3 \}$. The order of the graph is five. Arcs a_1 and a_2 are adjacent. This graph has no arcs in parallel. Arcs a_6, a_5, and a_4 are in series. The degree of vertex v_5 is three. This graph is not regular. The two directed arcs a_6 and a_7 form a bidirectional arc. This method of illustrating a graph is used and extended in the sequel.

[1] Since a direction from a predecessor to successor is implied, a *directed graph* has been defined using *directed arcs*. A *bidirectional arc* is shorthand for two arcs (x, y) and (y, x), Figure 2-1.



$a_1 = (v_1, v_2)$ $a_2 = (v_1, v_3)$ $a_3 = (v_1, v_4)$

$a_4 = (v_2, v_5)$ $a_5 = (v_5, v_2)$ $a_6 = (v_4, v_5)$

$a_7 = (v_5, v_4)$

$Y(v_1) = \{ p_1, p_2 \}$ $Y(v_2) = \{ p_1 \}$ $Y(v_3) = \{ p_1, p_3 \}$

$Y(v_4) = \{ p_3 \}$ $Y(v_5) = \varnothing$

Figure 2-1 : A labelled graph

Definition 2-2-5 : Subgraph

A *subgraph* of a graph $< V, A >$ is a tuple $S =$**df** $< V', A' >$ where :

i) $V' \subseteq V$;

ii) $A' \subseteq A$, where any arc $(v_i, v_j) \in A'$ has both $v_i, v_j \in V'$.

Definition 2-2-6 : Chain

A *chain* of length $q > 0$ of a graph $< V, A >$ is a sequence $C = c_1, c_2, \ldots, c_q$ of arcs $c_i \in A$, $1 \leq i \leq q$, where, given that either c_{i-1} and c_{i+1} exist, one endpoint of c_i is the same as an endpoint in c_{i-1} and the other endpoint of c_i is the same as an endpoint in c_{i+1}.

i) The *length* of a chain C is the length of the sequence C; i.e., $| C |$;

ii) An *elementary chain* is a chain that does not encounter the same vertex twice;[2]

iii) A *simple chain* is a chain that does not include the same arc twice;

iv) the *initial endpoint* of C is v_i for $(v_i, v_j) = c_1$ and the *terminal endpoint* of C is v_l for $(v_k, v_l) = c_q$;

v) A *path* is is a chain with all constituent arcs in series; that is, for the chain C, for $i < q$, the terminal endpoint of c_i is the initial endpoint of c_{i+1}; a path with initial endpoint x and terminal endpoint y is called a *path from x to y*; a single vertex is called a path of length 0;

vi) A *cycle* is a chain C where no arc appears twice in the chain and the endpoints of C are the same vertex;

vii) A *pseudo-cycle* is a cycle, but where an arc is encountered more than once;

viii) A *circuit* is a cycle $C = c_1, c_2, \ldots, c_q$ such that the arcs are in series.

Definition 2-2-7 : Connected Graph

A *connected graph* $< V, A >$ is a graph containing a chain for any two distinct vertices $x, y \in V$.

The relation $x \equiv y$ that holds when either $x = y$ or $x \neq y$ with a chain connecting x and y is an equivalence relation.

Definition 2-2-8 : Connected Component of a Graph

The equivalence classes of the relation $\equiv$ partition the graph $< V, A >$ into connected subgraphs, called *connected components*. A graph that is a connected component is called a *connected graph.*

For $x, y \in V$, the relation $x \approx y$ that holds when there exists a path from x to y and a path from y to x is an equivalence relation.

[2] That is, the vertex appears in at most two adjacent arcs.

Definition 2-2-9 : Strongly Connected Component of a Graph

The equivalence classes of the relation $\approx$ partition the graph $< V, A >$ into strongly connected subgraphs, called *strongly connected components*. A graph that is a strongly connected component is called a *strongly connected graph.*

The following portion of the Theorem 7 in Chapter 3 of [Ber] is useful.

Theorem 2-2-10 : Connected Graph Theorem

If $< V, A >$ is a connected graph, the following conditions are equivalent.

i) $< V, A >$ is strongly connected;

ii) every arc lies on a circuit.

The proof of Theorem 1-7-10 can be found in [Ber].

Definition 2-2-11 : Walk

A *walk* of length $q + 1 > 0$ is a sequence of vertices that correspond to a path of length $q > 0$.

The walk v_1, v_2, v_5, v_2, v_5 corresponds to the path, with the arcs explicitly denoted, $C = (v_1, v_2)_1, (v_2, v_5)_2, (v_5, v_2)_3, (v_2, v_5)_4$ of the graph in Figure 2-1. For a 1-graph, a path uniquely determines a walk and vice versa. It is clear that there can be walks of finite and infinite length.

Another fundamental mathematical structure used in this dissertation is a *tree* [Smu]. A tree is a connected 1-graph without cycles.

Definition 2-2-12 : Unordered tree

An *unordered tree* is a tuple $U =\mathbf{df} < N , \mu , Q >$ where :

i) N is a set of *nodes*;

ii) μ is a *level function* $\mu : N \rightarrow \mathbb{N}$, assigning a *level* to the nodes;

iii) $Q \subseteq N \times N$ is a *successor relation* such that :

a) for nodes x and y, if x Q y then y is the *successor* of x and x is the *predecessor* of y;

b) there is a unique node assigned a level of 1, called a *root node*;

c) every node x, except the root node, is a successor of some other node;

d) for nodes x and y, if x Q y then $\mu(y) = \mu(x) + 1$.

A node is called an *end node* if it does not have a successor. A node with one successor is called a *simple node*, while a node with more than one successor is called a *junction node*. A tree where all junction nodes have at most two successors is called a *binary tree*.

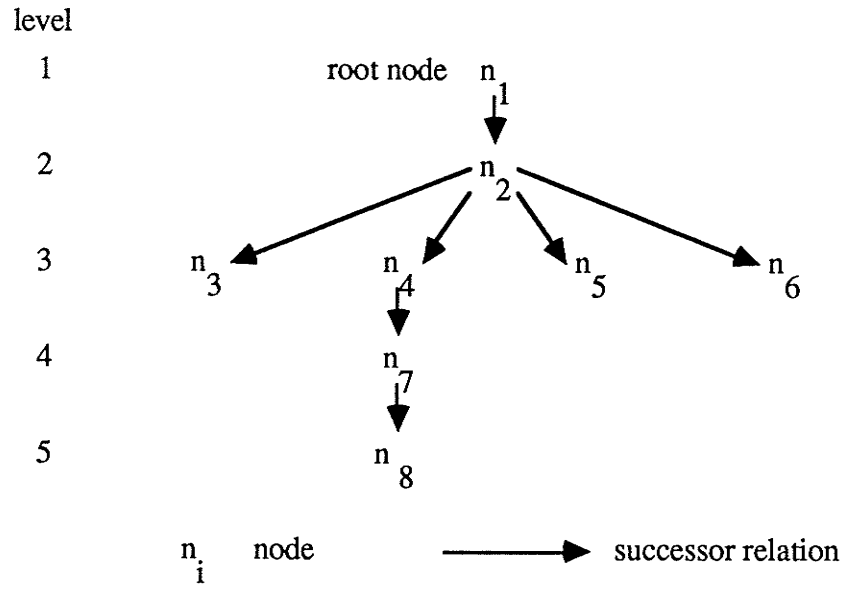
Definition 2-2-13 : Ordered tree

An *ordered tree* is a 4-tuple $T =_{df} < N , \mu , Q , O >$, where N, μ, and Q are as per Definition 1-7-12, and where $O : N \rightarrow \mathbb{N}$ is a injective function that orders the successors of a junction node.

For example, given the successors n_3, n_4, n_5, and n_6 of a node n_2, the function $O : \{ n_3, n_4, n_5, n_6 \} \rightarrow \{ 1, 2, 3, 4 \}$ orders the successors. This dissertation uses the illustration convention that ordering among successors is specified by the order the nodes are drawn on the page, from left to right. The *tree structure* is specified by the defining tuple $< N , \mu , Q >$, or $< N , \mu , Q , O >$. A tree with every node having a finite number of successors is called a *finitely generated tree*. A tree is *finite* if it has a finite number of nodes and a tree is *infinite* if it has an infinite number of nodes.

A *labelled tree* is a ordered or unordered tree with an additional function Y, which associates a node with an element of some set; for example, given the nodes n_3, n_4, n_5 and n_6, one labelling could be $Y(n_3) = a$, $Y(n_4) = b$, $Y(n_5) = c$, and $Y(n_6) = d$. There can be two different labelled trees with the same tree structure. The purpose of a labelled tree is to attach additional information to a tree.

For example, Figure 2-2 illustrates a labelled ordered tree, including some of the examples already presented.



$Y(n_1) = a$ $Y(n_2) = f$ $Y(n_3) = a$ $Y(n_4) = b$

$Y(n_5) = d$ $Y(n_6) = d$ $Y(n_7) = c$ $Y(n_8) = e$

Figure 2-2 : An Ordered Tree

Node n_3 is an end node, node n_4 is a simple node, and node n_2 is a junction node.

Definition 2-2-14 : Tree Path

A *path* in a tree is a countably finite (infinite) sequence of nodes $n_1, n_2, \ldots, n_i$ ($n_1, n_2, \ldots, n_i, \ldots$), such that n_1 is the root noot and n_i, $i > 1$, is the successor of n_{i-1}. A *maximal path* is either a finite path ending with an end node or an infinite path. For any node x in a tree there is a path that ends on x.

For example, in Figure 2-2, the sequence n_1, n_2, n_4, n_7, n_8 is a finite path that ends on n_8; Figure 2-1 does not contain an infinite path.

Lemma 2-2-15 : König's Unendlichkeitslemma

König's *Unendlichkeitslemma* states that a finitely generated infinite tree must contain at least one countably infinite path.

The proof of Lemma 2-2-15 can be found in [Smu].

Section 2-3 : Languages

A *language* is composed of a set of *words*. An *alphabet* is a finite set of *constituents* $X =\mathbf{df} \{ x_1, x_2, \ldots, x_n \}$ that comprise the words. A word is a sequence of constituents. A mechanism can be defined to construct words.

Definition 2-3-1 : Free Monoid

A *free monoid* [Pra] generated by the alphabet X is the tuple $X^* =\mathbf{df} < X^*, \bullet, \Lambda >$, where :

i) X^* is a set of countable sequences in the set X, called *words*;

ii) $\bullet$ is the operation of *concatenation*;

iii) the symbol Λ denotes the *empty word.*

The process of generating words begins with the empty word $\Lambda \in X^*$, where given a constituent $x \in X$, $\Lambda \bullet x \in X^*$. This process may continue ad infinitum; in general, given $\sigma \in X^*$, $\sigma \bullet x \in X^*$. A subset $\Gamma \subseteq X^*$ are selected because meaning can be associated with them and are called *formulae* of the language. The *syntax* defines the criteria for formulae selection and the *semantics* define the meaning of the formulae.

A *lexicographical* ordering, analogous to the ordering of a dictionary, can be defined for words $\Gamma \subseteq X^*$ generated by a free monoid.

Definition 2-3-2 : Alphabetical Ordering

An *alphabetical ordering* is a total order among the constituents of an alphabet, denoted <. Identical constituents x and y are denoted x = y.

Definition 2-3-3 : Lexicographical Tree

The *lexicograhical tree* for a free monoid $< X^*, \bullet, \Lambda >$ is an ordered tree defined by the tuple $< X^*, \mu, Q, O >$ where :

i) the set X^* comprises the nodes of the tree, the root node being λ;

ii) μ is the level function, such that given a word $\sigma \in X^*$, $\mu(\Lambda) = 1$ and $\mu(\sigma) =$**df** $| \sigma | + 1$;

iii) given σ and $\psi \in X^*$, $\sigma Q \psi$ iff $\psi =$**df** $\sigma \bullet x$, for $x \in X$;

iv) given any two successors $\psi =$**df** $y_1, y_2, ..., y_m$ and $\chi =$**df** $z_1, z_2, ..., z_m$ of $\sigma \in X^*$, which must have the same cardinality m, $\psi O \chi$ holds iff $y_1 < z_1$ or, for some maximal k, $y_i = z_i$, $1 \leq i \leq k \leq n$, and $y_{k+1} < z_{k+1}$, if $k < n$.

In fact, the ordering O is a total ordering that holds for all nodes in the same level of the tree. The lexicographical tree is a finitely generated, infinite tree. By König's *Unendlichkeitslemma* an infinite path through the tree can be identified. The set of all infinite paths are denoted X^ω. Since this dissertation is interested in defining a lexicographical ordering for $\Gamma \subseteq X^*$, the lexicographical tree need only be defined to the level that matches the greatest cardinality of any word in Γ.

Definition 2-3-4 : Lexicographical Ordering

A lexicographical ordering of the words in Γ specified by the sequence $(\phi_1, \phi_2, ... , \phi_p)$ is a total order derived by a *prefix, depth-first search* for the words in an appropriately defined lexicographical tree.

In Chapter 5, a lexicographical ordering is required; by virtue of this section, the lexicographical ordering need not be explictly defined in that chapter.

This page is intentionally left blank.

Chapter 3 SPECIFICATION AND ANALYSIS OF VALIDATION MODELS.

Section 3-1 : Introduction.

The objective of this chapter is to specify validation models from collections of component models. A validation model, Figure 3-1, consists of n component models.

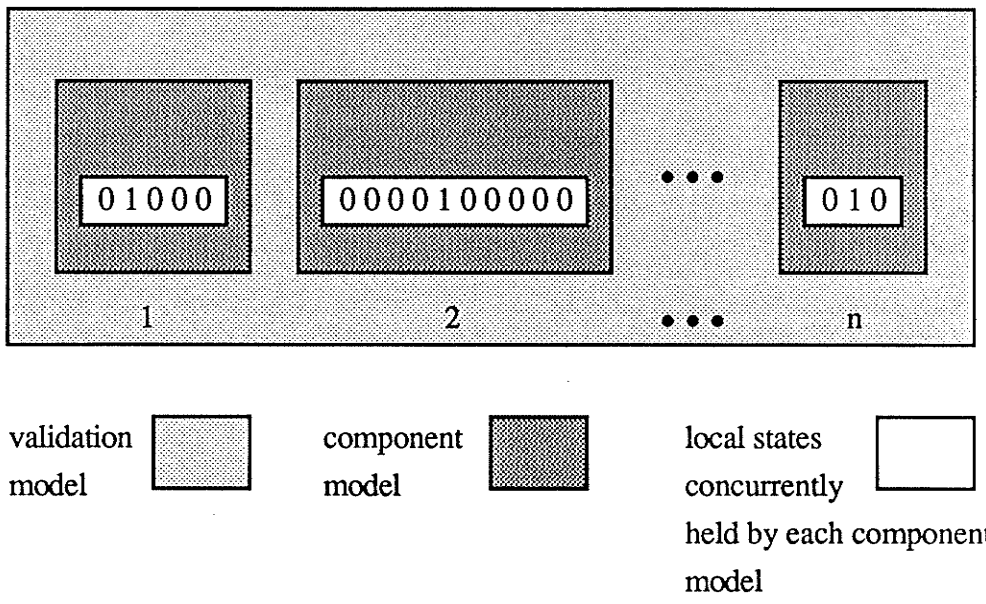


Figure 3-1 : Validation Model

Each component model is specified using a finite state machine state-based transition model, but is presented as a graph. Interaction among the component models is defined by coupling particular arcs of the graph. The resulting validation model forms a subclass

of *labelled general Petri nets*. Reachability analysis is used to generate a reachability graph from the validation model.

Section 3-2 : Component Models.

The component models are specified in terms of labelled graphs, with certain vertices distinguished from others. The vertices of the graph are labelled with a set of *atomic propositions* P.

Definition 3-2-1: Component Model

A *component model* is a connected, directed graph defined by a 5-tuple $< \Pi , \Delta , A , \Omega , H >$ where :

i) Π is a finite set of vertices $\{ \pi_1, \ldots , \pi_k \}$ called *states* ;

ii) Δ is a finite set of arcs, defined by a binary relation $\Delta : \Pi \rightarrow \Pi$, called *transitions*;

iii) A is the singleton set of *start states*, $A \in \Pi$;

iv) Ω is the set of *final states*, $\Omega \subseteq \Pi$;

v) H is an injective mapping which labels each $\pi \in \Pi$ with an element of 2^P, $H : \Pi \rightarrow 2^P$.

The states held by a component model are represented by the graph vertices Π. The graph arcs Δ, the transitions, describe how a component model may change its state. Given an arc $(\pi, \pi') \in \Delta$, when the component model holds the predecessor endpoint state π it may directly proceed to hold the successor endpoint state π'. Thus, the predecessor endpoint represents the enabling state of a transition while the successor endpoint represents the resulting state of a transition. It is clear that a chain can be constructed from the relation Δ. A component model begins execution in a start state $\{ \pi_0 \} = A \in \Pi$. Execution of the component model is represented by a walk through the graph, the holding of a succession of states. The set $\Omega \subseteq \Pi$ of final state(s) represent logical points at which the component model completes execution to provide a function; these states do not necessarily prevent further execution. The mapping $H : \Pi \rightarrow 2^P$ associates each

state $\pi \in \Pi$ with a distinct *state description* $\boldsymbol{C} \subseteq \boldsymbol{P}$, consisting of the atomic propositions that are true when the state π is held. A state may be labelled with any element of $2^{\boldsymbol{P}}$; but typically, a state is labelled with a proper subset of $\boldsymbol{P}$. Thus, $\boldsymbol{C}$ is an element of the range of the function H.

There are three restrictions placed on the definition of a component model. First, the graph must be connected. Second, there must be a walk from the start state to every other state in Π. These two restrictions avoid including states in the definition that can not possibly be held. A further restriction is that the relation Δ is irreflexive; that is, $\Delta = \{ (\pi, \pi') \ni \pi \neq \pi' \}$. Irreflexitivity means that a transition causes a change from one state to a different state; this restriction simplifies the analysis techniques presented in Chapter 4.

The mapping H from a vertex to a state description $\boldsymbol{C}$ associates additional information with the vertex to more accurately describe the corresponding state. An atomic proposition represents a phenomenon corresponding to the component model state; an atomic proposition is true when the corresponding phenomenon is observed, and is false at all other times. It is possible for a number of atomic propositions to be simultaneously true; thus, the states Π actually identify the set of atomic propositions that are true at any one time. In the next chapter, the atomic propositions serve as the foundation for the development of linear temporal logic.

A notation is required for the state currently held by the component model. A component model may hold at most one *local state* $\pi_i \in \{ \pi_1, \ldots, \pi_k \}$, $1 \leq i \leq k$, at any one time.

Definition 3-2-2 : Local State Function

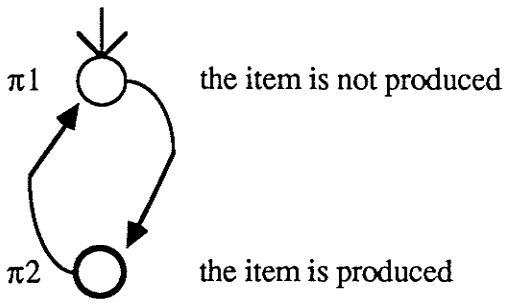
The *local state function* $\mu : \Pi \rightarrow \{ 0, 1 \}$ associates the value 1 with π_i if the component model holds that local state, and 0 with the remaining states.

The local state is an element of the range of μ and is denoted by a k-tuple $m = (\mu(\pi_1), \mu(\pi_2), \mu(\pi_3), \ldots , \mu(\pi_i), \ldots , \mu(\pi_k))$; thus, a local state $m \in \{ 0, 1 \}^k$. This notation is used in the formal definitions of execution defined below. But for the purposes of discussion, since only one state π_i can be held, a local marking is specified simply by (π_i). In the sequel, the local states will be combined into global states.

Accordingly, illustrations of component models are possible, based on Figure 2-1. The vertices of the graph corresponding to the finite set of states Π are represented by circles. A component model holding a particular state is indicated by placing a *token*, a black dot, inside the corresponding circle. The number of tokens in a state π_i, for a given a local state m, is specified by $m(\pi_i)$; hence, the local state denotes the number of tokens, at most one token in one vertex of the graph. The graph arcs describing the relation Δ among the states are represented by arrows. When a token exists in an enabling state of a transition, the edge(s) leading from that vertex indicates (indicate) the resulting state(s) the may be held by the component model. The start state A of a component model is indicated by a large arrow pointing to the vertex. The set Ω of final states of a component model are indicated by bold circles. The propositions that label the vertices are positioned close to the vertices in an unambiguous manner.

For illustrative purposes, the producer, n-slot buffer, and consumer system component designs are described and modelled.

A *producer* is a system component whose goal is to produce some arbitrary item. A producer is a commonly found in multiprocessor systems, e.g., a sender creating a message to be transmitted. A producer begins in a state where "the item is not produced", proceeds to a state where "the item is produced". Figure 3-2 illustrates a component model of a producer.



$\Pi_1 =$**df** $\{ \pi_1, \pi_2 \}$

$\Delta_1 =$**df** $\{ (\pi_1, \pi_2), (\pi_2, \pi_1) \}$

$A_1 =$**df** $\{ \pi_1 \}$

$\Omega_1 =$**df** $\{ \pi_2 \}$

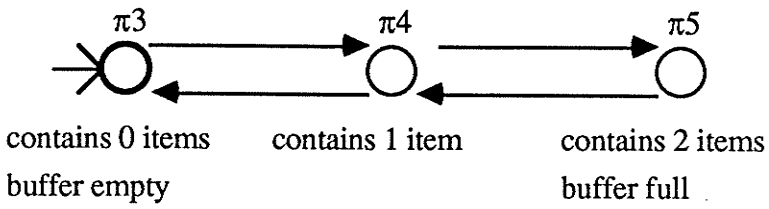
$H_1 =$**df** $\{ (\pi_1, \{$ the item is not produced $\}), (\pi_1, \{$ the item is produced $\}) \}$

Figure 3-2 : Producer Component Model

The final state does not cause the producer to terminate; thus, a producer holds a succession of local states that alternate between π_1 and π_2.

When a producer creates an item, it can not continue execution until the item is stored somewhere. A system component called a *n-slot buffer* is defined here for this purpose. This system component is responsible for managing n item storage slots. Clearly, there are n + 1 states that a n-slot buffer system component may hold. The buffer begins holding a state of storing no items, expressed by the propositions "buffer contains 0 items" and "buffer empty". Items can be added to the buffer to a maximum of n items; as each item is stored, the buffer holds a new state where the proposition "buffer contains k items", $1 \leq k \leq n$, is true. When the buffer holds the state where the proposition "buffer

contains n items" is true, the proposition "buffer full" is also true; this means that no more items may be added to the buffer. Figure 3-3 illustrates a 2-slot buffer component model.



$\Pi_2 =$ **df** $\{ \pi_3, \pi_4, \pi_5 \}$

$\Delta_2 =$ **df** $\{ (\pi_3, \pi_4), (\pi_4, \pi_5), (\pi_5, \pi_4), (\pi_4, \pi_3) \}$

$A_2 =$ **df** $\{ \pi_3 \}$

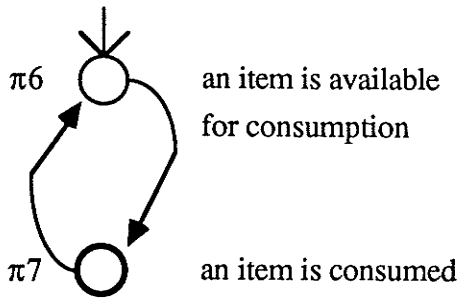
$\Omega_2 =$ **df** $\{ \pi_3 \}$

$H_2 =$ **df** $\{ (\pi_3, \{$ contains 0 items, buffer empty $\}),$
$(\pi_4, \{$ contains 1 item $\}),$
$(\pi_5, \{$ contains 2 items, buffer full $\}) \}$

Figure 3-3 : 2-Slot Buffer Component Model

The set of the possible succession of states is considerably more complex than a producer.

Finally, a *consumer* is a system component that takes items off the buffer and uses them for some purpose. A consumer begins holding a state where "an item is available for consumption", then proceeds to complete the function of consuming an item and ends up holding a state where "an item is consumed". Figure 3-4 illustrates this component model.



$\Pi_3 =$ **df** $\{ \pi_6, \pi_7 \}$

$\Delta_3 =$ **df** $\{ (\pi_6, \pi_7), (\pi_7, \pi_6) \}$

$A_3 =$ **df** $\{ \pi_6 \}$

$\Omega_3 =$ **df** $\{ \pi_7 \}$

$H_3 =$ **df** $\{ (\pi_6, \{$ an item is available for consumption $\}),$

$(\pi_7, \{$ an item is consumed $\}) \}$

Figure 3-4 : Consumer Component Model

Again, this component model alternates between holding two states.

Section 3-3: Validation Models.

Validation models consist of n concurrently executing, interacting component models. Interaction implies that, when at least two component models execute concurrently, in certain situations their executions can not proceed without aid from each other. For example, the producer can not store an item if the buffer has no available slot. Thus, interaction suggests that certain transitions in Δ of interacting component models are related. In fact, interaction is represented by having the respective component models each simultaneously hold some particular enabling state and then each proceed in step via the related transitions to some new resulting state; thus, a state-based transition model is required that combines, in this manner, a number of concurrently executing component models into a validation model. After the formal definitions of interaction are presented, this dissertation constructs a validation model called a Producer / Consumer system by combining the component models of the previous section.

Specifically, an interaction among n component models is defined by a collection of transitions, each selected from a distinct component model.

Definition 3-3-1: Validation Model Transition

A *validation model transition* is a q-tuple $t =\mathbf{df} (\delta_1, \delta_2, \ldots , \delta_q)$, where for every j, $1 \leq j \leq q$ such that $q \leq n$, $\delta_j \in \Delta_i$, where Δ_i, $1 \leq i \leq n$, is the set of transitions among the states of one component model involved in the interaction, and where the validation model transition can include only one member of the relation Δ_i.

A validation model transition indicates the simultaneous change in the state of all system component designs involved in the interaction. A validation model transition is an *atomic action*; either all involved component models change state or none of those involved do. This definition includes a validation model transition (δ_1), to provide for a change in state

of a single component model without interaction with other component models. In Figure 3-5, portions of two component models p and q are shown.

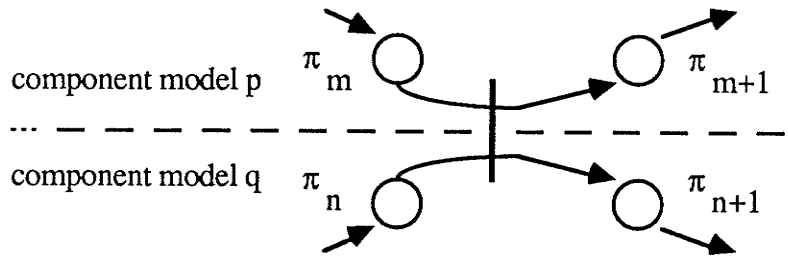


Figure 3-5 : Interaction Among Two Component Models

The bold line identifies the arcs involved in the validation model transition; viz. (π_m, π_{m+1}) and (π_n, π_{n+1}). When component models p and q simultaneously hold, respectively, the states π_m and π_n they are said to hold the enabling states of the validation model transition. This particular validation model transition requires that, simultaneously, the token in p moves from state π_m to π_{m+1} while the token in q moves from state π_n to state π_{n+1}. When p and q hold respectively the states π_{m+1} and π_{n+1} at the same time, they are said to hold the resulting states of the validation model transition. Because the transitions of the component models define an irreflexive relation, the resulting states are different from the enabling states. The simultaneous change in the component models' state is called a validation model transition *firing*; a formal definition of validation model transition firing is forthcoming. The combination of component models into a validation model is obtained through the definition of validation model transitions.

The interaction among component models is defined using a mathematical structure called a Petri net. The following is a general definition of Petri nets, which includes all conceivable Petri nets. Following this definition, a restricted Petri net class resulting from

indicating interaction among component models is defined and is used as a modelling technique.

Definition 3-3-2 : General Petri Net

A *General Petri Net* , GPN, [Hac] is a 5-tuple GPN =**df** $< \Pi , \Sigma , I , O , m_0 >$ satisfying the following statements :

i) Π is a finite set of *places* $\{ \pi_1, ... , \pi_r \}$; the *current place state* $\mu(\pi_i)$ is a function $\mu : \pi_i \rightarrow \mathbb{N}$, which determines the number of *tokens* in a place; the r-tuple $m = (\mu(\pi_1), ... , \mu(\pi_r))$ of all current place states is a called a *global marking*, and represents the *current global state* of the GPN;

ii) Σ is a finite set of *GPN transitions* $\{ t_1, ... , t_s \}$ that represent the changes in the marking of the GPN; the details of how the markings change are described by the following functions;

iii) I is the *input function* $I : \Sigma \times \Pi \rightarrow \mathbb{N}$ representing the enabling conditions of the GPN transitions; a GPN transition t is said to be *enabled* for a given marking $m_\circ$ if it satisfies the condition that $(\forall \pi) \in \Pi, m(\pi) \geq I(t, \pi)$;[1] this is always true when $I(t, \pi) = 0$, so only the entries where $I(t, \pi) > 0$ are inspected; a GPN transition is said to *fire* when it causes a state change; only enabled GPN transitions may fire;

iv) O is the *output function* $O : \Sigma \times \Pi \rightarrow \mathbb{N}$ which represents the conditions resulting from the firing of the GPN transition;

v) For a GPN transition $t \in \Sigma$, enabled at marking m, the result of firing the GPN transition is a marking $m' = (\mu(\pi_1), ... , \mu(\pi_r))$ where $(\forall \pi) \in \Pi, \mu(\pi) = m(\pi) - I(t, \pi) + O(t, \pi)$; the change of marking from m to m', by the firing of a GPN transition t, is denoted m[t >m', where [t > is a relation between the markings m

[1] That is, the GPN holds the enabling states of the GPN transition.

and m'; an underlying assumption is that no two GPN transitions may fire simultaneously.

vi) Finally, m_0 is a marking which represents the *initial state* of the GPN.

For a GPN transition t to be enabled, every place with an arc leading to t must have at least as many tokens as it has arcs leading to t. The firing of a transition t removes from a place one token for each arc leading from t to that place. The firing of t also puts in a place one token for each arc leading from t to that place. The conditions of transition firing are called *firing rules*. A GPN is *k-bounded* if no current place state exceeds the integer k; further, a GPN is *safe* if it is 1-bounded. *Unbounded* nets have place states of any integer. Only safe nets are defined here.

A validation model is specified with a restricted sub-class of General Petri nets.

Definition 3-3-3 : Validation Model

A *validation model* is a 7-tuple $N = \mathbf{df} < \Pi , \Sigma , I , O , m_0 , M_f , H >$ defined from n component models $< \Pi_i , \Delta_i , A_i , \Omega_i , H_i >$, $1 \leq i \leq n$, satisfying the following conditions :

i) the finite set $\Pi = \Pi_1 \cup \Pi_2 \cup ... \cup \Pi_n$, denoted $\{ \pi_1, ... , \pi_r \}$, contains all the local states that may be held concurrently by the validation model; of course, only one state per Π_i, $1 \leq i \leq n$, may be a constituent of a *global state* at a time;

ii) $\Sigma = \{ t_1, ... , t_s \}$ is a finite set of *validation model transitions*, where transition $t_u = (\delta_1, \delta_2, ... , \delta_q)$, $1 \leq u \leq s$, such that $1 \leq q \leq n$, where for every j, $1 \leq j \leq q$ such that $q \leq n$, $\delta_j \in \Delta_i$, where Δ_i, $1 \leq i \leq n$, is the set of transitions among the states of one component model i involved in the interaction, and where the validation model transition can include only one member of the relation Δ_i; each $\delta_j = (e_j, r_j)$, where e_j and r_j are, respectively, one of the enabling and resulting states of the transition; because $q \leq n$, some component models may not be involved in the transition definition; this definition also includes the case where

a transition is a singleton set of edges; the rules for firing the transitions are the same as for GPN, but the input and output functions must be defined;

iii) $I : \Sigma \times \Pi \rightarrow \{ 0, 1 \}$, where, for each $\delta_j = (e_j, r_j) \in t_u = (\delta_1, \delta_2, \ldots , \delta_q)$, when $\pi \in \{ e_1 \cup e_2 \cup \ldots \cup e_q \}$, $I(t_u, \pi) = 1$; otherwise, $I(t_u, \pi) = 0$;

iv) $O : \Sigma \times \Pi \rightarrow \{ 0, 1 \}$, where, for each $\delta_l \in t_u$, when $\pi \in \{ r_1 \cup r_2 \cup \ldots \cup r_q \}$, $O(t_i, \pi) = 1$; otherwise, $O(t_u, \pi) = 0$;

v) $m_0 = A_1 \cup A_2 \cup \ldots \cup A_n$ is the validation model *Start Set*;

vi) $M_f = \Omega_1 \cup \Omega_2 \cup \ldots \cup \Omega_n$ is the validation model *End Set*;

vii) $H = H_1 \cup H_2 \cup \ldots \cup H_n$ are the mappings from states to the set of propositions $\boldsymbol{P} = \boldsymbol{P}_1 \cup \boldsymbol{P}_2 \cup \ldots \cup \boldsymbol{P}_n$.

Validation models hold global states specified by *global markings*. A validation model has clearly defined initial and final global markings. The word "concurrent" implies that, at some time, component model i, viz. $< \Pi_i , \Delta_i , A_i , \Omega_i, H_i >$ where $\Pi_i = \{ \pi_1, \ldots , \pi_k \}$, comprising the validation model, displays a local state $m^i = (\mu_i(\pi_1), \mu_i(\pi_2), \mu_i(\pi_3), \ldots , \mu_i(\pi_z), \ldots , \mu_i(\pi_k))$, $1 \leq z \leq k$. A global marking of a validation model is an n-tuple $m = (m^1 ; m^2 ; \ldots ; m^n)$, where the component model local states are separated by semi-colons. Given a validation model $N =$**df** $< \Pi , \Sigma , I , O , m_0 , M_f , H >$ and a global marking m, the number of tokens in a state $\pi \in \Pi$, is denoted $m(\pi)$. A marking $m \in \{ 0, 1 \}^r$.

There are some important characteristics of the validation models inherited from the component models. First, because only one state from a component model can be held at one time, a validation model is a safe Petri net. Second, because the relation Δ in the component models is irreflexive, the firing of a transition produces a resulting global state different from the enabling global state. By the definition, a validation model begins in one initial global state; hence, the definition uses the lower-case denotation m_0 to indicate the fact that there is only one initial marking. Unlike the start set, the end set allows more

than one final marking; this accounts for the upper-case denotation M_f used in the definition.

To illustrate the use of the validation models, a Producer / Consumer validation model is specified by combining the producer, consumer, and buffer component models defined above. The producer and buffer interact when a producer creates some item and places it in a buffer slot. A consumer and a buffer interact when the consumer takes an item from the buffer and consumes it. The interaction among a producer and buffer may proceed only if there is a slot in the buffer to store an item; likewise, the consumer may proceed only if there is some item stored in the buffer. Figure 3-6 illustrates a P / C validation model with a two-slot buffer, using the techniques of Figures 2-1 and 3-5. For clarity, the propositions are not illustrated.

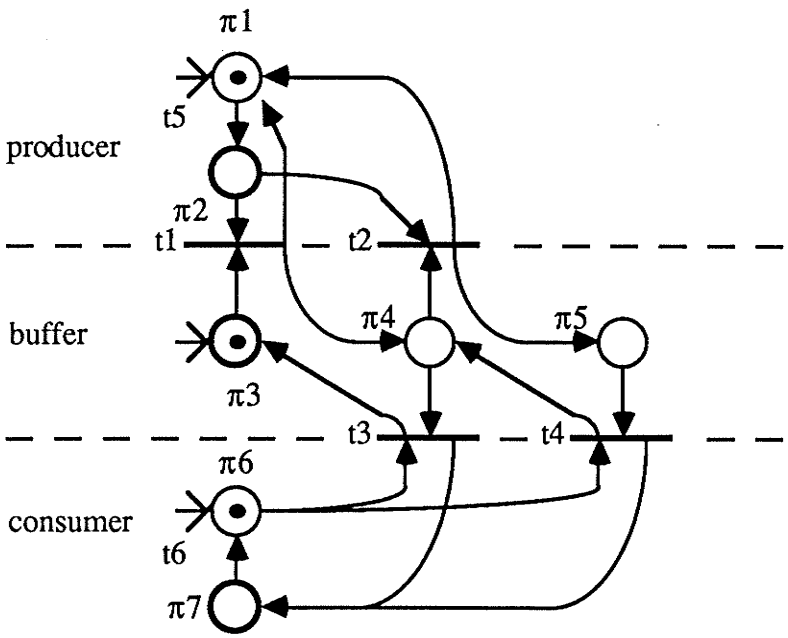


Figure 3-6 : The Producer / Consumer Validation Model

The circles represent the places $\Pi = \{ \pi_1, \ldots, \pi_7 \}$. The dots in the start set places are the tokens. The validation model start set of Figure 3-6 is $m_0 = (\pi_1, \pi_3, \pi_6) = (1, 0; 1, 0, 0; 1, 0)$. The validation model end set of Figure 3-6 is $M_f = (\pi_2, \pi_3, \pi_7) = (0, 1; 1, 0, 0; 0, 1)$.

The bars represent the validation model transitions $\Sigma =$ **df** $\{ t_1, \ldots, t_6 \}$. The validation model transitions t_5 and t_6, between π_1 and π_2 and between π_6 and π_7 respectively, are not illustrated with bars, because they consist of one arc; these validation model transitions imply that the respective component models can change state without interaction with other component models. Figures 3-7 (a) and 3-7 (b) represent the input function I and output function O respectively.

I	π_1	π_2	π_3	π_4	π_5	π_6	π_7
t_1	0	1	1	0	0	0	0
t_2	0	1	0	1	0	0	0
t_3	0	0	0	1	0	1	0
t_4	0	0	0	0	1	1	0
t_5	1	0	0	0	0	0	0
t_6	0	0	0	0	0	0	1

(a)

O	π_1	π_2	π_3	π_4	π_5	π_6	π_7
t_1	1	0	0	1	0	0	0
t_2	1	0	0	0	1	0	0
t_3	0	0	1	0	0	0	1
t_4	0	0	0	1	0	0	1
t_5	0	1	0	0	0	0	0
t_6	0	0	0	0	0	1	0

(b)

Figure 3-7 : Input and Output Functions

In Figure 3-6, only validation model transition t_5 is enabled.

In Figure 3-6 the places are associated with the following propositions :

Places :

Producer

π_1 : the item is not produced

π_2 : the item is produced

Buffer

π_3 : buffer contains 0 items; buffer empty

π_4 : buffer contains 1 item

π_5 : buffer contains 2 items; buffer full

Consumer

π_6 : an item is available for consumption

π_7 : an item is consumed

The proposition(s) associated with a place is (are) true, when there is a token in that place.

The validation model transitions describe the actions of the validation model. In Figure 3-6 the following interpretations are used with the validation model transitions :

t_1 and t_2 : produce item and place on buffer

t_3 and t_4 : take item off buffer and consume item

t_5 : produce item

t_6 : get ready to consume next item

The interpretations aid in understanding the validation model.

A validation model can be partitioned into validation model *phases* [CG], which represent portions of the overall function represented by the validation model. Each phase is represented by a separate validation model and is composed of component models. The validation model phases are combined into a validation model by taking the union of all tuples specifying a phase, where the end set elements of one phase is matched with the singleton start set of another phase, while conserving the number of tokens in the resulting validation model. Phases provide a means of composing the design problem. The

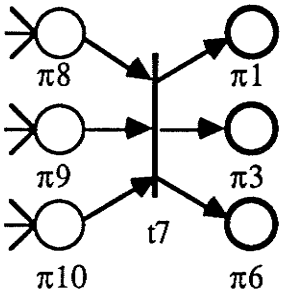
intention is to specify and analyse each validation model phase, assemble the phases into the entire validation model, and analyse the entire system design. The producer / consumer example is defined below with phases.

Phases provide flexibity in specifications. The validation model allows for *terminating phases*, with well defined end states having no possibility of further execution. Terminating phases can be used as building blocks to form numerous validation model structures. Structures with loops define *cyclic* validation models, defined not to terminate except in certain situations.[2] This dissertation is restricted to validation models which are cyclic.

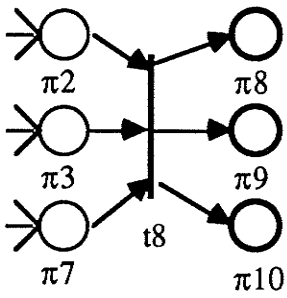
Fortunately, a terminating validation model N may be redefined as a cyclic validation model N' by the addition of two phases. The *turn on phase* has a *turned off place* for each component model, that represents all of the component models holding a state where they are turned off and not executing. A single validation model transition t is included in this phase, with an input function mapping each turned off place to t and an output function mapping each place of the start set with t. The firing of the transition changes the component models' global state from holding the turned off states to holding the start set of N. The *turn off phase* consists of the same turned off places, but has a number of transitions t'. For each transition t', the input function maps one element selected from the phase end set of each component model and the output function maps t' to each of the turned off places. The component models must all simultaneously hold the local state corresponding to the turned off place; the corresponding global state is called the *turned off marking*. The intent is that any validation model that includes the turn on and turn off phases is cyclic; the proof of this statement follows in a forthcoming section.

[2] Such as turning off the system for maintenance.

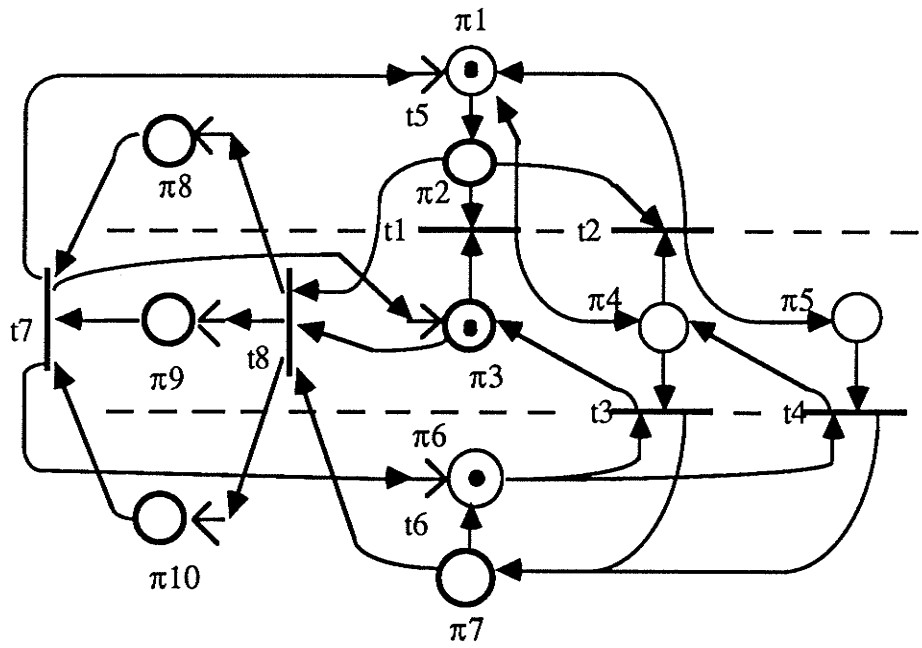
To illustrate the concept of assembling phases into structures, the P / C validation model is extended. The P / C validation model, defined so far, can be thought of as a single cyclic phase, called the P / C phase. It would useful to be able to turn on and turn off the P / C validation model. Figure 3-8 illustrates the P / C validation model with these additions.



(a) Turn on Phase



(b) Turn off Phase



(c) Combination of Three Phases

Figure 3-8 : A 3 Phase P / C Validation Model

Turning on the validation model places the P / C validation model into its initial global state. The P / C validation model can only be turned off when it holds a state in the end set of P / C phase; this was done to simplify the turn off validation model phase. Figure 3-8 (c) illustrates the P / C validation model holding the state represented by the start set of the P / C validation model phase. The 3 Phase P / C validation model conserves the number of tokens.

Section 3-4 : Reachability Analysis of Validation Models.

The validation models are useful because they simulate system execution. The principles of validation model execution are illustrated with the example of Figure 3-6. The initial marking $m_0 = (\pi_1 ; \pi_3 ; \pi_6) = (1, 0; 1, 0, 0; 1, 0)$ represents the initial conditions of the phase, namely that the producer is ready to produce, $m_0(\pi_1) = 1$, the buffer is empty, $m_0(\pi_3) = 1$, and the consumer is ready to accept an item, $m_0(\pi_6) = 1$. Transition t_5 is enabled at m_0, because its input conditions are satisfied; that is, $(\forall \pi) \in \Pi$, $m_0(\pi) \geq I(\pi, t_5)$, viz. $(1, 0; 1, 0, 0; 1, 0) \geq (1, 0; 0, 0, 0; 0, 0)$. The firing of transition t_5, denoted $m_0[t_5 > m_1$, represents the production of an item and results in a new marking $m_1 = (0, 1; 1, 0, 0; 1, 0)$. The result of firing the transition t_5 is illustrated in Figure 3-9.

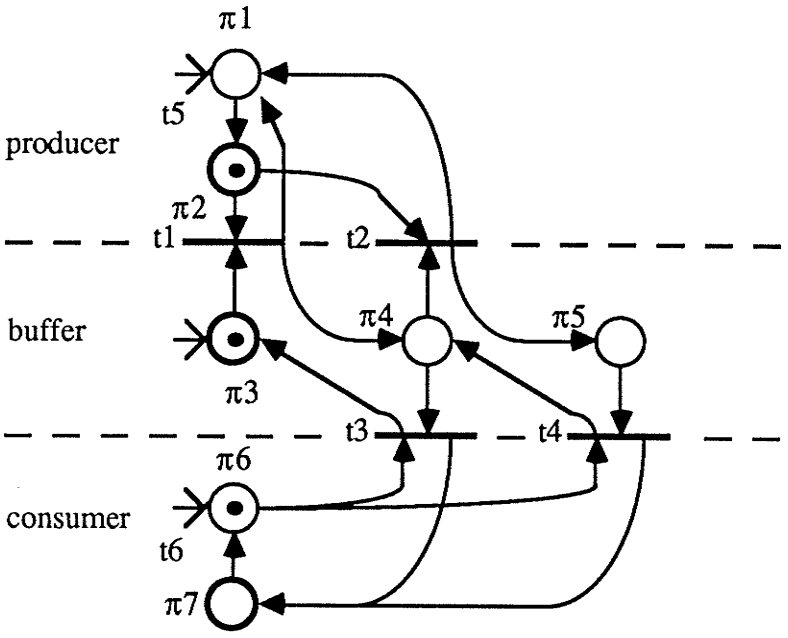


Figure 3-9 : P / C Validation Model Marking $m_1 = (\pi_2 ; \pi_3 ; \pi_6)$

An item has been produced, $m(\pi_2) = 1$, but the buffer is empty, $m(\pi_3) = 1$; thus, transition t_1 is enabled. The firing of t_1, $m_1[t_1 > m_2$, places an item in the buffer. The result of firing t_1 is shown in Figure 3-10.

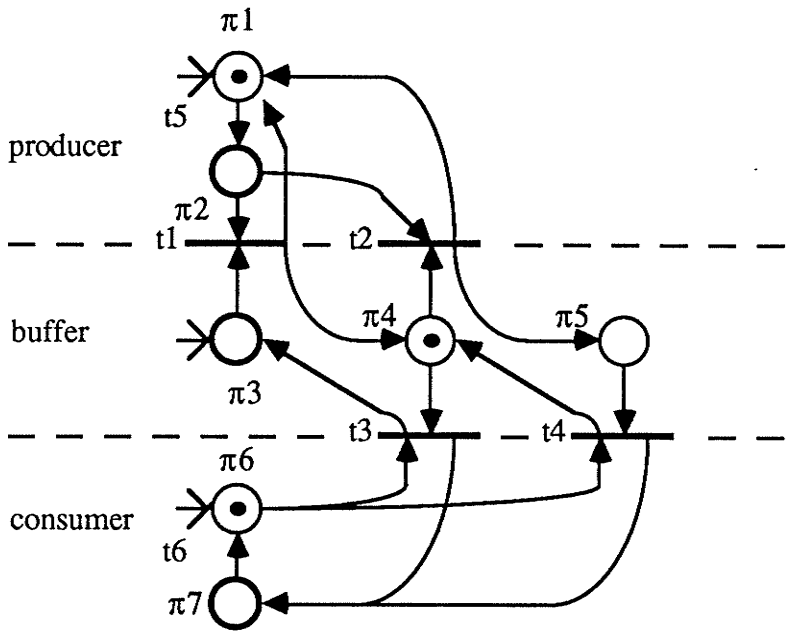


Figure 3-10 : P / C Validation Model Marking $m_2 = (\pi_1 ; \pi_4 ; \pi_6)$

Now there are two transitions enabled, t_5 and t_3. Either transition may fire in lieu of the other, but not both at once. If t_5 fires the marking (0, 1; 0, 1, 0; 1, 0) results, and if t_3 fires the marking (1, 0; 1, 0, 0; 0, 1) results. Figures 3-11 and 3-12 illustrate the firing of transition t_3 and then transition t_5, yielding markings m_3 and m_4.

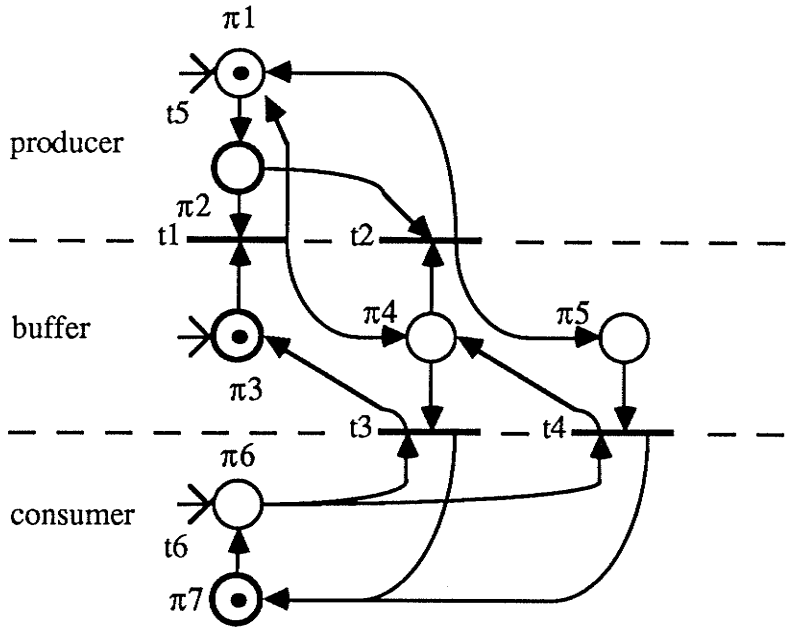


Figure 3-11 : P / C Validation Model Marking $m_3 = (\pi_1 ; \pi_3 ; \pi_7)$

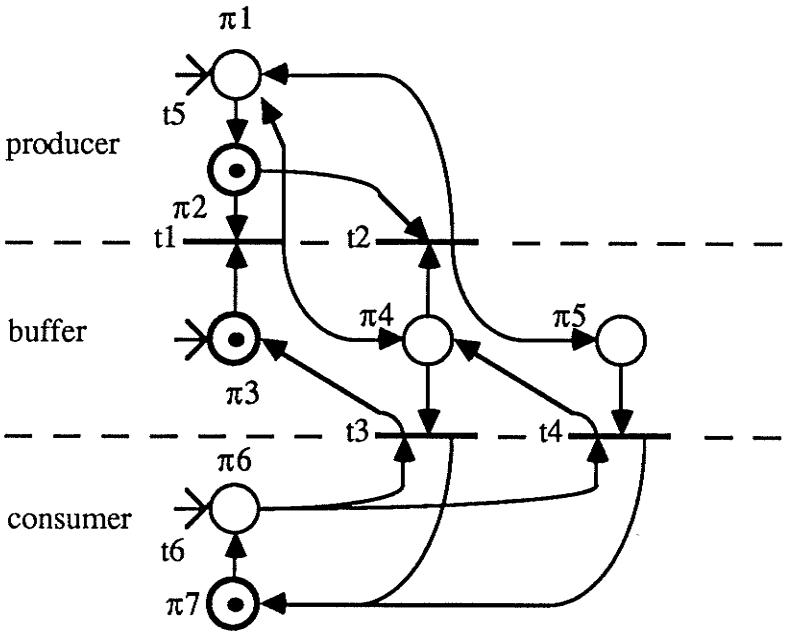


Figure 3-12 : P / C Validation Model Marking $m_4 = (\pi_2 ; \pi_3 ; \pi_7)$

The Figures 3-6, 3-9 through 3-12 represent a succession of transition firings denoted by $m_0[t_5 > m_1, m_1[t_1 > m_2, m_2[t_3 > m_3, m_3[t_5 > m_4$. The sequence of transitions t_5, t_1, t_3, t_5 can be denoted by σ, which provides an abbreviated notation for the succession of transition firings; i.e., $m_0[\sigma > m_4$. Finally, the firing of the sequence σ of transitions results the sequence of markings m_0, m_1, m_2, m_3, m_4.

Three sets formally describe the finite execution of a validation model $N = < \Pi , \Sigma , I , O , m_0 , M_f , H >$.

Definition 3-4-1 : Finite and Terminal Transition Firing Sequences

The following sets define transition firings that cause finite execution of a validation model [Hac][Peter81] :

i) the set $S_N(m_0) =$**df** $\{ \sigma \in \Sigma^* \ni \exists m' \in \{ 0, 1 \}^r \wedge m_0[\sigma > m' \}$ contains *finite transition firing sequences* beginning with the initial marking m_0;

ii) a subset of $S_N(m_0)$ is the set of *terminal firing sequences from* m_0, $T_N(m_0, M_f) =$**df** $\{ \sigma \in \Sigma^* \ni \exists m' \in \{ 0, 1 \}^r \wedge m' \in M_f \wedge m_0[\sigma > m' \}$, that places the validation model into a final global state.

The sets $S_N(m_0)$ and $T_N(m_0, M_f)$ can be infinite sets of finite transition firing sequences. Later, a definition of infinite transition firing sequences is defined. If there is a transition sequence σ such that $m_0[\sigma > m$, then marking m is said to be *reachable* from the initial marking m_0. In fact, for a validation model N, it is possible to construct the set of all markings that are reachable from the initial marking m_0.

Definition 3-4-2 : Reachability Set

The *reachability set* for a validation model N from an initial marking m_0 is the set $RM_N(m_0) =$**df** $\{ m \in \{ 0, 1 \}^r \ni \exists \sigma \in \Sigma^* \wedge m_0[\sigma > m \}$.

The initial marking $m_0 \in RM_N(m_0)$, because $m_0[\sigma > m_0$ for the empty prefix σ. What is required is a means to generate theses sets.

Analysis techniques [Peter77][Peter81] are available to generate the sets $S_N(m_0)$, $T_N(m_0, M_f)$, and $RM_N(m_0)$. For a validation model N, analysis techniques are based on the following structure.

Definition 3-4-3 : Reachability Tree

A *reachability tree* is an unordered tree $RT(m_0)$ **=df** < RTN, μ, Q > where :

i) the nodes RTN are generated inductively, where

   a) $m_0 \in$ RTN; i.e., the root node is m_0;

   b) If $m \in$ RTN and $m[t >m'$ for some $t \in \Sigma$ then $m' \in$ RTN;

ii) the level function μ : RTN $\rightarrow \mathbb{N}$ is defined such that,

   a) $\mu(m_0)$ **=df** 1;

   b) for $m \in$ RTN, where $m_0[\sigma >m$, and where $\sigma \in \Sigma^*$, $\mu(m)$ **=df** $| \sigma | + 1$;

iii) the binary relation m Q m' holds if $m[t >m'$, for $t \in \Sigma$.

A reachability tree for a validation model contains a finite set of distinct markings; in fact, the markings contained in $RT(m_0)$ contain the reachability set $RM_N(m_0)$. A reachability tree is finitely generated, but can be infinite. By König's *Unendlichkeitslemma*, such an infinite reachability tree contains at least one infinite path of markings.

For example, the markings generated in Figures 3-6, 3-9 through 3-12 for the P / C validation model are arranged into a portion of a reachability tree illustrated in Figure 3-13. The firing of transitions from the initial marking are indicated by the bold arrows.

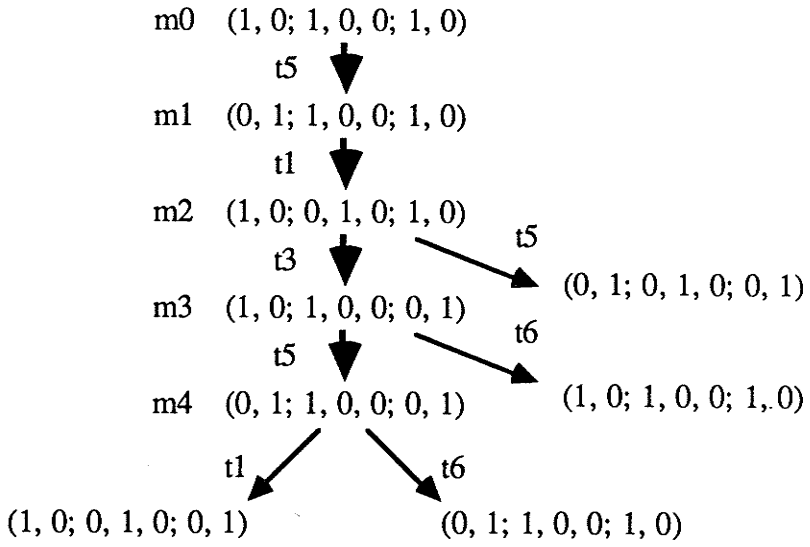


Figure 3-13 : Portion of Reachability Tree for P / C Validation Model

This tree is infinite. A *Reachability Algorithm*, found in the cited references, is used to construct a reachability tree.

The markings in a reachability tree of a validation model can be classified. The initial marking is the *start state* of the reachability tree and is the root node of the tree. *Internal markings* are intermediate markings that are found along a path in a reachability tree. If an internal marking appears in a reachability tree more than once, it is called a *duplicate marking*. Markings that correspond to the end set of a validation model are called *final states*. Markings that have no enabled transitions are called *terminal markings*, an infinite path can not have a terminal marking. Markings that represent undesirable validation model behaviour are called *error markings*; and should not be reachable by a validation model. A terminal marking that does not correspond to a final marking is an error condition called a *deadlock marking*; a deadlock marking occurs if, and only if, the validation model is incorrectly specified.

Section 3-5 : The Reachability Graph

The reachability tree, though useful, is reduced to a *reachability graph*, a compact and finite means of depicting an infinite reachability tree. The intention is to let the reachability set define the graph vertices, let the firing of a single transition define the arcs among the vertices, and label the graph using the validation model mapping H. Accordingly, in the next chapter, graph theory is used for temporal analysis purposes.

Definition 3-5-1 : The Reachability Graph

A reachability graph is a tuple RG **=df** $< S , T , m_0 , M_f , H >$, derived from a validation model N **=df** $< \Pi , \Sigma , I , O , m_0 , M_f , H >$, where :

i) S is a finite set $\{ s_1, s_2, \ldots , s_n \}$ of vertices representing validation model global states corresponding to $RM_N(m_0)$;

ii) T is a finite set of directed arcs, each corresponding to a distinct transition $t \in \Sigma$ of N; for a transition t **=df** (s_i, s_j), s_i is its enabling global state and s_j is its resulting state; i.e., a validation model holding a global state represented by s_i may fire transition t to hold the global state represented by s_j; the firing of the transition t is denotated $s_i[t > s_j$.

iii) m_0 is the start set of N;

iv) M_f is the end set of N;

v) H is the mappings of N.

The significance of the reachability graph is that it simplifies the validation model without removing the ability to represent its execution. Definition 3-5-2 modifies the reachability tree algorithm [Peter81] so that it produces a graph instead of a tree.

Definition 3-5-2 : Reachability Graph Algorithm

A *reachability graph* RG =**df** < S , T , m_0 , M_f , H > is constructed from a validation model N =**df** < Π , Σ , I , O , m_0 , M_f , H > by the following algorithm :

{ - Create the first vertex of the reachability graph with the information contained in the validation model.

- m_0, M_f, and H are the same in the reachability graph and validation model. }

$S := \{ m_0 \}$
$T := \varnothing$
$\phi_{fro} := \{ m_0 \}$
state_index := 1
$\phi_{int} \wedge \phi_{ter} := \varnothing$

{ main line }

WHILE $\phi_{fro} \neq \varnothing$ DO

a) SELECT m_{test} FROM ϕ_{fro}

b) IF $\neg\exists t \in \Sigma \ni \forall \pi \in \Pi : m_{test}(\pi) \geq I(\pi, t)$
THEN DO $\phi_{ter} := \phi_{ter} + \{ m_{test} \}$
ELSE DO

c) $\forall t \in \Sigma \ni \forall \pi \in \Pi : m_{test}(\pi) \geq I(\pi, t)$ DO { for every enabled transition }

d) $\forall \pi \in \Pi$ DO $m'(\pi) = m_{test}(\pi) - I(\pi, t) + O(\pi, t)$ OD { fire transition }

e) IF $\exists m \in S \ni m = m'$
THEN DO $T := T + \{ (m_{test}, m) \}$

f) ELSE DO
$m_{state\_index} := m'$
$S := S + \{ m_{state\_index} \}$
$T := T + \{ (m_{test}, m_{state\_index}) \}$
$\phi_{fro} := \phi_{fro} + \{ m_{state\_index} \}$
state_index := state_index + 1
END ELSE

g) $\phi_{int} := \phi_{int} + \{ m_{test} \}$
OD
END ELSE

h) $\phi_{fro} := \phi_{fro} - \{ m_{test} \}$
END WHILE

Lemma 3-5-3 : S is Equivalent to $RM_N(m_0)$.

For a reachability graph $< S , T , m_0 , M_f , H >$ constructed from a validation model $N =\mathbf{df} < \Pi , \Sigma , I , O , m_0 , M_f , H >$, by the reachability graph algorithm, $S = RM_N(m_0)$.

Proof : There are two cases that must be shown, that $RM_N(m_0) \subseteq S$ and $S \subseteq RM_N(m_0)$.

Case i : $RM_N(m_0) \subseteq S$. That any element in $RM_N(m_0)$ must be an element of S is shown by induction on x, the number of times a reachable marking m is added to S, and by construction using the reachability graph algorithm definition.

x = 1 : The initial marking m_0 is in both $RM_N(m_0)$ and S.

x > 1 : It is assumed that S contains x - 1 markings and that $\exists m_{test} \ni (\exists \sigma \in \Sigma^* \ni m_0[\sigma >m_{test}) \wedge m_{test} \in RM_N(m_0) \wedge m_{test} \in S \cap \phi_{fro}$. When m_{test} is selected for processing and a transition $t \in \Sigma$ is enabled in m_{test}, then $m_{test}[t >m$. Thus, because $m_0[\sigma >m_{test}$ and $m_{test}[t >m$, there is a sequence of transition firings $\sigma' =\mathbf{df} \sigma \bullet t \in \Sigma^*$ such that $m_0[\sigma' >m$ and $m \in RM_N(m_0)$. If a marking m is distinct from any other marking in S, it is the x-th marking added to S; otherwise, it is already included in S. Thus, any element in $RM_N(m_0)$ must be an element in S.

Case ii : $S \subseteq RM_N(m_0)$. It is assumed that $\exists m \ni m \in S \wedge m \notin RM_N(m_0)$. Since $m \notin RM_N(m_0), \neg \exists \sigma' \in \Sigma^* \ni m_0[\sigma' >m$. Clearly, for m_0, $m_0 \in S \wedge m_0 \in RM_N(m_0)$, contradicting the assumption. Furthermore, for $m \in S$, $\exists m_{test} \ni m_{test}[t >m$, where $\exists \sigma \in \Sigma^* \ni m_0[\sigma >m_{test}$. Therefore, $\exists \sigma' =\mathbf{df} \sigma \bullet t \in \Sigma^* \ni m_0[\sigma' >m$; reductio ad absurdum, $S \subseteq RM_N(m_0)$. **Q.E.D.**

Theorem 3-5-4 : Definition 3-5-2 produces a Reachability Graph.

The reachability graph algorithm terminates when applied to a validation model $N =df < \Pi , \Sigma , I , O , m_0 , M_f , H >$ and produces a reachability graph $RG =df < S , T , m_0 , M_f , H >$ as per Definition 3-5-1.

Proof : By construction of RG using the reachability graph algorithm.

Case i) : To show that the algorithm terminates, it is sufficient to show that at some point ϕ_{fro} is empty. Every time a m_{test} is selected from ϕ_{fro} and processed, it is deleted from ϕ_{fro}. If it can be shown that the ELSE DO, statements f) are executed a finite number of times, then the algorithm will terminate. The algorithm starts with the initial marking m_0 and, by firing enabled transitions, proceeds to define the vertices S. The statement b), $(\forall \pi) \in \Pi : m_{test}(\pi) \geq I(\pi, t)$, tests if a transition is enabled. A maximum of all transitions may be enabled and a minimum of no transitions may be enabled. If no transition is enabled m_{test} is removed from ϕ_{fro}. If the new marking m' is not a member of S, it is added to ϕ_{fro}, S, and an arc (m_{test}, m') is added to T. When the result of firing a transition is a marking m' that duplicates m already in ϕ_{fro}, the algorithm creates an arc leading from m_{test} to m. For a given validation model N there is a maximum of $MAX = | \Pi_1 | \times | \Pi_2 | \times ... \times | \Pi_1 |$ markings.[3] Since MAX is a finite value and $x \leq MAX$, a distinctly new m is added to ϕ_{fro} and S a finite number of times. Therefore, the algorithm must terminate, S must be finite, and the reachability graph is of finite order. That S corresponds to $RM_N(m_0)$ follows from Lemma 3-5-3.

Case ii) : An arc $t =df (m_{test}, m)$ is added to T if $\exists t \in \Sigma$ that is enabled and $m_{test}[t > m$. The firing of a transition can result in a distictly new marking m or a duplicate marking m, but never a marking $m = m_{test}$. Thus, the arcs are directed and

[3] Here, $\times$ is integer multiplication.

do not form loops. The number of transitions enabled at m_{test} may vary; therefore, the reachability graph is non-regular. Clearly, an arc $t \in T$ represents the firing of t at the enabling state m_{test} producing the resulting state m.

Cases iii), iv) and v) : The sets m_0 and M_f , and the mapping H, follow immediately. **Q.E.D.**

Lemma 3-5-6 : Definition 3-5-2 Produces a Strongly Connected Graph.

The reachability graph algorithm applied to a validation model N that includes turn on and turn off phases generates a strongly connected reachability graph.

Proof : By Construction Based on Definition 3-5-2.

By convention, a deadlock marking implies an incorrectly defined validation model. Because the turn on and turn off phases are included in the validation model, none of the end sets are terminal markings. The end set markings are reachable from the start set markings. Because the turned off marking is reachable from all of the end set markings and the start set marking is reachable from the turned off marking, all markings in S are reachable; thus for any two u, $v \in S$, this is tantamount to a path from u to v and a path from v to u. **Q.E.D.**

The implication of the previous proof is that a validation model generating a strongly connect reachability graph is cyclic.

To illustrate the results of applying the reachability graph algorithm to a validation model, Figure 3-14 is the reachability graph for the 2-slot buffer producer consumer validation model of Figure 3-6. The vertices of the graph are expressed in terms of the global state notation. The arrows represent the arcs among the vertices. The state m_0 is indicated in the normal manner, but the final state is enclosed in a box.

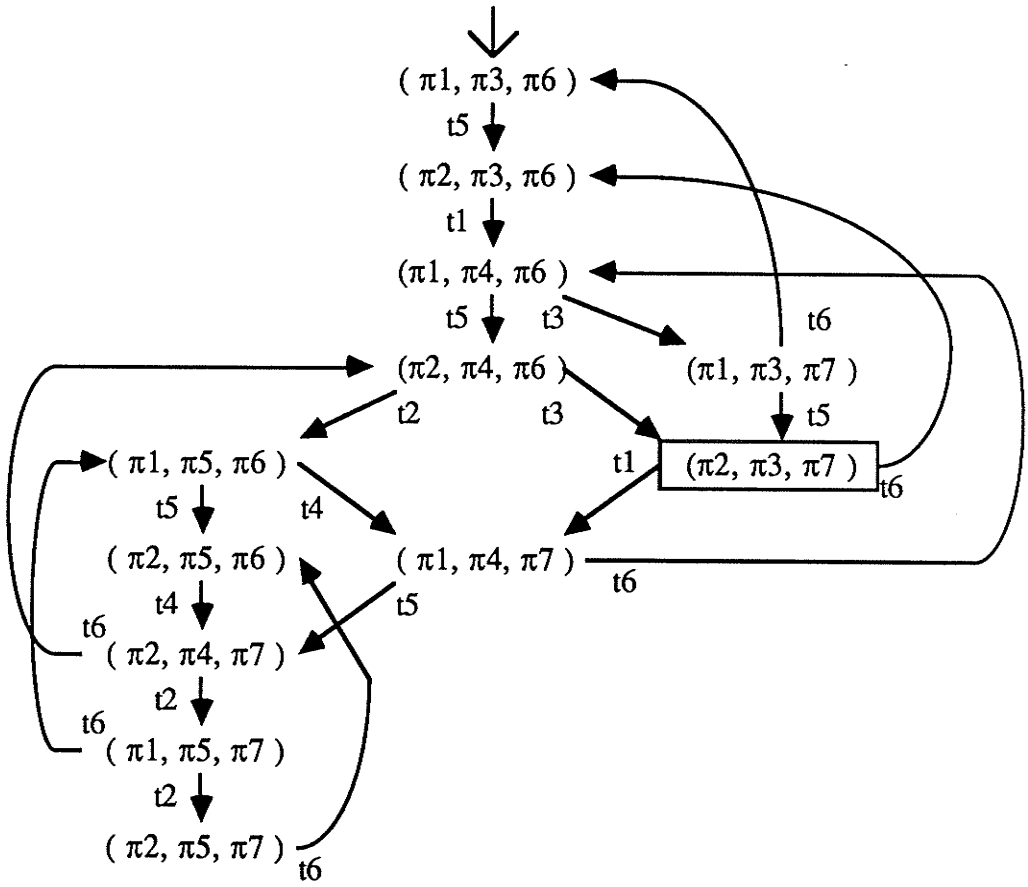


Figure 3-14 : Reachability Graph for the 2-Slot P / C Validation Model

This page is intentionally left blank.

Chapter 4 : TEMPORAL ANALYSIS OF THE VALIDATION MODEL

Section 4-1 : Introduction - The View of Time

The PLTL defined in the next chapter is based on the *linear, discrete view of time* [RU][HC]. This view of time is defined formally [Bri][Hal][LPS].

Definition 4-1-1 : The Time Line

The *time line* is defined by the tuple $< T_{\aleph}, \mathbf{R} >$, where :

i) $T_{\aleph}$ is a non-empty set of *time instants* with, possibly transfinite, cardinality $\aleph$; i.e., $\aleph =$**df** $| T_{\aleph} |$;

ii) the symbol **R** denotes the relation of *temporal precedence*, a strict well order on $T_{\aleph}$;

iii) the well ordered set $< T_{\aleph}, \mathbf{R} >$ is order-isomorphic to an unique $\alpha \in$ ORD; there is a bijective function $\rho : T_{\aleph} \rightarrow \alpha$, such that if, for τ and $\tau' \in T_{\aleph}$, $\tau \mathbf{R} \tau'$ then $\rho(\tau) \in \rho(\tau') \in \alpha$;

iv) the well ordered set $< T_{\aleph}, \mathbf{R} >$ =**df** $\{ \tau_{\beta} \ni \beta \in \alpha \}$, where τ_{β} is the unique inverse image of β under ρ; thus, the function ρ^{-1} conforms to the definition of a sequence.[1]

It is useful to choose a temporal reference point τ_{β}, $\beta \in \alpha$, called the *present time instant*. The present time instant τ_{β} identifies, for inspection, the history of time instants contained in $T_{\aleph}$ that temporally precede τ_{β}.

Definition 4-1-2 : The Temporal Predecessors of τ_{β}

The present time instant τ_{β} selects a prefix of the sequence ρ^{-1}, using the function $\underline{\tau}_{\beta} : \{ 0, 1, 2, \ldots , \beta - 1 \} \rightarrow T_{\aleph}$, where $\underline{\tau}_{\beta}$ is the restriction of ρ^{-1} to the domain

[1] i.e., the set $T_{\aleph} =$**df** $\{ \tau_0, \tau_1, \ldots , \tau_{\alpha - 1} \}$ is indexed by ordinal numbers.

$\beta \in \alpha \in$ ORD. The range of the function $\underline{\tau_\beta}$ is the set of *predecessors* of τ_β, $PRE(\underline{\tau_\beta})$ =**df** $\{ \tau_0, \tau_1, \tau_2, \ldots, \tau_{\beta-1} \}$. Thus, $\tau_\beta \in T_\aleph \Leftrightarrow PRE(\underline{\tau_\beta}) \subset T_\aleph$.

The set $PRE(\underline{\tau_\beta})$ is countably finite and can be indexed by a subset of $\mathbb{N}$, when $\beta < \omega$. The set $PRE(\underline{\tau_\beta})$ is countably infinite and can be indexed by the set $\mathbb{N}$, when $\beta = \omega$.

Definition 4-1-3 : The Next Time Instant

Given the present time instant τ_β, the *next* future time instant is $\tau_{\beta+1}$, where by definition no other time instants are identified between τ_β and $\tau_{\beta+1}$; this is achieved by the set $PRE(\underline{\tau_{\beta+1}})$ =**df** $PRE(\underline{\tau_\beta}) \cup \{ \tau_\beta \}$.[2]

By the principle of transfinite induction, $PRE(\underline{\tau_\alpha})$ =**df** $\{ \tau_\beta \ni \beta \in \alpha \}$ =**df** $T_\aleph$.

The linear, discrete view of time is summarized.

Definition 4-1-4 : The α-relation

The *α-relation* holds between any two time instants τ_β and $\tau_\gamma \in T_\aleph$, denoted by $\tau_\beta \; \alpha \; \tau_\gamma$, iff $| PRE(\underline{\tau_\beta}) | > | PRE(\underline{\tau_\gamma}) |$. Consequently, $\tau_\beta \; \alpha \; \tau_\gamma$ implies that τ_γ temporally precedes τ_β.[3]

Consistent with this definition, for the next time instant $\tau_{\beta+1}$ of τ_β, $\tau_{\beta+1} \; \alpha \; \tau_\beta$ holds and τ_β is called the immediate subordinate of $\tau_{\beta+1}$; for any other time instants such that $\tau_\beta \; \alpha \; \tau_\gamma$ holds, τ_γ is called the subordinate of τ_β.

Moreover, PLTL requires the linear, discrete, and *absolute* view of time. Whenever there is an order-equivalent isomorphic mapping $I : w \rightarrow T_\aleph$ matching the index set of a sequence w to the time line $< T_\aleph, \mathbf{R} >$, the function I is an *interpretation* and w is a sequence of observed phenomena corresponding to the linear, discrete view of

---

[2] This differs from *continuous time*, where a time instant τ_k can be selected from between any two time instants τ_i and τ_j.

[3] The term "α-relation" is chosen since the ordinal number α is involved with the ordering among the time instants.

time.[4] The notion of absolute time requires that only a single determinate sequence w of phenomena from an interpretation I is observed. But, the study of the potential observations requires the definition of the α-relation among a number of *alternative* interpretations.[5] Linear, discrete, and absolute *temporal analysis* defines the *α-relation structure* of alternative interpretations using elements of the set W for their components. It is said that the α-relation structure is defined over W, where each interpretation is implied by the α-relation.

Temporal analysis is the foundation upon which the PLTL language and validation is based. It is the definition of the α-relation structure over the global state holdings S of the reachability graph and their corresponding atomic propositions. It is based on the notion that given the present observation of any alternative phenomena, the sets of observed phenomena sequences leading up to the present observation can be identified. The remaining sections construct the α-relation structure.

[4] The time instants, e.g. τ_α, are actually a notation for a generic form of observed phenomena.

[5] In contradistinction to a set of sequences, the notion of *relative time* posits a tree structure of time, where paths through the tree represent sequences of observed phenomena.

Section 4-2 : The Well Founded α-Relation

The definition of temporal analysis must be extended to solve one difficulty. Since walks of finite and transfinite length can be defined through a reachability graph, and since validation is performed over both, temporal analysis must define a finite α-structure that contains sufficient information for validation purposes.

A certain type of α-relation is required.

Definition 4-2-1 : A Well Founded α-Relation [LPS]

A *well founded* binary α-relation α over a set S implies that, for any infinite sequence $(s_0, s_1, \ldots, s_i, s_{i+1}, \ldots)$ such that $s_i \in S : i \geq 0$, it is not the case that $s_{i+1} \; \alpha \; s_i$ holds for all $i \in 0, 1, \ldots$; if the relation α is a total order, then $< S, \alpha >$ is a *well founded structure.*

A structure $< \alpha, > >$ is a well founded structure, where $\alpha \in$ ORD represents the set of all ordinals smaller than α and $>$ is the standard ordering relation over ORD. For $A \subseteq S$, a number of alternative well founded structures $< A, \alpha >$ are defined.

The implication of the well founded α-relation is that no matter what element $s \in S$ is assigned to the present time instant, the well founded α-relation defines certain future and past finite sequences of elements based on s.[6] For example, given an infinite sequence $(s_0, s_1, \ldots, s_i, s_{i+1}, \ldots)$ from an interpretation, an element $s_i : i \geq 0$ can be associated with the present time instant. For $k < \omega$ and $0 \leq j \leq k$, there is a sequence portion $(s_j, s_{j+1}, \ldots, s_{k-1}, s_k)$, such that $s_i = s_k$ or $s_i = s_j$, and where

$$s_k \; \alpha \; s_{k-1}, \; s_{k-1} \; \alpha \; s_{k-2}, \; \ldots, \; s_{j+2} \; \alpha \; s_{j+1}, \text{ and } s_{j+1} \; \alpha \; s_j$$

[6] i.e., when s temporally precedes s', for s and s' $\in$ S, then s can be assigned to a time instant that temporally precedes the time instant associated with s'.

may hold; by transitivity, $s_k \alpha s_j$ holds. By virtue of observing the present phenomena s_k, the observations that may have lead up to the present can be identified. Conversely, by virtue of observing the present phenomena s_j, the potential future observations leading to s_k may also be identified.

Definition 4-2-2 : α-Minimal Element [LPS]

Given that the well founded α-relation is defined over S, any non-empty set $A \subseteq S$ has at least one *α-minimal* element $s \in A \ni \neg\exists s' \in A \ni s \alpha s'$. The set of all α-minimal elements of a subset A is $MIN(A) =\mathbf{df} \{ s \ni \neg\exists s' \in A, s \alpha s' \}$.

That is, the α-minimal element s is associated with a time instant that temporally precedes those associated with all other elements in a well founded structure defined over A. The fact that s is an α-minimal element does not preclude s being associated with other time instants. The α-minimal element must exist by definition of the well founded relation α.

The remainder of this chapter is devoted to the definition of a well founded α-relation; the techniques of [LPS] based on the fundamental Theorem 1. and proof are required. The following theorem shows that every well founded relation α can be mapped into a well founded structure $< \alpha, > >$, viz. the time line.

Theorem 4-2-3 : A Well Founded Relation α Over the Set of States S can be Mapped into the Time Line $< T_{\aleph}, R >$.

A well founded α-relation α is defined over S. There exists an ordinal α and a *ranking* function $\rho : S \rightarrow \alpha$ such that :

i) $s \alpha s' \Rightarrow \rho(s) > \rho(s')$;

ii) if, for every $s'' \in S$, $s' \alpha s'' \Rightarrow s \alpha s''$ then $\rho(s) \geq \rho(s')$.

Proof : By a Means Similar to Theorem 1. [LPS].

The following, possibly transfinite, sequence is defined :

$S_0 =\mathbf{df} MIN(S)$,

$S_1 =\mathbf{df}\ \mathrm{MIN}(\ S - \cup_{i<1} S_i\ ),$

... ,

$S_k =\mathbf{df}\ \mathrm{MIN}(\ S - \cup_{i<k} S_i\ ),$

... .

By transfinite induction for every limit ordinal β,

$$S_\beta =\mathbf{df}\ \mathrm{MIN}(\ S - \cup_{\gamma<\beta} S_\gamma\ ).$$

The *ranking function* is $\rho : S \rightarrow \alpha \ni \rho(s) =\mathbf{df}\ \gamma \Leftrightarrow s \in S_\gamma$.

The ranking function ρ satisfies cases i) and ii).

i) $s\ \alpha\ s' \Rightarrow \rho(s) > \rho(s')$:

Because, in any well founded structure $< S, \alpha >$, $s\ \alpha\ s'$ implies that s' temporally precedes s, the sequence $S_0, S_1, \ldots, S_\omega, \ldots$ implies that $s \in S_\psi$ and $s' \in S_\gamma$ for $\psi > \gamma$; by the definition of the ranking function, $\rho(s) > \rho(s')$.

ii) if, for every $s'' \in S$, $s'\ \alpha\ s'' \Rightarrow s\ \alpha\ s''$ then $\rho(s) \geq \rho(s')$:

The statement <u>for every $s'' \in S$, $s'\ \alpha\ s'' \Rightarrow s\ \alpha\ s''$</u> implies that every s'' is α-minimal in the set $\{ s, s', s'' \} \subseteq S$. Since s'' can be any element of S, including either s or s', or both, the only way $s'\ \alpha\ s'' \Rightarrow s\ \alpha\ s''$ can be guaranteed is if s and s' are one in the same and $\rho(s) = \rho(s')$ or s and s' are different and $\rho(s) > \rho(s')$; for example, if $\rho(s') > \rho(s'') > \rho(s)$, $s'\ \alpha\ s''$ holds but does not imply $s\ \alpha\ s''$; Therefore, the statement $\rho(s) \geq \rho(s')$ must hold. **Q.E.D.**

There are a number of consequences of this theorem. If S has finite cardinality n, then the range of ρ is a finite set $\{ 0, 1, 2, \ldots, n - 1 \}$. If, for every $s \in S$, the degree of α, i.e. set $\{ s' \ni s\ \alpha\ s' \}$, has finite cardinality, then the range of ρ may be taken as a subset of $\mathbb{N}$. If S is countable, the range of ρ is a countable ordinal. There exists an ordinal α such that $\forall s \in S, \rho(s) < \alpha$; in this case, $\alpha =\mathbf{df}\ \omega$.

Section 4-3 : Generating Countable Walks

Here, the observed phenomena are validation model global states, conveniently structured by the reachability graph < S, T, m_0, M_f, H >. Temporal analysis defines a well founded α-relation over S, such that MIN(S) =**df** m_0, whose set of alternative interpretations are countably finite and infinite walks through the global states S of the reachability graph. The approach of [CV] is used, defining a *λ-free Petri net language* [Hac] to generate the sets $S_N(m_0)$ and $T_N(m_0, M_f)$, Definition 3-4-1, for the validation model N. The validation model transitions Σ comprise the *finite alphabet* of the Petri net languages. Because Σ corresponds directly to T, each word in the Petri net language specifies a path of arcs in T generating a walk. Since the walks are countable, the following definitions use $\mathbb{N}$ to index sequences.

Definition 4-3-1 : Countably Finite Language

The *countably finite language* of a validation model N is the set of all words

$L_\lambda(N)$ =**df** { $\sigma_\circ \in \Sigma^*$ $\ni$ $\sigma_\circ \in S_N(m_0)$ }.

Definition 4-3-2 : Terminal Petri net Language

The *terminal Petri net language* $L'_\lambda(N)$ of a validation model N is the set of all words

$L'_\lambda(N)$ =**df** { $\sigma_\circ \in \Sigma^*$ $\ni$ $\sigma_\circ \in T_N(m_0, M_f)$ }.

Definition 4-3-2 defines words corresponding to transition firing sequences that put the validation model into a final state. Clearly, $L'_\lambda(N) \subseteq L(N)$.

Definition 4-3-3 : Countably Infinite Languages

The *countably infinite language* of a validation model N is the set of all words

$L_\omega(N)$ =**df** { $\sigma_\omega \in \Sigma^\omega$ $\ni$ $\sigma_\omega[i] \in S_N(m_0), i \in \mathbb{N}$ }.

Each word in the countably finite language corresponds to a path in the reachability graph. Since the arcs T correspond to the firing of a single transition, the words in Σ explicitly denote the sequence of arcs comprising the path.

Definition 4-3-4 : A Countably Finite Path

A countably finite word $\sigma_\circ : \{ 1, \ldots , k \} \rightarrow \Sigma$ such that $\sigma_\circ =$**df** $(t_1, \ldots , t_k) \in \Sigma^*$ generates a *countably finite path*

$C =$**df** $\{ c_1 = (m_0, s_1), c_2 = (s_1, s_2), \ldots , c_k = (s_{k-1}, s') : c_1, c_2, \ldots , c_k \in T \}$

of a reachability graph iff there exists $m_0, s_1, \ldots , s_{k-1}, s' \in (S \wedge RM_N(m_0))$, such that there is countably finite transition firing sequence $m_0[\sigma_\circ > s'$ in a validation model N denoted $m_0[t_1 > s_1, s_1[t_2 > s_2, \ldots , s_{k-1}[t_k > s'$.

Definition 4-3-5 : A Countably Infinite Path

A *countably infinite path*, viz. a countably infinite transition firing sequence, of a reachability graph is an infinite word $\sigma_\omega =$**df** $t_1, \ldots , t_k, \ldots \in \Sigma^\omega$, such that each prefix $\sigma_\omega[i]$, where $i \in \mathbb{N}$, is a countably finite path.

The set of all countably finite paths of a reachability graph RG is denoted $F_\circ(RG)$ and the set of all countably infinite paths of a reachability graph RG is denoted $F_\omega(RG)$. For a reachability graph RG generated from a validation model N, $F_\circ(RG)$ is defined by $L_\circ(N)$ and $F_\omega(RG)$ is defined by $L_\omega(N)$.

It is useful to have a precise notation of the sequence of global states comprising the walks through the reachability graph.

Definition 4-3-6 : Countably Finite (Infinite) Walk

The *walk* generated by the countably finite (infinite) transition firing sequence $\sigma_\circ$ (σ_ω) of a validation model N, where $\sigma_\circ \in F_\circ(N)$ ($\sigma_\omega \in F_\omega(N)$), is the mapping $\delta_\circ(\sigma_\circ) : \{ 1, \ldots , | \sigma_\circ | + 1 \} \rightarrow S$ ($\delta_\omega(\sigma_\omega) : \mathbb{N} \rightarrow S$), such that

$\delta_\circ(\sigma_\circ)(1) = m_0$ ($\delta_\omega(\sigma_\omega)(1) = m_0$) and, for $1 < i \leq | \sigma_\circ | + 1$ ($\forall i \in \mathbb{N} \ni i > 1$),

there exists an $s \in S$ where $\delta_\circ(\sigma_\circ)(i) = s$ ($\delta_\omega(\sigma_\omega)(i) = s$), where

$m_0[\sigma_\circ[i] > s$ ($m_0[\sigma_\omega[i] > s$).

To denote the collection of all countably finite (infinite) walks generated by a validation model N for the corresponding reachability graph RG, the notation $\Delta_\circ(RG)$ ($\Delta_\omega(RG)$) is used.

Definition 4-3-7 : Collections of Countably Finite (Infinite) Walks

As per Definition 2-2-3 : Collection of Sequences, since for a(n) finite (infinite) sequence $\sigma_\circ$ (σ_ω) the i-th element can be identified, viz. $\sigma_\circ(i)$ ($\sigma_\omega(i)$), the following definitions are made :

i) the set of i-th components of the sequences in $\Delta_\circ(RG)$ ($\Delta_\omega(RG)$) is denoted $\Delta_\circ(RG)_i \ni i \in I \subset \mathbb{N}$ ($\Delta_\omega(RG)_i \ni i \in \mathbb{N}$);

ii) the union is $\cup_{i \in I \subset N} \Delta_\circ(RG)_i$ ($\cup_{i \in N} \Delta_\omega(RG)_i$);

iii) the intersection is $\cap_{i \in I \subset N} \Delta_\circ(RG)_i$ ($\cap_{i \in N} \Delta_\omega(RG)_i$);

iv) the product $\times_{i \in I \subset N} \Delta_\circ(RG)_i$ is the set of finite choice functions

$\{ \delta_\circ(\sigma_\circ) \ni \delta_\circ(\sigma_\circ) : I \rightarrow \cup_{i \in I} \Delta_\circ(RG)_i$, where $\delta_\circ(\sigma_\circ)(i) \in \Delta_\circ(RG)_i$ for all $i \in I$ $\}$; thus, $\delta_\circ(\sigma_\circ)$ defines the tuple, a countably finite walk of cardinality $| \sigma_\circ |$, $(\delta_\circ(\sigma_\circ)(1), \ldots , \delta_\circ(\sigma_\circ)(| \sigma_\circ |)) \in \times_{i \in | \sigma_\circ |} \Delta_\circ(RG)_i$; furthermore, the prefixes of $\delta_\circ(\sigma_\circ)$ define walks in $\Delta_\circ(RG)_i$ of lesser cardinality;

v) the product $\times_{i \in N} \Delta_\omega(RG)_i$, is the set of infinite choice functions

$\{ \delta_\omega(\sigma_\omega) \ni \delta_\omega(\sigma_\omega) : \mathbb{N} \rightarrow \cup_{i \in N} \Delta_\omega(RG)_i$, where $\delta_\omega(\sigma_\omega)(i) \in \Delta_\omega(RG)_i$ for all $i \in \mathbb{N}$ $\}$; thus, $\delta_\omega(\sigma_\omega)$ defines the tuple, a countably infinite walk of cardinality $\aleph_0$, $(\delta_\omega(\sigma_\omega)(1), \ldots , \delta_\omega(\sigma_\omega)(n), \ldots) \in \times_{i \in N} \Delta_\omega(RG)_i$.

Clearly the prefixes of all $\delta_\omega(\sigma_\omega) \in \times_{i \in N} \Delta_\omega(RG)_i$ defines $\times_{i \in I \subset N} \Delta_\circ(RG)_i$. An infinite set of finite predicates $A_i(v_1, v_2, \ldots v_i)$ is defined, whose sentences define countably finite walks.

Section 4-4 : Restricting Countably Infinite Walks.

The purpose of correctness analysis is to show that the execution of the validation model meets the requirements of the correctness criteria PLTL formulae. Certain conditions, defined by *standards of conduct* [LPS], can be imposed on the validation model execution. For the simple reachability graph illustrated in Figure 4-1, a liveness correctness criterion could be that the validation model begins in global state s_1 and eventually terminates by holding global state s_2.

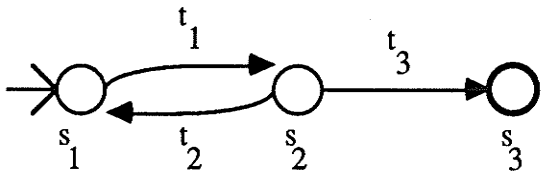


Figure 4-1 : A Reachability Graph Producing a Race Condition

Unfortunately, this reachability graph generates the countably infinite path $(t_1t_2)^\omega$, which produces a countably infinite walk that alternates between s_1 and s_2; this is a simple race condition, which does not meet the correctness criteria. One could reason that the validation model is flawed, since it generates a walk where the final state s_3 is never held; however, another approach is to reason that such walks would not occur under certain conditions. Close examination of the path $(t_1t_2)^\omega$ reveals that transition t_3 is enabled an infinite number of times, but is never fired. While there is no reason why the walk resulting from $(t_1t_2)^\omega$ can not be considered acceptable, the walk can be removed from consideration by imposing the condition that any transition enabled an infinite number of

times is eventually fired. Correctness analysis under such a condition of execution would show that the validation model honours the correctness criterion.

In [LPS] the concepts of *impartial*, *fair*, and *just* standards of conduct are defined, based on explicit definitions of a well founded relation α, to prove the termination of multiple, concurrently executing processes. The methods in [LPS] are extended here for cyclic validation models.[7] This dissertation employs the study of infinite behaviour of Petri nets in [CV] derived from the concepts of [LPS] to select those countably infinite walks deemed acceptable.

Two preliminary definitions are required to define the concepts of impartial, fair, and just standards of conduct in infinite walks. First, a means of determining the number of times a transition t is enabled in a given walk $\delta_\omega(\sigma_\omega)$ is required.

Definition 4-4-1 : The Enabling Counter

The *enabling counter* for an infinite firing sequence σ_ω and a transition t is the mapping

$EN : F_\omega(RG) \times \Sigma \rightarrow \mathbb{N}$, where $EN(\sigma_\omega, t) =$**df** $| \{ i \in \mathbb{N} \ni \delta_\omega(\sigma_\omega)(i) \geq I(t, -) \} |$.

Here I(t, -) denotes the row of the input function I corresponding to the transition t. The relation $\delta_\omega(\sigma_\omega)(i) \geq I(t, -)$ corresponds to the firing rule defined in Definition 3-3-2 and only the entries of the row I(t, -) greater than zero are inspected.

Definition 4-4-2 : The Continuity Predicate

The *continuity predicate* $CON(\sigma_\omega, t)$ is true when

$\exists i \in \mathbb{N}$, where $\forall j \in \mathbb{N} \ni j \geq i$, $\delta_\omega(\sigma_\omega)(j) \geq I(t, -)$ and is false in any other situation.

$CON(\sigma_\omega, t)$ is true when the transition t is enabled at some point in the infinite transition firing sequence σ_ω and enabled continuously thereafter.

---

[7] the condition of execution imposed in the previous paragraph conforms the the impartial standard of conduct.

The infinite paths of the reachability graph RG specified by the corresponding validation model honouring the impartial, just, and fair standards of conduct can now be defined.

Definition 4-4-3 : Impartial Path

The *impartial paths* of RG are defined as

$$F_{\omega\text{–impartial}}(RG) =\mathbf{df} \{ \sigma_\omega \in \Sigma^\omega \ni \forall t \in \Sigma, EN(\sigma_\omega,t) = \omega \}.$$

Definition 4-4-4 : Just Paths

The *just paths* of RG are defined as

$$F_{\omega\text{–just}}(RG) =\mathbf{df} \{ \sigma_\omega \in \Sigma^\omega \ni \forall t \in \Sigma, CON(\sigma_\omega, t) \Rightarrow t \in INF(\sigma_\omega) \}.$$

Definition 4-4-5 : Fair Paths

The *fair paths* of RG are defined as

$$F_{\omega\text{–fair}}(RG) =\mathbf{df} \{ \sigma_\omega \in \Sigma^\omega \ni EN(\sigma_\omega,t) = \omega \Rightarrow t \in INF(\sigma_\omega) \}.$$

The languages generated by a validation model N are $L_{\omega\text{–impartial}}(N)$, $L_{\omega\text{–just}}(N)$, and $L_{\omega\text{–fair}}(N)$, respectively.

Definition 4-4-6 : Impartial, Just, and Fair Walks

The set of countably infinite walks through a reachability graph RG specified by the paths $F_{\omega\text{–impartial}}(RG)$, $F_{\omega\text{–just}}(RG)$, and $F_{\omega\text{–fair}}(RG)$ are denoted $\Delta_{\omega\text{-impartial}}(RG)$, $\Delta_{\omega\text{-just}}(RG)$, and $\Delta_{\omega\text{-fair}}(RG)$, respectively.

It is useful to speak of countably infinite transition firing sequences and countably infinite walks generated by a validation model honouring a standard of conduct, without explicitly naming one of impartial, just, and fair [LP]. Accordingly, $F_{\omega-\alpha}(RG)$ denotes the countably infinite *α-paths* of RG and $\Delta_{\omega-\alpha}(RG)$ denotes the countably infinite *α-walks* specified by the α-paths.

Section 4-5 : Defining the Well Founded α-Relation.

The task of explicitly defining the well founded α-relation α over the reachable global states S of the strongly connected reachability graph $< S, T, m_0, M_f, H >$ is dealt with here. The countably infinite walks $w \in \Delta_{\omega-\alpha}(RG)$ consist of an infinite number of finite prefixes. Each finite walk portion can be specified by a sentence of a predicate.

Definition 4-5-1 : Walk Predicates

A portion of length i of a countably infinite walk is a sentence of an i-ary *walk predicate* $A_i(v_1, v_2, \ldots , v_i)$. For each predicate, v_1 is called the *leading variable* and v_i is called the *trailing variable*; likewise, for a sentence

$$(s_1, s_2, \ldots , s_i) \text{ of } A_i(v_1, v_2, \ldots , v_i),$$

s_1 is called the *leading term* and s_i is called the *trailing term.* A walk predicate $A_i(v_1, v_2, \ldots , v_i)$ implies the set of *subordinate predicates*

$$A_i(v_1, v_2, \ldots , v_{i-1}), \ldots , A_2(v_1, v_2), A_1(v_1).$$

$\Delta_{\omega-\alpha}(RG)$ defines an infinite set of finite predicates $A_i(v_1, v_2, \ldots , v_i) : i < \omega$.

The impartial, just, and fair standards of conduct select infinite sequences, whose finite portions are sentences of the predicates $A_i(v_1, v_2, \ldots , v_i) : i < \omega$. The countably infinite walks $w \in \Delta_{\omega-\alpha}(RG)$ can be mapped to the time line by defining interpretations $I : w \rightarrow T_{\aleph}$; therefore, as in Section 4-1, the α-relation can be defined over the components of each infinite walk and each finite portion.

Since the countably infinite walks $\Delta_{\omega-\alpha}(RG)$ are sequences in the finite set S of global states, for every $\delta_\omega(\sigma_\omega) \in \Delta_{\omega-\alpha}(RG)$, an infinity set $INF(\delta_\omega(\sigma_\omega))$ is defined. Likewise, an infinity set $INF(\Delta_{\omega-\alpha}(RG))$ is defined, consisting of all those global states that appear infinitely often in every infinite walk;

$$\text{i.e., } INF(\Delta_{\omega-\alpha}(RG)) =\mathbf{df} \cap_{\delta\omega(\sigma\omega) \in \Delta\omega-\alpha(RG)} INF(\delta_\omega(\sigma_\omega)).$$

Therefore, $INF(\Delta_{\omega-\alpha}(RG))$ defines a strongly connected component of the reachability graph with an equivalence relation $\approx$, where, for every x, $y \in INF(\Delta_{\omega-\alpha}(RG)) \cap S$, $x \approx y$ implies the existence of a set of path portions from $F_{\omega-\alpha}(RG)$ defining a finite walk from x to y and y to x. By definitions of the standards of conduct, and since the reachability graph is strongly connected, $INF(\Delta_{\omega-\alpha}(RG)) = S$; hence, the equivalence relation $\approx$ holds over S. Therefore, what is required is the minimal set F_{min} of finite path portions of $F_{\omega-\alpha}(RG)$ that specify the equivalence relation $\approx$ over S; F_{min} must define all elementary paths and cyclic elementary paths of the reachability graph. Transitivity of the equivalence relation $\approx$ implies that longer path protions can be defined from the minimal set.

The equivalence relation $\approx$ is defined among the leading and trailing terms of all sentences. All longer finite walk portions are constructed using transitivity. The elementary and cyclic elementary paths select certain walks.

Definition 4-5-2 : Acceptable Walk Portions

A *cyclic walk portion* is a sequence of global states

$$(s_j, s_{j+1}, \cdots, s_{j+i}) : j + i \ni j + i =\mathbf{df} k, k \geq j,$$

where the trailing term s_k duplicates the leading term s_j. Cyclic walk portions are *acceptable* iff the trailing and leading terms are the only duplicates. Only portions that do not duplicate a component, except those acceptable cyclic walk portions, are *acceptable.*

Definition 4-5-3 : The Maximal Length of an Acceptable Portion

Since the reachability graph is loop free, the *maximal length* of a portion of an infinite walk is a cyclic walk portion of length $| S | + 1$.

The predicate large enough to contain a maximal sentence of length $n \leq | S | + 1$ and the family of subordinate predicates $\{ \mathbf{A}_i : 1 \leq i \leq n \}$ are required. Not all subordinate

predicates may have a sentences conforming to acceptable cyclic walk portions, but the sentences of maximal predicate must be acceptable cyclic walk portions.

Definition 4-5-4 : SCC-$\Delta_{\omega-\alpha}$(RG) Acceptable Portions

The acceptable walk portions of $\Delta_{\omega-\alpha}$(RG) specified by $\{ A_i : 1 \leq i \leq n \}$ are called *SCC-$\Delta_{\omega-\alpha}$(RG) acceptable portions.*

The SCC-$\Delta_{\omega-\alpha}$(RG) acceptable portions define an equivalence relation among the leading and trailing terms of all sentences.

A well founded binary α-relation is defined among the sentences of $A_1(v_1)$ to define the minimal set of finite predicates $A_n(v_1, \ldots, v_i) : n > 2$. Thus, the sentences of predicates

$$A_n(v_1, v_2, \ldots, v_i), \ldots, A_2(v_1, v_2), A_1(v_1)$$

are portions $(s_j, s_{j+1}, \ldots, s_{k-1}, s_k) : k > j$, such that $s_i = s_k$ or $s_i = s_j$, of infinite walks $\Delta_{\omega-\alpha}$(RG).

Definition 4-5-5 : The Well Founded α-Relation

A *well founded α-relation* is defined over all distinct elements of S, sentences of A_1. The predicates $\{ A_i : 2 \leq i \leq n \}$ correspond to the SCC-$\Delta_{\omega-\alpha}$(RG) acceptable portions. For every leading term s and trailing term s' of a sentence, s' α s holds and specifies at least one acceptable portion of SCC-$\Delta_{\omega-\alpha}$(RG) using a path $\sigma_\circ$ such that $s[\sigma_\circ > s'$; s α s holds implies a cyclic path $\sigma_\circ$.

Temporal analysis uses Definition 4-5-5 to define future executions of the validation model in terms of the linear, discrete, and absolute view of time.

Lemma 4-5-6 : The Well Founded α-relation is an Equivalence Relation.

Proof : By Construction.

The paths F_{min} select portions of walks in SCC-$\Delta_{\omega-\alpha}$(RG). The acceptable portions allow all global states S to be a leading or trailing term. Reflexitivity, symmetry, and transitivity clearly hold. **Q.E.D.**

Lemma 4-5-6 is of critical significance to this dissertation.

For example, Figure 4-2 shows a simple reachability graph, Figure 4-2(a), and the α-relation defined over cyclic portions, Figure 4-2(b), for the graph.

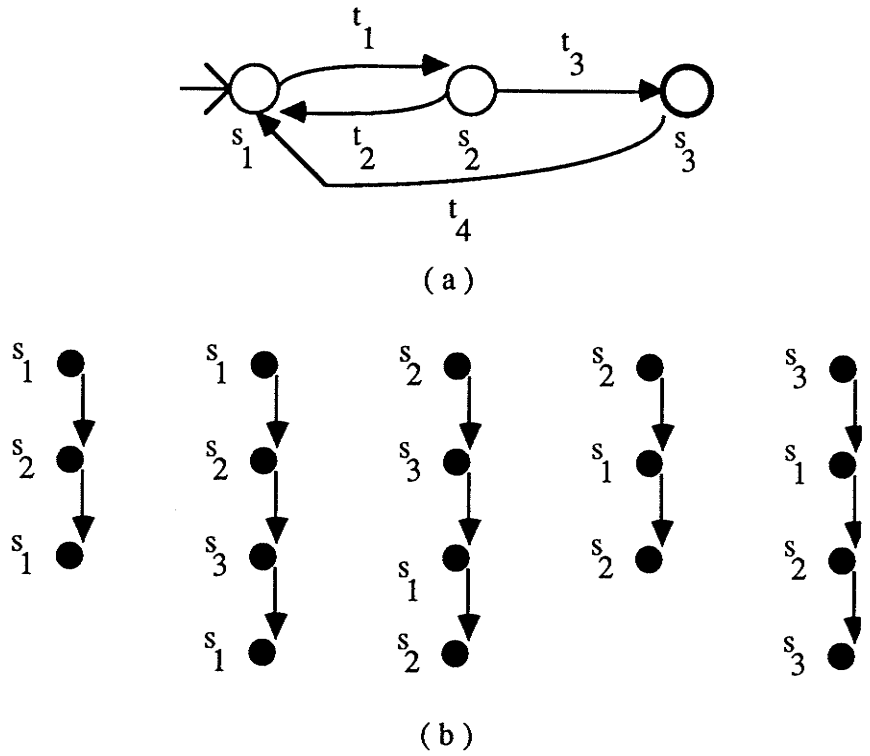


Figure 4-2 : Well Founded α-relation Example.

In figure 4-2(b), the dots represent time instants and the arrows represent the α-relation. The α-relation requires four predicates :

$$A_1(v_1),\ A_2(v_1, v_2),\ A_3(v_1, v_2, v_3),\ \text{and}\ A_4(v_1, v_2, v_3, v_4),$$

whose sentences define the acceptable portions. The α-relation and the corresponding portions for Figure 4-2 follow.

s_1	α	s_1	s_1	s_2	s_1	s_2	α	s_1	s_2	s_1
			s_1	s_2	s_3 s_1				s_2	s_3 s_1
s_1	α	s_2	s_1	s_2		s_2	α	s_2	s_2	s_1 s_2
s_1	α	s_3	s_1	s_2	s_3				s_2	s_3 s_1 s_2
						s_2	α	s_3	s_2	s_3
s_3	α	s_1	s_3	s_1						
s_3	α	s_2	s_3	s_1	s_2					
s_3	α	s_3	s_3	s_1	s_2 s_3					

It is clear, that a well founded relation can be defined for any portion of maximal length n of any walk in $\Delta_{\omega-\alpha}(RG)$, by applying Theorem 4-2-2. An infinite walk can be constructed with an infinite concatenation of acceptable portions. However, the task of temporal analysis is not complete, because the SCC-$\Delta_{\omega-\alpha}(RG)$ acceptable portions must be expressed in a form compatible with PLTL. This is the subject of the next section of this chapter.

Section 4-6 : The Validation Model α-Relation Structure.

PLTL is a language for reasoning and inference about phenomena arranged in a temporal structure. *Model theory* [Bri] is based on the definition of a *relation structure*, a generic mathematical structure that underlies any logic in general, and PLTL in particular. Any logic should be sufficiently comprehensive to express its relation structure [Rob77]; indeed, PLTL is a language that can express its relation structure. Here, the relation structure describes α-SCC-Δ_ω(RG) acceptable portions, providing temporal structure for PLTL.

Definition 4-6-1 : Relation Structure

A *relation structure* [Bri][Rob73][Sac] is a 4-tuple,

$Z =_{\mathbf{df}} < Z , \{ A_i \}_{i \in I} , \{ f_j \}_{j \in J} , \{ c_k \}_{k \in K} >$, where :

i) Z is a non-empty *domain*;

ii) $\{ \mathbf{A}_i \}_{i \in I}$ is a family of *predicates* defined over the domain, with a function $\lambda : I \rightarrow \mathbb{N}$ such that, for each $i \in I$, $\lambda(i) \geq 0$ and $\mathbf{A}_i$ is a $\lambda(i)$-ary relation on Z; i.e., $\mathbf{A}_i \subseteq Z^{\lambda(i)}$;

iii) $\{ f_j \}_{j \in J}$ is a family of *choice functions*, with a function $\mu : J \rightarrow \mathbb{N}$ such that, for each $j \in J$, $\mu(j) \geq 0$ and a given f_j is a $\mu(j)$-ary function in Z;

iv) $\{ c_k \}_{k \in K}$ is a set of *distinguished elements*, with a possibly empty set K of integers such that, for $k \in K$, c_k is associated with a corresponding element of the domain Z.

I, J, and K are the respective indexing sets for the families of relations, functions, and distinguished elements of the relation structure. The $\lambda(i)$-ary relation $\mathbf{A}_i$ with domain Z is a $\lambda(i)$-tuple $(z_1, \ldots , z_{\lambda(i)})$, where z_j is called a *component with index j*. A function f_j has the domain of $\mu(j)$-ary tuples of z and a range of single elements of Z; therefore, a

function f_j can be thought of as a $(\mu(j) + 1)$-ary relation. A distinguished element c_k is a unary relation that associates some item with an element from the domain. Hence, a relation structure could be defined completely in terms of relations; it is only for convenience that functions and distinguished elements are used.

Particular classes of relation structures can be identified.

Definition 4-6-2 : The Relation Structure Type

The *type* of a relation structure Z is the tuple θ_Z =**df** $< \lambda, \mu, K >$, where λ and μ are functions associated with the families of predicates and functions respectively, and where K is the set of integers corresponding to the distinguished elements. Two structures Z and Y have the *same type* iff $\theta_Z = \theta_Y$. Two structures Z and Y are of *similar type* iff θ_Z =**df** $< \lambda, \mu, K >$, θ_Y =**df** $< \lambda', \mu', K' >$, and there are bijective mappings $h_1 : I \rightarrow I'$, $h_2 : J \rightarrow J'$, and $h_3 : K \rightarrow K'$.

When two relation structures Z and Y are of similar type, then Z and Y are isomorphic.

Definition 4-6-3 : Relation Substructure

If the relation structures Z and Y have the same type, then

Y =**df** $< Y, \{ \mathbf{B}_i \}_{i \in I}, \{ g_j \}_{j \in J}, \{ d_k \}_{k \in K} >$ is a *substructure* of

Z =**df** $< Z, \{ \mathbf{A}_i \}_{i \in I}, \{ f_j \}_{j \in J}, \{ c_k \}_{k \in K} >$ iff :

i) $Y \subseteq Z$;

ii) the family of relations in Y, $\{ \mathbf{B}_i \}_{i \in I}$, is the *restriction* [Hal][Bri] of the family of predicates in Z, $\{ \mathbf{A}_i \}_{i \in I}$, to the domain of Y; i.e., for every $i \in I$, $\mathbf{A}_i = \mathbf{B}_i \cap Y^{\lambda(i)}$;

iii) given any family of functions $\{ g_j \}_{j \in J}$ in Y and any family of functions $\{ f_j \}_{j \in J}$ in Z, for each $y_1, \ldots, y_{\mu(j)} \in Y$ and for every $j \in J$, it is the case that $g_j(y_1, \ldots, y_{\mu(j)}) = f_j(y_1, \ldots, y_{\mu(j)})$;

iv) for each $k \in K$, $c_k = d_k$; each distinguished element of Y is a distinguished element of Z.

Given a relation structure, Definition 4-6-3 shows that a substructure can be extracted from the relation structure.

The α-relation is defined in terms of the relation structure. The domain Z consists of the global states S and the families of predicates and choice functions define the α-relation among the domain. The global states are abstract representations of phenomena observed of the validation model. The atomic propositions assigned to the global states by the function $H : S \rightarrow 2^P$ are a more explicit meanigs to the global states.

Definition 4-6-4 : Atomic Tableau

An *atomic tableau* is the set of atomic propositions, denoted a_i, that are mapped to the value of a variable v_i; i.e., a_i =**df** $H(v_i)$.

The PLTL language is based on sequences of atomic tableaux.

Therefore, the well founded α-relation is specified in terms of the atomic tableaux. The predicate of the well founded α-relation $\mathbf{A}(v_1, v_2, \ldots , v_{\lambda(i)})$ defines a subset of all sentences, those sequences of bound variables $(a_1, a_2, \ldots , a_{\lambda(i)})$ with cardinality $\lambda(i)$, where $a_i \subseteq P : 1 \leq i \leq \lambda(i)$ explicitly defines an atomic tableaux of the sentence.

Definition 4-6-5 : Finite Linear Ordered Atomic Tableaux

The sequence of *finite linear ordered atomic tableaux*, abbr. FLOAT, results from mapping each sentence in a predicate $\mathbf{A}_{\lambda(i)}(v_1, v_2, \ldots , v_{\lambda(i)})$ with the function H;

i.e., a FLOAT T =**df** $H(\mathbf{A}_{\lambda(i)}(v_1, v_2, \ldots , v_{\lambda(i)}))$ =**df**

$(H(v_1), H(v_2), \ldots , H(v_{\lambda(i)}))$ =**df** $(a_1, a_2, \ldots , a_{\lambda(i)})$.

Thus, a member of the α-relation a' α a corresponds to a number of sentences of the well founded α-relation, FLOATS of maximal length n, where a is called the *leading tableau* and a' is called the *trailing tableau* of the sequence. Those reflexive relations

a' α a where a = a', viz. a $\subset$ a' and a' $\subset$ a, are called *cyclic* FLOATs. The notation T =**df** (a_1, a_2, ... , $a_{\lambda(i)}$) explicitly lists of the sequence of atomic tableaux ($H(v_1)$, $H(v_2)$, ... , $H(v_{\lambda(i)})$) comprising a sentence. A FLOAT is a countably finite sequence T, its cardinality $| T |$ =**df** $\lambda(i)$.

The relation structure for a validation model N is constructed from the SCC-Δ_ω(RG) acceptable portions.

Definition 4-6-6 : The Validation Model α-Relation Structure Ξ

A *validation model α-relation structure* Ξ of type θ =**df** $< \lambda, \mu, | \boldsymbol{P} | >$ constructed from the reachability graph corresponding to the validation model N is the 4-tuple

Ξ =**df** $< S, \{ A_i : 1 \leq i \leq n \}, \text{SCC-}\Delta_{\omega-\alpha}(RG), \{ H(s) \ni \forall s \in S \} >$ where :

i) S is the domain of global states of the reachability graph;

ii) $\{ A_i : 1 \leq i \leq n \}$ is the family of predicates corresponding to the α-relation;

iii) α-SCC-Δ_ω(RG) walks are choice functions for the family of predicates;

iv) $\{ H(s) \ni \forall s \in S \}$ is the set of 2^P mapped by the function H to each element of the domain S.

The validation model α-relation structure Ξ has type $\theta_\Xi = < \lambda, \mu, | \boldsymbol{P} | >$. The function λ determines the length of the maximal acceptable portions portions up to a maximum of n. The function μ determines the the length of predicate sentences up to a maximum of n. Since the variables assume values of global states mapped to the time line, α denotes the binary relation of temporal precedence among the variables; thus, given two variables v_i and v_j, i < j, v_{n-1} α v_n can be written.

The well founded α-relation is defined among the atomic tableaux. The predicate $A_i(v_1, v_2, ... , v_i) : 1 \leq i \leq n$ defines a set of FLOATs of length i defined by sentences of a predicate $\Xi_i(v_1, v_2, ... , v_i) : 1 \leq i \leq n$. A system of FLOATs of cardinalities no greater than some maximal cardinality n can be defined. Accordingly,

there is a sequence $\Xi_1, \Xi_2, \ldots, \Xi_n$ of such sets, where Ξ_1 is the set of atomic tableaux. It follows from definition of the α-relation, Ξ_n must contain only cyclic FLOATs. Moreover, $\Xi^j : 1 \leq j \leq n$ denotes the set of all FLOATs of length less than or equal to j; i.e., $\Xi^j =_{\mathbf{df}} \Xi_1 \cup \Xi_2 \cup \ldots \cup \Xi_j$. Therefore, $\Xi^1 =_{\mathbf{df}} \Xi_1$ and $\Xi^1 \subset \Xi^2 \subset \ldots \subset \Xi^j$. Because, $\Xi^i \subset \Xi^j : i < j$, Ξ^i is said to be the *subordinate* of Ξ^j; for Ξ^i and Ξ^{i+1}, Ξ^i is said to be the *immediate subordinate* of Ξ^{i+1}.

Definition 4-6-7 : System of FLOATs

A *system* of FLOATs Ξ^n for a validation model defined by the validation model α-relation structure Ξ having maximal cardinality n is the set $\Xi^n =_{\mathbf{df}} \Xi_1 \cup \Xi_2 \cup \ldots \cup \Xi_n$.

Thus, the validation model α-relation structure Ξ and the system of FLOATs Ξ^n are equivalent.

Chapter 5 : Syntax and Semantics of Propositional Linear Temporal Logic

Section 5-1 : Introduction

The previous chapter defined the α-relation structure, the foundation from which PLTL is constructed. PLTL is now defined.

The PLTL *syntax* has a *pre-fix notation.* PLTL is a quantifier-free *modal logic* based on *first order classical propositional logic* [HC]; i.e., PLTL does not include the *existential quantifier* ∃, nor the *universal quantifier* ∀. Propositional logic has sufficient expressive power to construct correctness criteria formulae for a single automic tableaux. Because PLTL employs the temporal operators X and U, it has sufficient expressive power to construct correctness criteria formulae for the entire α-relation structure.

The *temporal semantics* of PLTL provide a means to evaluate the formulae with respect to a semantic tableaux of the α-relation structure. Exploiting the connection between subformulae and the semantics, the *semantic tableaux procedure* is a proof, reductio ad absurdum, that the conjugate of an assertion is valid with respect to the relation structure. If the conjugate of the assertion results in a contradiction in all attempts at validation, then the assertion is valid. It is shown that a contradiction occurs if, and only if, the assertion is valid.

Section 5-2 : The PLTL Syntax.

This section describes how the PLTL *language* is constructed. It consists of PLTL *formulae* selected according to the PLTL *syntax.*

Definition 5-2-1 : PLTL Alphabet

The PLTL *alphabet* L consists of the following *constituents* :

i) the *logical constants* **T** =**df** *truth* and $\perp$ =**df** *falsum*;

ii) the elements of the finite set of atomic propositions $\boldsymbol{P}$ =**df** $\{ p_1, p_2, \ldots, p_n \}$;

iii) the *logical connectives* $\neg$ =**df** *not* and $\wedge$ =**df** *conjunction*;

iv) the *temporal operators* X =**df** *next-time* and U =**df** *until.*

First order classical propositional logic is constructed from i) through iii) of Definition 5-2-1. By adding iv), derived modal logic operators interpreted in terms of time, the PLTL language is defined. The next-time operator X is used to express the fact that, at the next time instant relative to the present time instant, some formulae is true. Formulae of the form $U\psi\chi$ are read as "until ψ is true, χ is true"; that is, ψ is true at some time instant in the future, but in the mean time χ is true. The alphabet

$$L = \textbf{df} \{ T, \perp, p_1, p_2, \ldots, p_n, \neg, X, \wedge, U \}$$

has an *alphabetical ordering*, a total ordering, determined by the order of appearance of the constituents in the definition of the set L.

The PLTL syntax defines the criteria for selection of PLTL formulae Γ_{PLTL} from L^*. For the remainder of this discussion, PLTL formulae are denoted with lower-case Greek letters, e.g. ϕ, ψ, and χ.

Definition 5-2-2 : Alphabet Stratification

The PLTL alphabet L is a *stratified alphabet* [Bra], a structure < L, b >, where $b : L \rightarrow \{ 0, 1, 2 \}$ such that :

i) $b(\mathbf{T}) = 0$, $b(\perp) = 0$, $b(p_1) = 0, \ldots, b(p_n) = 0$;

ii) $b(\neg) = 1$, $b(X) = 1$;

iii) $b(\wedge) = 2$, $b(U) = 2$.

The notation $L_n = b^{-1}(n)$ identifies the set of constituents having stratification n. A formula is constructed by taking a constituent and concatenating a number of previously constructed formulae to the right of the constituent; the constituent stratification determines how many formulae may appear after the constituent.

Definition 5-2-3 : PLTL Syntax Schemata

The PLTL *syntax schemata* [Smu][Bra] consists of the following syntax schemata with stratification 0, 1, or 2 :

Stratification 0 : schemata are $\mathbf{T}, p_1, \ldots, p_n$, and $\perp$;

Stratification 1 : schemata are $\neg\psi$ and $X\psi$;

Stratification 2 : schemata are $\wedge\psi\chi$ and $U\psi\chi$.

The essence of Definition 5-2-3 is that certain strings of constituents from the alphabet L are PLTL formulae. The following definition specifies how the strings of constituents, PLTL formulae, are constructed.

Definition 5-2-4 : The PLTL Formulae

The set of *PLTL formulae* is the smallest set, denoted Γ_{PLTL}, defined inductively :

$\underline{n = 1}$: $\mathbf{T} \in \Gamma_{PLTL}$, $\perp \in \Gamma_{PLTL}$, and $\boldsymbol{P} \subseteq \Gamma_{PLTL}$;

$\underline{n > 1}$: given that ψ and $\chi \in \Gamma_{PLTL}$ at the (n - 1)-th stage of the induction, then

$$\neg\psi \in \Gamma_{PLTL},\ X\psi \in \Gamma_{PLTL},\ \wedge\psi\chi \in \Gamma_{PLTL}, \text{ and } U\psi\chi \in \Gamma_{PLTL}.$$

From now on, a PLTL formula ϕ is referred to as a formula, which implies that $\phi \in \Gamma_{PLTL}$. Γ_i denotes the set of formulae generated in the i-th stage of the induction.

For example, the strings $\neg\neg$ and $\chi X \wedge U$ are not formulae, while the strings $\wedge U \chi p \psi$ and $\neg X p$ are formulae.

The formulae are in a Polish prefix notation, where the formulae are read from left to right. Given a formula ϕ and given that a constituent is read, the stratification of the constituent indicates the number of formulae, the *subformulae* [Bri] of ϕ, that can appear to the right of the constituent.

Definition 5-2-5 : Subformulae

The *subformulae* of a formula, SUBFORM(ϕ) are defined recursively :

i) for the logical constants, SUBFORM(**T**) **=df** { **T** } and
SUBFORM($\perp$) **=df** { $\perp$ };

ii) for all atomic propositions $p \in \boldsymbol{P}$, SUBFORM(p) **=df** { p };

iii) SUBFORM($\neg\psi$) **=df** { $\neg\psi$ } $\cup$ SUBFORM(ψ);

iv) SUBFORM($X\psi$) **=df** { $X\psi$ } $\cup$ SUBFORM(ψ);

v) SUBFORM($\wedge\psi\chi$) **=df** { $\wedge\psi\chi$ } $\cup$ SUBFORM(ψ)
$\cup$ SUBFORM(χ);

vi) SUBFORM($U\psi\chi$) **=df** { $U\psi\chi$ } $\cup$ SUBFORM(ψ)
$\cup$ SUBFORM(χ).

The constituents with stratification 0 are the trivial case in the recursion. Constituents with stratification $n > 0$ are followed by n subformulae. For ϕ **=df** $\wedge\neg Upq \wedge pXq$, SUBFORM($\phi$) = { $\wedge\neg Upq \wedge pXq$, $\neg Upq$, Upq, p, q, $\wedge pXq$, p, Xq }.

In Section 2-3, a lexicographical ordering is defined for any words constructed from an alphabet; likewise, a lexicographical ordering can be defined for the infinite set Γ_{PLTL}. For any set of formulae $\Gamma_i \subseteq \Gamma_{PLTL}$ at the i-th stage of induction, vide Definition 5-2-4, a lexicographical tree can be constructed with lexicographical ordering among Γ_{PLTL} defined by a prefix, depth-first traversal of the lexicographical tree. Using a proof

by induction on the length of formulae, which is not stated here, it can be shown that a lexicographical ordering can be defined over Γ_{PLTL}. In the remaining sections, references to the lexicographical ordering over Γ_{PLTL} are made.

The number of constituents in a formula has significance.

Definition 5-2-6 : Formulae Length

The *length* of a formula ϕ, denoted LENGTH(ϕ), is the cardinality of the sequence of constituents defining the formula.

The significance of formulae length is that certain properties can be proved by induction on the length of formulae. Here, that the arbitrary property PROP holds for a given formula ϕ is denoted PROP(ϕ).

Definition 5-2-7 : Induction on the Length of Formulae

A proof by induction on the length n of every $\phi \in \Gamma_{PLTL}$ to show that PROP(ϕ) holds [Bri][Dal] is achieved iff :

n = 1 : the property is shown for formulae of length 1; i.e., PROP(**T**), PROP($\perp$), and, for every atomic formula $p \in \boldsymbol{P}$, PROP(p);

n > 1 : the induction hypothesis assumes that for every formula $\phi \in \Gamma_{PLTL}$, such that LENGTH(ϕ) < n, PROP(ϕ) holds. The property is shown for formulae of length n iff :

a) for every ψ such that LENGTH(ψ) = n - 1,
PROP(ψ) $\Rightarrow$ PROP($\neg\psi$); PROP(ψ) $\Rightarrow$ PROP(Xψ);

b) For every ψ and χ, such that LENGTH(ψ) + LENGTH(χ) = n - 1,
PROP(ψ) and PROP(χ) $\Rightarrow$ PROP($\wedge\psi\chi$) and
PROP(ψ) and PROP(χ) $\Rightarrow$ PROP(U$\psi\chi$).

In later sections, certain properties of PLTL are shown with this proof technique.

Although PLTL has sufficient power to express correctness criteria, for ease of understanding, it is sometimes useful to abbreviate the formulae using the following definition.

Definition 5-2-8 : PLTL Formulae Abbreviations

The following *formula abbreviation schemata* may be used instead of the syntax schemata :

i) *disjunction* : $\vee\psi\chi$ **=df** $\neg\wedge\neg\psi\neg\chi$;

ii) *implication* : $\supset\psi\chi$ **=df** $\neg\wedge\psi\neg\chi$;

iii) *if and only if* : $\equiv\psi\chi$ **=df** $\wedge\supset\psi\chi\supset\chi\psi$;

iv) *possible* : $F\psi$ **=df** a) $U\psi\mathbf{T}$ or b) $\neg G\neg\psi$;

v) *henceforth* : $G\psi$ **=df** a) $U\psi\perp$ or b) $\neg F\neg\psi$;

vi) *double negate* : $\neg\neg\psi$ **=df** ψ.

Abbreviated formulae, conforming to the abbreviation schemata, do not increase the expressive power of the language, but are simply a shorthand method of writing formulae. All theorems are presented and proved in terms of formulae; any abbreviated formula are converted into formulae, if necessary.

Definition 5-2-9 : Reduction Sequence

A *reduction sequence* of abbreviated formula ϕ [Dal] is a finite sequence of formulae ($\phi_1, \phi_2, \ldots, \phi_n$) where $\phi = \phi_1$, where, for every $1 < i \leq n$, ϕ_i is obtained from ϕ_{i-1} by replacing the abbreviations i) through v) of Definition 5-2-8 with the appropriate definition, and where ϕ_n is a formulae.

For example, the following is a reduction sequence for the formulae $\phi = \supset Fa \vee bc$; a, b, and c are atomic propositions.

ϕ_1 : $\supset Fa \vee bc$

ϕ_2 : $\neg \wedge Fa \neg \vee bc$; by Definition 5-2-8 ii)

ϕ_3 : $\neg \wedge Fa \neg \neg \wedge \neg b \neg c$; by Definition 5-2-8 i)

ϕ_4 : $\neg \wedge Ua\mathbf{T} \neg \neg \wedge \neg b \neg c$; by Definition 5-2-8 iv) a)

Definition 5-2-9 provides a means of converting any abbreviated formula into a formula.

For further clarity when reading formulae, other conventions are adopted. Blanks are used to separate components of the formulae, particularly the next formulae. A conjunction of a finite set of formulae $\{ \phi_1, \phi_2, \ldots, \phi_n \}$, $\wedge \phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_{n-1} \phi_n$, is denoted ${}_1\wedge_{n-1} \phi_1 \phi_2 \ldots \phi_n$ and a disjunction of a finite set of formulae $\{ \phi_1, \phi_2, \ldots, \phi_n \}$, $\vee \phi_1 \vee \phi_2 \vee \ldots \vee \phi_{n-1} \phi_n$, is denoted ${}_1\vee_{n-1} \phi_1 \phi_2 \ldots \phi_n$.

Section 5-3 : Temporal Semantics.

Temporal semantics are rules that determine if formulae state something "true" or something "false" with respect to a system Ξ^n of FLOATs. The definition of a FLOAT must be extended. Again, the set of atomic propositions $\boldsymbol{P}$ is assumed.

Definition 5-3-1 : Semantic Tableau

A *semantic tableau*, constructed from a possibly empty atomic tableau a, is the tuple s =**df** < LC(s), RC(s) >, where :

i) the *left column* of s, denoted LC(s) =**df** a $\in$ $2^{\boldsymbol{P}}$, contains the subset of the atomic propositions specified by the atomic tableau a and contains any "true" formulae, including **T**;

ii) the *right column* of s, denoted RC(s) =**df** â =**df** { $\boldsymbol{P}$ - LC(s) }, contains those atomic propositions not specified by the atomic tableau a and contains any "false" formulae, including $\perp$;

iii) an *atomic* semantic tableaux contains only the truth constants and atomic propositions;

iv) an *empty* semantic tableau, denoted $s_\circ$, contains no atomic propositions nor any "true" or "false" formulae except the logical constants; i.e., LC($s_\circ$) =**df** { **T** } and RC($s_\circ$) =**df** { $\perp$ }.

A non-empty semantic tableau may include any atomic propositions of $\boldsymbol{P}$ and other formulae. A semantic tableau s *contains* a semantic tableau s' if LC(s') $\subset$ LC(s) and RC(s') $\subset$ RC(s). All non-empty semantic tableaux must contain the empty tableau $s_\circ$. Two semantic tableaux s and s' are *equal*, denoted s = s', iff LC(s) $\subset$ LC(s'), RC(s) $\subset$ RC(s'), LC(s') $\subset$ LC(s), and RC(s') $\subset$ RC(s). Equal semantic tableau have the same assignment of atomic propositions and formulae in each column; otherwise, they

are not equal, denoted $s \neq s'$. It is said that any atomic tableaux can be *converted* into an atomic semantic tableaux.

The system Ξ^n of FLOATs is converted into a system of sequences of atomic semantic tableaux.

Definition 5-3-2 : System of Atomic α-Related Semantic Tableaux

Each *atomic α-related semantic tableaux*, α-RST, is derived from a corresponding FLOAT T included in the system Ξ^n of FLOATs, $T =\mathbf{df} (a_1, a_2, \ldots , a_p) \in \Xi^n$, by converting each atomic tableau a_j into an atomic semantic tableau s_j, $1 \leq j \leq n$.[1] The set of α-RSTs $T =\mathbf{df} (s_1, s_2, \ldots , s_p) : p \leq n$ is called a *system* Ξ^n of α-RSTs.

Accordingly, sentences of the predicates $\Xi_i : 1 \leq i \leq n$ are atomic α-RSTs. Two atomic α-RSTs $T =\mathbf{df} (s_1, s_2, \ldots , s_p) : p \leq n$ and $T' =\mathbf{df} (s'_1, s'_2, \ldots , s'_p)$ of the same length are *equal*, denoted $T = T'$, iff their corresponding atomic semantic tableaux are equal; i.e. $s_j = s'_j : 1 \leq j \leq p \leq n$; otherwise, they are not equal, denoted $T \neq T'$. No two distinct α-RSTs in Ξ^n are equal, but an α-RST T of Ξ_i may be equal to a prefix of another T' of $\Xi_j : i < j$. Thus, an *atomic sentence* can be defined for any non-atomic sentence of the predicate Ξ_i.

This dissertation adopts a notation convention for specifying α-RSTs with the α-relation. The term "α-related semantic tableaux" refers to the fact that the distinct semantic tableaux comprising the α-RSTs are ordered by the same well founded α-relation α defined among the atomic tableaux. Any sentence of Ξ^n is a well founded structure that specifies a portion of an infinite sequence of semantic tableaux. The atomic α-RST $T =\mathbf{df} (s_1) \in \Xi_1$ defines an atomic semantic tableaux. The α-relation is defined among

[1] The same notation is used for FLOATs and α-RSTs, and systems of FLOATs and α-RSTs, since there is a direct correspondence; no confusion should arise. Accordingly, the notations Ξ_i and Ξ^i apply to a system of α-related semantic tableaux.

Ξ_1. When s' α s holds, s' refers to the trailing term of a predicate sentence, s refers to the leading term of a predicate sentence, and there exists at least one atomic α-RST $T =\mathbf{df} (s_1, \ldots , s_p) \in \Xi^n : 2 \leq p \leq n$, where $s_1 = s$ and $s_p = s'$. All semantic tableaux components of an α-RST sentence are indexed with natural numbers; $s_i \in S : 1 \leq i \leq p$. When s' α s holds and corresponds to an atomic α-RST $T =\mathbf{df} (s_1, s_p) \in \Xi_2$, s' is the *next state* with respect to s; these sentences are defined by the reachability graph arcs T. Clearly, since no loops are allowed in the reachability graph, $(s, s) \notin \Xi_2$. When s α s holds, it corresponds to a cyclic atomic α-RST

$$T =\mathbf{df} (s_1, s_2, \ldots , s_i, \ldots , s_n) \in \Xi_n.$$

All sentences in Ξ_n are maximal cyclic walks that can be defined from s to s.

Using this notation, the α-RSTs define the future of all semantic tableaux defined by the predicate Ξ_1. All semantic tableaux of Ξ_1 must appear in the infinite sequences. The α-relation defines the next states of all s with the predicate Ξ_2. In general, the predicate $\Xi_i : i \leq n$ defines the future sequences of finite length of all semantic tableaux. The non-cyclic sentences of Ξ_i are prefixes of the sentences in Ξ_{i+1}; since Ξ_n defines only cyclic sentences, Ξ_{n+1} is not defined. The future portions of any infinite sequence that contains a semantic tableaux s is determined by the sentences of Ξ^n.

Temporal semantics are evaluation rules to establish a *truth relation* $\models$ between a formula $\phi \in \Gamma$ and a system Ξ^n. Establishment of the truth relation implies the assignment of certain formulae to either column of all semantic tableaux. A formula ϕ is *verified* at a semantic tableau s iff $\phi \in$ LC(s). A formula ϕ is *falsified* at a semantic tableau s iff $\phi \in$ RC(s). It is absurd to have $\phi \in$ LC(s) and $\phi \in$ RC(s), $\perp \in$ LC(s), and $T \in$ RC(s); a *contradiction* is said to have occurred in these three cases. The notation $\Xi_j \models \phi$ indicates that the formula ϕ is verified at every trailing term s_t of every sentence in Ξ_j. In contradistinction, the notation $\Xi_j \not\models \phi$ indicates that the formula ϕ is falsified in

every trailing term of the sentences in Ξ_j. When either $\Xi_j \models \phi$ and $\Xi_j \not\models \phi$, $\Xi_j \not\models T$, or $\Xi_j \models \perp$ are true, a contradiction results. Any formulae that is assigned to the left column of one semantic tableaux and the right column of another semantic tableaux, without producing a contradiction, is *satisfied* by Ξ_j; the truth relation, one way or the other, can not be written for satisfied formulae. Informally, the relation $\Xi_i \models \phi : i \leq n$ indicates that the formula ϕ says something "true" about all sentences of Ξ_i. The relation $\Xi_i \not\models \phi : i \leq n$ indicates that the formula ϕ does not say something "true" in all sentences in Ξ_i. The following definition establishes the truth relation for each predicate Ξ_i, in order of predicate length.

Definition 5-3-3 : Evaluation of Formulae [SC]

The *evaluation* of ϕ in Ξ^n is achieved by evaluating the formula in terms of every predicate $\Xi_j \in \Xi^n$, in order of predicate length. The formula ϕ is evaluated in terms of each predicate Ξ_j with respect to trailing terms s_t of predicate sentences. Evaluation begins with Ξ_1 and is defined recursively. For the trivial case, the formulae **T** is, by definition, verified no matter what $\Xi_j \in \Xi^n$ is chosen; viz., $\Xi_j \models \mathbf{T}$; the formula $\perp$ is, by definition, falsified no matter what reference point is chosen; viz., $\Xi_j \not\models \mathbf{T}$. Given any other formulae ϕ :

$\Xi_j \models p_i \in \boldsymbol{P}$ iff for every $s_t \in \Xi_j$, $p_i \in LC(s_t)$;

$\Xi_j \models \neg\psi$ iff $\Xi_j \not\models \psi$;

$\Xi_j \models \wedge\psi\chi$ iff $\Xi_j \models \psi$ and $\Xi_j \models \chi$;

$\Xi_j \models X\psi$ iff i) $j < n : \Xi_{j+1} \models \psi$;

ii) $j = n : \Xi_1 \models \psi$;

$\Xi_j \models U\psi\chi$ iff $\Xi_j \models \neg\wedge\neg\psi\neg\chi$, $\Xi_j \models \neg\wedge\chi\neg XU\psi\chi$, and $\Xi_j \models U\psi\mathbf{T}$.

Thus, the leading variable of each predicate can be thought of as the present time instant to which alternative atomic semantic tableaux are associated. If the relation is established for Ξ^n then the formula is verified at any semantic tableaux in an infinite sequence.

Part of this dissertation's thesis is that a means can be defined to show that correctness criteria hold for a validation model. In other words, that a means to establish the truth relation between the system of atomic α-RSTs Ξ^n corresponding to a validation model and the correctness criteria, a subset $\Gamma \subseteq \Gamma_{PLTL}$, can be defined. The following definitions make this notion explicit.

Definition 5-3-4 : Validity

A formula ϕ is *valid* in a system of atomic α-RSTs Ξ^n iff $\Xi^n \models \phi$.

It is said that Ξ^n *validates* ϕ if $\Xi^n \models \phi$. Clearly, $\Xi^n \models \phi$ iff $\Xi^n \not\models \neg\phi$.

Definition 5-3-5 : Tautology

A valid formula ϕ is called a *tautology*.

For example, **T** is a tautology; in the next chapters, a syntactic means of defining tautologies is presented. Only formulae $\phi \in \Gamma$ have the truth relation defined; either ϕ or its conjugate $\neg\phi$ are validated.

Definition 5-3-6 : Model

Analysis of validation models for correctness criteria is summarized by the following definitions.

i) If $\Xi^n \models \phi$, then Ξ^n is called a *model* for ϕ;

ii) If $\Xi^n \models \neg\phi$, then Ξ^n is called a *counter-model* for ϕ;

iii) A system of α-RSTs Ξ^n is a *model* for a set $\Gamma \subseteq \Gamma_{PLTL}$ iff $\forall\phi \in \Gamma$, $\Xi^n \models \phi$; this fact denoted $\Xi^n \models \Gamma$.

Some formulae are valid no matter what system of alternative α-RSTs is associated with the formulae; they are called *axioms*.

Section 5-4 : Semantic Tableaux Procedure.

This section continues semantic analysis of PLTL formulae by presenting a proof method called the *semantic tableaux procedure* [HC][Kri59][Kri63][Smu][RU]. The semantic tableaux procedure exploits a connection between PLTL syntax and the evaluation rules in Definition 5-3-3 to answer the question : is the system of α-RSTs Ξ^n a model for the formulae comprising the correctness criteria Γ? The semantic tableaux procedure answers this question by considering each formula $\phi \in \Gamma$ separately.

Essentially, the semantic tableaux procedure is a proof, reductio ad absurdum, that Ξ^n is a model for a formula ϕ. The proof begins with the assumption that Ξ^n is a model for $\neg\phi$, the conjugate of ϕ. The procedure attempts to show that a *contradiction* results from all attempts to validate $\neg\phi$, with the consequence of contradiction that formula ϕ is valid. If a contradiction is found in only some validation attempts, then satisfiability is shown. The semantic tableaux procedure investigates the validity of $\neg\phi$, with respect to every sentence of Ξ^n, by applying a succession of *semantic tableaux rules* to every predicate $\Xi_i : 1 \leq i \leq n$. The rules place formulae into semantic tableaux of sentences, based on the syntactic consequences of formulae in a semantic tableaux column. The semantic tableaux rules control the order in which predicates and sentences are investigated; in this manner, an exhaustive evaluation of the conjugate is performed for a sentence.

The semantic tableaux procedure creates a *system of semantic tableaux constructions* from a system of atomic α-RSTs Ξ^n. A *semantic tableaux construction* is an attempt to validate in all possible ways the reductio ad absurdum assumption, the conjugate of ϕ, in each predicate sentence. The *initial stage* of each semantic tableaux procedure posits the assumption, by placing ϕ in the right column of every $s \in \Xi_1$; the

result is a set of sentences $T_1 =$**df** $(s_1) \in \Xi_1$, with the reductio ad absurdum assumption $\phi \in RC(s_1)$. A semantic tableaux rule is applied to the predicate Ξ_1 and affects all sentences. In general, a *stage* of the semantic tableaux procedure is an application of a semantic tableaux rule to a predicate. The rule is applied at stage m to every alternative sentence of the predicate $\Xi_i : 1 \leq i \leq n$, but may not affect all alternative sentences. The rules are defined in terms of a formula ϕ and the trailing tableaux s_t of the sentence. Details are defined in terms of a semantic tableaux construction for one sentence, and is easily extended to the general case.

Definition 5-4-1 : Semantic Tableaux Procedure

A stage of the semantic tableaux procedure is defined by induction. A stage m consists of an alternative set of semantic tableaux constructions each associated with sentences of Ξ^m.

m = 1: Each sentence T_m of Ξ_1 begins a semantic tableaux construction. The sentence $T_m =$**df** $(s) \in \Xi_1$, with ϕ placed in the right column of semantic tableau s; i.e., $\forall(s) \in \Xi_1, \phi \in RC((s))$.

m > 1 : The (m - 1)-th stage of the procedure is an alternative set of semantic tableaux constructions. Each semantic tableaux construction is a set of alternative validation attempts T_{m-1}, sentences in $\Xi_i : 1 \leq i \leq n$. The validation attempt T_{m-1} has a leading tableau s_l and trailing tableau s_t. A validation attempt T_m of a predicate $\Xi_j : j \geq i$ results from applying one of the following *Semantic Tableaux Rules* [Kri63] to a formula in either column of trailing tableau s_t.

i) *Negation Left*, NL : If $\neg\psi \in LC(s_t)$ in T_m, then T_{m+1} is T_m with ψ placed in $RC(s_t)$.

ii) *Negation Right*, NR : If $\neg\psi \in RC(s_t)$ in T_m, then T_{m+1} is T_m with ψ placed in $LC(s_t)$.

iii) *Conjunction Left*, $\wedge$L : If $\wedge\psi\chi \in LC(s_t)$ in T_m, then T_{m+1} is T_m with ψ and χ placed in $LC(s_t)$.

iv) *Conjunction Right*, $\wedge$R : If $\wedge\psi\chi \in RC(s_t)$ in T_m, then the sentence T_m is replicated creating two new possible sentences T'_{m+1} and T''_{m+1}; T'_{m+1} is T_m with ψ placed in $RC(s_t)$ and T''_{m+1} is T_m with χ placed in $RC(s_t)$. The result is that $\psi \in RC(s'_t)$ of T'_{m+1} and $\chi \in RC(s''_t)$ of T''_{m+1}.

v) *Next-time Left*, XL : If $X\psi \in LC(s_t)$ in $T_m \in \Xi_i : i < n$, then ψ is placed in $LC(s_t)$ and, for every sentence $T_{m+1} \in \Xi_{i+1}$ with trailing tableau s_{t+1} such that the sentence T_m is a prefix of T_{m+1}, the prefix $T_{m+1}[i]$ is T_m and ψ is placed in $LC(s_{t+1})$. If $X\psi \in LC(s_t)$ in $T_m \in \Xi_i : i = n$, then T_{m+1} is T_m with ψ placed in $LC(s_t)$.

vi) *Next-time Right*, XR : If $X\psi \in RC(s_t)$ in $T_m \in \Xi_i : i < n$, then ψ is placed in $RC(s_t)$ and, for every sentence $T_{m+1} \in \Xi_{i+1}$ with trailing tableau s_{t+1} such that the sentence T_m is a prefix of T_{m+1}, the prefix $T_{m+1}[i]$ is T_m and ψ is placed in $RC(s_{t+1})$. If $X\psi \in RC(s_t)$ in $T_m \in \Xi_i : i = n$, then T_{m+1} is T_m with ψ placed in $RC(s_t)$.

vii) *Until Left*, UL : If $U\psi\chi \in LC(s_t)$ in T_m, then T_{m+1} is T_m with $\neg\wedge\neg\psi\neg\chi$, $\neg\wedge\chi\neg XU\psi\chi$, and $U\psi\chi$**T** placed in $LC(s_t)$.

viii) *Until Right*, UR : If $U\psi\chi \in RC(s_t)$ in T_m, then T_{m+1} is T_m with $\neg\wedge\neg\psi\neg\chi$, $\neg\wedge\chi\neg XU\psi\chi$, and $U\psi\chi$**T** placed in $RC(s_t)$.

The new stage T_{m+1} may be equal to the previous stage, except with formulae added to columns of some semantic tableaux of T_{m+1}. The rules XL and XR force the addition of α-RSTs $T_{m+1} \in \Xi_{i+1}$ to the semantic tableaux construction. The rule $\wedge$R requires the replication of α-RST T_m and the addition of the resulting α-RSTs T'_{m+1} and T''_{m+1} to

the semantic tableaux construction. Thus, at any stage, there are a number of *alternative validation attempts*.

The semantic tableaux procedure applies semantic tableaux rules to a construction until it is impossible to apply further rules. No rule is applied if it results in duplicating a formula in a column; the application of such a rule, using Kripke's terminology, is said to be *superfluous*. The semantic tableaux procedure does not apply a rule to an alternative validation attempt once a contradiction has been found.

Definition 5-4-2 : Closure

Certain situations in the system of semantic tableaux constructions imply a contradiction :

i) a semantic tableau is *closed* either when a formula occurs in both columns of the semantic tableau, when **T** is in the right column, or when $\perp$ is in the left column;

ii) an alternative validation attempt is closed iff some semantic tableau s of that sequence is closed;

iii) a semantic tableaux construction is closed iff each alternative validation attempt is closed;

iv) a system of alternative semantic tableaux constructions is closed iff each of the alternative semantic tableaux constructions are closed.

A closed semantic tableaux construction implies that a contradiction is generated in all alternative validation attempts. A closed system of alternative semantic tableaux constructions implies that ϕ is valid. A semantic tableaux rule is not applied to a closed validation attempt, since that attempt has generated a contradiction. The procedure terminates for sentence in a predicate when its semantic tableaux construction is closed. The procedure terminates for a system of semantic tableaux constructions when each of the constructions is closed.

A semantic tableaux construction is a tree structure, where the alternative validation attempts comprise the nodes. Before the formal definition is made, the successor relation **Q** is defined. The successor relation is defined only as a result of applying a rule. There are two ways in which an alternative validation attempt T_m may be a successor of another T_{m-1} in the tree.

Definition 5-4-3 : The Successor Relation Q

There are two ways an alternative validation attempt T_m can be a successor of another non-closed alternative validation attempt T_{m-1} :

i) The alternative validation attempt T_m is the *immediate descendant in* $\Xi_i : i \leq n$ obtained from the non-closed alternative validation attempt T_{m-1} when :

   a) T_{m-1} is not affected by a semantic tableaux rule; in this case $T_m = T_{m-1}$.

   b) T_{m-1} is affected by any semantic tableaux rule, except rules XL and XR; in this case $T_m \neq T_{m-1}$. The rule $\wedge$R is a special case when there are two immediate descendants in Ξ_i, T'_m and T''_m, where

$$T'_m \neq T_{m-1} \text{ and } T''_m \neq T_{m-1}.$$

ii) An *immediate descendant of* $\Xi_i : i \leq n$ occurs when T_{m-1} is affected by the semantic tableaux rules XL and XR.

An alternative validation attempt T_m is a descendant of an alternative validation attempt $T_o : o > m + 2$.

Definition 5-4-4 : Semantic Tableaux Tree

A semantic tableaux construction defines an unordered *semantic tableaux tree* STT $=_{df} < \Xi^n, \mu, Q >$, where :

i) the each sentence T of Ξ^n is an alternative validation attempt and a node in the tree; sentence T_1 of the main stage is the root-node of the tree;

ii) the range of μ determines at which stage the alternative validation attempt is added to the construction; μ determines the index of an alternative validation attempt in a construction;

iii) for T and T' $\in$ P, T **Q** T' holds as per Definitions 5-4-3.

Since a semantic tableaux rule is not applied to a closed alternative validation attempt, a closed alternative validation attempt must be an end-node of the tree. The semantic tableaux tree is a finitely generated, finite tree; this is clear, because there are a finite number of semantic tableaux rules, alternative validation attempts are finite in length, and because of the criteria by which the semantic tableaux procedure terminates.

This section concludes with a proof that the system of semantic tableaux constructions for a formula ϕ is closed iff Ξ^n is a model for ϕ. For ϕ to be valid, all alternative semantic tableaux constructions $< \Xi^n, \mu, \mathbf{Q} >$ must be closed. The following lemma shows the connection between the truth relation $\models$ and the semantic tableaux procedure. Until a contradiction occurs in all alternative validation attempts, the conjugate is satisfied. An alternative validation that produces a contradiction is removed from consideration. Thus, the truth relation $\models$ in this lemma refers to the non-closed sentences in Ξ^n. Any non-closed sentences after the semantic tableaux procedure implies that there is a substructure of Ξ^n that validates $\neg\phi$.

Lemma 5-4-7 : Membership in a Column Preserves Formulae Evaluation.

At stage m in the semantic tableaux procedure, there is an alternative validation attempt $T_m =$**df** $(s_1, s_2, \ldots, s_i) \in \Xi_i \in \Xi^n : 1 \leq i \leq n$, where $s_t =$**df** s_i. The lemma asserts the following "condition" : For every non-closed alternative validation attempt T_m, if $\psi \in LC(s_t) : s_t \in T_m$ then $\Xi_i \models \psi$; otherwise, for every non-closed alternative validation attempt T_m, if $\psi \in RC(s_t) : s_t \in T_m$ then $\Xi_i \not\models \psi$.

Proof : By Induction.

The lemma is proved by induction on m, the number of stages in a semantic tableaux construction. Each validation attempt T_m is included in a semantic tableaux construction STT =**df** $< \Xi^n, \mu, \mathbf{Q} >$ at stage m =**df** $\mu(T_m)$. The proof must show that the successors of any alternative validation attempt T_m satisfy the condition. An important fact required by this proof is that at each stage the evaluation rules hold, Definition 5-3-3.

<u>m = 1</u> : The initial stage of the construction consists of a sentences $T_1 \in \Xi_1$ with $\phi \in RC(s_t)$. Because of the definition of the initial stage, the assertion $\Xi_1 \not\models \phi$ can be made satisfying the condition.

$\underline{m > 1}$: The induction hypothesis posits that the condition is satisfied by any non-closed validation attempt $T_{m-1} \in \Xi_i$ in the (m - 1)-th level of the semantic tableaux tree. An alternative validation attempt $T_m \in \Xi_j : j \geq i$ in the m-th level is obtained from the (m - 1)-th stage by applying a semantic tableaux rule to some formula ϕ in a column of T_{m-1}. If $T_m = T_{m-1}$, T_{m-1} is not affected by the rule, and the condition holds for T_m in the m-th stage. If $T_m \neq T_{m-1}$, the effects of applying every semantic tableaux rule must be investigated to show that the condition is satisfied in T_m.

$\underline{\neg L}$: $\neg\psi \in LC(s_t)$ in T_{m-1}. By the induction hypothesis, $\Xi_i \models \neg\psi$ and $\Xi_i \not\models \psi$. The rule $\neg L$ places ψ in $RC(s_t)$ resulting in the immediate successor T_m. Since T_m is equal to T_{m-1}, except that $\psi \in RC(s_t)$ in T_m, $\Xi_j \not\models \psi$.

$\underline{\neg R}$: $\neg\psi \in RC(s_t)$ in T_{m-1}. By the induction hypothesis, $\Xi_i \not\models \neg\psi$ and $\Xi_i \models \psi$. The rule $\neg R$ places ψ in $LC(s_t)$ resulting in the immediate successor T_m. Since T_m is equal to T_{m-1}, except that $\psi \in LC(s_t)$ in T_m, $\Xi_j \models \psi$.

$\underline{\wedge L}$: $\wedge\psi\chi \in LC(s_t)$ in T_{m-1}. The induction hypothesis implies that $\Xi_i \models \wedge\psi\chi$, $\Xi_i \models \psi$, and $\Xi_i \models \chi$. The rule $\wedge L$ places ψ and χ in $LC(s_t)$ resulting in the immediate successor T_m. Since T_m is equal to T_{m-1}, except that $\psi \in LC(s_t)$ and $\psi \in RC(s_t)$ in T_m, $\Xi_j \models \psi$ and $\Xi_j \models \chi$.

$\underline{\wedge R}$: $\wedge\psi\chi \in RC(s_t)$ in T_{m-1}. The induction hypothesis implies that either

i) $\Xi_i \not\models \wedge\psi\chi$, $\Xi_i \not\models \psi$, and $\Xi_i \models \chi$,

ii) $\Xi_i \not\models \wedge\psi\chi$, $\Xi_i \models \psi$, and $\Xi_i \not\models \chi$, or

iii) $\Xi_i \not\models \wedge\psi\chi$, $\Xi_i \not\models \psi$, and $\Xi_i \not\models \chi$.

The rule $\wedge R$ constructs two alternative validation attempts T'_m and T''_m from T_{m-1}, where T'_m is T_{m-1} with ψ in $RC(s_t)$ and T''_m is T_{m-1} with χ in $RC(s_j)$. The tableaux $s'_t \in T'_m$ and $s''_t \in T''_m$ are the same as $s_t \in T_{m-1}$, but with ψ in $RC(s'_t)$ and χ in $RC(s''_t)$.

i) $\Xi_i \not\models \wedge\psi\chi$, $\Xi_i \not\models \psi$, and $\Xi_i \models \chi$: The condition is satisfied for T'_m because $\Xi_i \not\models \psi$ implies $\Xi_j \not\models \psi$. The truth relation $\Xi_i \models \chi$ implies that $\chi \in LC(s_t)$ of T_{m-1}. T''_m produces a contradiction because $\chi \in LC(s''_t)$ and $\chi \in RC(s''_t)$ of T''_m.

ii) $\Xi_i \not\models \wedge\psi\chi$, $\Xi_i \models \psi$, and $\Xi_i \not\models \chi$: The truth relation $\Xi_i \models \psi$ implies that $\psi \in LC(s_j)$ of T_{m-1}. T'_m produces a contradiction because $\psi \in LC(s'_t)$ and $\psi \in RC(s'_t)$ of T'_m. The condition is satisfied for T''_m because $\Xi_i \not\models \chi$ implies $\Xi_j \not\models \chi$.

iii) $\Xi_i \not\models \wedge\psi\chi$, $\Xi_i \not\models \psi$, and $\Xi_i \not\models \chi$: The condition is satisfied because $\Xi_i \not\models \psi$ implies $\Xi_j \not\models \psi$ and because $\Xi_i \not\models \chi$ implies $\Xi_j \not\models \chi$.

XL : $X\psi \in LC(s_t)$ in T_{m-1}. The induction hypothesis implies that $\Xi_i \models X\psi$ and two possibilities :

i) $i < n : \Xi_j \models \psi : j =$**df** $i + 1$. The rule XL creates at least one T_m of length j with ψ in $LC(s_j)$; T_{m-1} is a prefix of T_m, a prefix of length i. The condition is satisfied for all T_m.

ii) $i = n : \Xi_1 \models \psi$: The rule XL places ψ in $LC(s_1)$ of T_{m-1}, creating T_m. The condition is satisfied.

XR : $X\psi \in RC(s_j)$ in T_{m-1}. The induction hypothesis implies that $\Xi_i \not\models X\psi$ and two possibilities :

i) $i < n : \Xi_j \not\models \psi : j =$**df** $i + 1$. The rule XL creates at least one T_m of length j with ψ in $RC(s_j)$; T_{m-1} is a prefix of T_m, a prefix of length i. The condition is satisfied for all T_m.

ii) $i = n : \Xi_1 \not\models \psi$: The rule XL places ψ in $RC(s_1)$ of T_{m-1}, creating T_m. The condition is satisfied.

UL : $U\psi\chi \in LC(s_j)$ in T_{m-1}. The induction hypothesis implies that $\Xi_i \models U\psi\chi$, which, in turn, implies that $\Xi_i \models \neg\wedge\neg\psi\neg\chi$, $\Xi_i \models \neg\wedge\chi\neg XU\psi\chi$, and $\Xi_i \models U\psi\mathbf{T}$. The rule UL creates a T_m with $\neg\wedge\neg\psi\neg\chi$, $\neg\wedge\chi\neg XU\psi\chi$, and $U\psi\mathbf{T}$ placed in $LC(s_t)$. Clearly, $\Xi_j \models \neg\wedge\neg\psi\neg\chi$, $\Xi_j \models \neg\wedge\chi\neg XU\psi\chi$, and $\Xi_j \models U\psi\mathbf{T}$. The condition is satisfied.

UR : $U\psi\chi \in RC(s_j)$ in T_{m-1}. The induction hypothesis implies that $\Xi_i \not\models U\psi\chi$, which, in turn, implies that $\Xi_i \not\models \neg\wedge\neg\psi\neg\chi$, $\Xi_i \not\models \neg\wedge\chi\neg XU\psi\chi$, and $\Xi_i \not\models U\psi\mathbf{T}$. The rule UR creates a T_m with $\neg\wedge\neg\psi\neg\chi$, $\neg\wedge\chi\neg XU\psi\chi$, and $U\psi\mathbf{T}$ placed in $RC(s_t)$. Clearly, $\Xi_j \not\models \neg\wedge\neg\psi\neg\chi$, $\Xi_j \not\models \neg\wedge\chi\neg XU\psi\chi$, and $\Xi_j \not\models U\psi\mathbf{T}$. The condition is satisfied.

Q.E.D.

Lemma 5-4-8 : A Closed System of Semantic Tableaux Constructions for ϕ Implies that Ξ^n is a Model for ϕ.

If all semantic tableaux constructions are closed for a formulae ϕ in a system of α-RSTs Ξ^n, then Ξ^n is a model for ϕ.

Proof : Reductio ad Absurdum.

The semantic tableaux procedure is a proof, reductio ad absurdum, that the conjugate of ϕ is valid. Each semantic tableaux construction begins with the assumption $\phi \in RC(s_1)$ of the initial stage T_1; i.e., $\Xi_1 \not\models \phi$. The semantic tableaux procedure attempts to show that $\Xi^n \not\models \phi$. An immediate descendant is constructed from a non-closed alternative validation attempt. Lemma 4-5-7 ensures that membership of ϕ within a column of a non-closed maintains the correct evaluation of ϕ in terms of Ξ^n. Because every semantic tableaux construction is closed, every end-node of every semantic tableaux tree contains a closed semantic tableau; hence, an evaluation contradiction has been found in all possible attempts to validate $\neg\phi$.[1] Since every attempt at validation of the conjugate results in an absurdity, reductio ad absurdum, ϕ must be valid. **Q.E.D.**

[1] The conjugate $\neg\phi$ is not satisfied by Ξ^n.

Lemma 5-4-9 : A Non-closed System of Semantic Tableaux Constructions for ϕ Implies that Ξ^n is Not a Model for ϕ.

If the system of semantic tableaux constructions Ξ^n are not closed for a formula ϕ when the semantic tableaux procedure termintes, then Ξ^n is a counter-model for ϕ.

Proof : By Inspection of Each Semantic Tableaux Tree.

It must be shown that $\neg\psi$ is satisfied by Ξ^n, implying that Ξ^n is a counter-model for ϕ, and ϕ is not valid. If the system of semantic tableaux constructions Ξ^n for ϕ is not closed after the semantic tableaux procedure terminates, then at every stage of some construction there exists at least one non-closed alternative validation attempt $T =$**df** $(s_1, s_2, \ldots , s_n)$. T is a non-closed end-node of the finite semantic tableaux tree. By Lemma 4-5-7, if $\phi \in RC(s_j) : 1 \leq j \leq n$, then $\Xi^n \not\models \phi$. That Ξ^n is a counter-model for ϕ is determined by identifying a semantic tableau $s \in T$ with $\phi \in RC(s_j)$; by definition of the semantic tableaux procedure, semantic tableau $s_1 \in T$ is such a tableau; therefore, $\Xi_1 \not\models \phi$. Thus, Ξ^n is a counter-model for ϕ. **Q.E.D.**

The following theorem is fundamental to the semantic tableaux procedure.

Theorem 5-4-10 : A formulae ϕ is valid iff the semantic tableaux construction for ϕ is closed.

Proof : a iff b $\Leftrightarrow$ if a then b and if not b then not a.

The proof of this theorem follows immediately from lemmata 5-4-8 and 5-4-9.

Chapter 6 : THE PLTL FORMAL DEDUCTIVE SYSTEM.

Section 6-1 : Introduction.

The purpose of this chapter is to define a mechanism to generate formulae valid when interpreted with respect to validation model, a deductive means to show that correctness criteria formulae are tautologies with respect to validation models. The advantage of such a mechanism is that the iterative evaluation process [McA] and the semantic tableaux procedure are not explicitly invoked, but are implicit in the mechanism. The mechanism starts with a set of tautologies that express a system of atomic α-RSTs Ξ and selects additional tautologies using syntax definitions.

Definition 6-1-1 : Formal Deductive System [Cop]

The *Formal Deductive System* is a collection of :

i) *terms*, arbitrary symbols interpreted as objects of discourse,

ii) *theses* and *hypotheses*, constructed using terms, and

iii) *theorems*, derived from ii).

A PLTL formal deductive system is defined here, based on the concepts of model theory [Bri][Sac], semantic tableaux [Kri63][RU], *modal normal form formulae* [Fine], and *Henkin proofs* [HC]. For the following definitions, ϕ, ψ, and χ are formulae.

The PLTL deductive system is a syntactic technique of selecting tautologies called *theorems* using a *derivation*. That formulae are valid is expressed in terms of the system of semantic tableaux constructions.

Definition 6-1-2 : Left and Right Side

The *left side*, LS, of the system of semantic tableaux constructions is all left columns in the system and the *right side*, RS, is all right columns in the system.

Evaluation of the derived formulae assigns theorems to the left side and the conjugates of the theorems to the right side. The deductive system must ensure that if theorem ψ is assigned to the left side then its conjugate $\neg\psi$ is assigned to the right side, and vice versa; to have a formula on the left and right side implies that the deductive system derives a contradiction. Neither non-theorems nor conjugations of non-theorems can be assigned strictly to either side of the system of alternative semantic tableaux constructions, because evaluations of non-theorems may place a formulae in some of the left side and in some of the right side. However, non-theorems ψ and $\neg\psi$ must not generate an evaluation contradiction.

The items in Definition 6-1-1 are formulae in the PLTL deductive system. The semantic tableaux are terms of the deductive system.[1] The *axiom schemata types* [HC][McA][MP83], that imply the linear, discrete, and absolute view of time, define the syntax of valid formulae called theses. The hypotheses are a set of valid formulae that pose the system Ξ^n of RSTs. From this *axiomatic basis*, theorems are derived. The PLTL formal deduction system must be *sound*, a theorem ϕ is valid in a model, and be *sufficient*, any formulae ϕ valid in a model is a theorem.

[1] An atomic proposition assigned to a left side is a valid formulae and implies that the atomic proposition is a member of all atomic tableaux.

Section 6-2 : Terms.

The terms of the PLTL deductive system are the observed validation model phenomena, viz. the semantic tableaux in the system of atomic α-RSTs Ξ^n. As usual, it is assumed that atomic tableaux are constructed from a subset of the finite set $\boldsymbol{P} =_{\mathbf{df}} \{ p_1, p_2, \ldots, p_p \}$ of atomic propositions. The definition includes only those atomic tableaux that are found in a system of semantic tableaux constructions, but allows for any possible atomic tableaux that can be constructed from $\boldsymbol{P}$.

Definition 6-2-1 : Semantic Tableaux Formulae

The set of all possible *semantic tableaux formulae* $\underline{s}_k : 1 \leq k \leq |S| \leq 2^{|P|}$ defined for every semantic tableau $s \in \Xi_1 \subset \Xi^n$ is :

$$\wp =_{\mathbf{df}} \{ \underline{s}_k \ni \underline{s}_k =_{\mathbf{df}} {}_{1\wedge p-1} \pi_1 p_1 \pi_2 p_2 \ldots \pi_p p_p : 1 \leq i \leq p,$$

$$\pi_i \text{ is blank iff } p_i \in LC(s) \text{ and } \pi_i \text{ is a } \neg \text{ iff } p_i \in RC(s) \}.$$

A term $\underline{s}$ implies a semantic tableaux s, and vice versa; thus, the well founded α-relation can be specified among the terms. The semantic tableaux formulae describe which atomic propositions are included in the left and right columns of a semantic tableaux $s \in \Xi_1$. Using this definition, a semantic tableau formula $\underline{s}_k$ is said to contain a semantic tableau formula $\underline{s}_j$, denoted $\underline{s}_j \subset \underline{s}_k$, iff every π_i that is blank in $\underline{s}_j$ is blank in $\underline{s}_k$; the definition of equality among semantic tableaux formulae follows similarily.

Atomic tableaux formulae are not necessarily theorems. If for every $\underline{s}_k \in \wp$ there is a $p_i : 1 \leq i \leq p$, where π_i is blank, then that atomic proposition is true in every semantic tableaux; such an atomic proposition is called an *atomic theorem.* The atomic theorems are useful in what follows.

Section 6-3 : Theses.

This section formally defines and informally discusses the theses of the PLTL deductive system. Since theses form part of the axiomatic basis, the generic term *axiom* is used in the following definitions.

The formulae **T** and $\perp$ are special cases in the syntax and semantics of PLTL; because **T** is always verified and $\perp$ is always falsified, the PLTL deductive system automatically assign them to the left side and right side, respectively.

Definition 6-3-1 : Axioms of Tautology, Falsification, and No Contradiction

i) The formula **T** is called the *axiom of tautology*.

ii) The formula $\neg\perp$ is called the *axiom of falsification*.

iii) The formula $\equiv\mathbf{T}\neg\perp$ is called the *axiom of no contradiction*.

These axioms are useful in the proofs that follow.

Theses corresponding to *propositional axiom schemata*, based on conjunction and negation found in the references [Dal][HC], are included in the deductive system. From the propositional axiom schemata, theses, such as De Morgan's laws, the Law of Double Negation, the Commutative Laws, the Associative Laws, the Distributive Laws, and the Law of Transposition, are derived. The cited references list additional theses.

Some theses contain the temporal operators X and U and formulae abbreviations F and G.

Definition 6-3-2 : PLTL Temporal Axiom Schemata

The *PLTL temporal axiom schemata* [MP83] consist of the following axiom schemata types.

A1. $\vdash \equiv\neg F\psi G\neg\psi$.

A2. $\vdash \supset G\psi\psi$

A3. $\vdash \supset G\psi X\psi$

A4. $\vdash \supset G\psi XG\psi$

A5. $\vdash \supset G\supset\psi\chi\supset G\psi G\chi$

A6. $\vdash \supset X\supset\psi\chi\supset X\psi X\chi$

A7. $\vdash \equiv X\neg\psi\neg X\psi$

A8. $\vdash \supset G\supset\psi X\psi\supset\psi G\psi$

A9. $\vdash \equiv U\psi\chi\vee\psi\wedge\chi XU\psi\chi$

A10. $\vdash \supset U\psi\chi F\psi$

A11. $\vdash \supset\psi X\mathbf{T}$

The temporal axioms are better understood by the following informal discussion.

The meaning of axiom A1 is that a formula ψ is false at all semantic tableaux iff it is not the case that there is a future semantic tableaux where ψ is true. Axiom A1 establishes that the temporal operators F and G are *duals*; that is, they can be defined in terms of each other with the definitions $G\psi =\mathbf{df} \neg F\neg\psi$ and $F\psi =\mathbf{df} \neg G\neg\psi$. The next four axioms define the implications of the G operator. Axiom A2 states that a consequence of $G\psi$ being valid is that ψ must be valid. An implication of Axiom A2 is that the present is part of the future; clearly, the next time instant is a future time instant. Axiom A3 states that $G\psi$ implies that $X\psi$ is true in all future semantic tableaux. Axiom A4 states that when henceforth ψ is true, at the next semantic tableaux ψ must be

henceforth true. Axiom A5 establishes that if a *material implication* is henceforth true then the *antecedant* and the *consequent* [HC] of the implication must also be henceforth true.

The X operator is a fundamental temporal operator of PLTL, since it is used to create formulae that express the linear structures contained in the validation model α-relation structure. Axiom A6 is similar to axiom A5, since if at the next semantic tableaux a material implication is true then the antecedant and the consequence of the implication must also be true at the next semantic tableaux. Axiom A7 is important because it restricts the language PLTL to linear structures; Figure 6-1 illustrates why. In that figure, the dots represent time instants and the arrows represent the relation of temporal precedence among the time instants; Figure 6-1(a) illustrates a *branching structure* [MP83][RU] Figure 6-1(b) illustrates a linear structure.

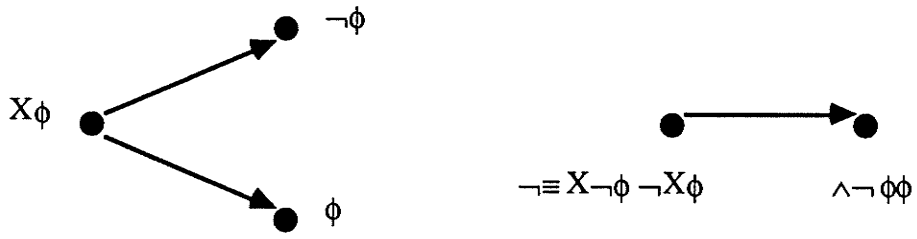


Figure 6-1 : Branching and Linear Structures that do not Verify Axiom A7

Clearly, the formula ≡X¬ϕ¬Xϕ is not an axiom for a branching structure. In a linear structure the axiom can only be false if there is a contradiction, when both ϕ and ¬ϕ are verified at the same time instant.

Axiom A8 is called the *computational induction axiom schema* [MP83]. It states that if henceforth a property established at one semantic tableau implies the establishment of the property at the next semantic tableau, then the establishment of that property implies

that it is henceforth true. Axioms A9 and A10 pertain to the U operator. Axiom A9 distributes the effect of the U operator to the present semantic tableau and the next semantic tableau. Formulae of the form $U\psi\chi$ are true in two situations. Either ψ is true, or χ is true and in the next semantic tableau $U\psi\chi$ is true. Also, both ψ and χ can be simultaneously true. The semantic tableaux rules UL and UR are defined from Axiom A9. Axiom A10 states that for $U\psi\chi$ to be true, it must be the case that some semantic tableau in the future ψ must be true; thus $U\psi\chi$ cannot be true if only χ is established.

Axiom schemata A11 is called the *next-time consistency axiom schema.* Because the validation models can be cyclic, there is always a next state. Thus, any valid formula is true in the next semantic tableaux. Since **T** is always verified, any formulae of Axiom schemata A11 type can be added to the deductive system without causing a contradiction at the next semantic tableaux. For example, if χ is valid then $\supset\psi X\mathbf{T}$ is an axiom that does not contradict χ at the next time instant. This axiom is used to define the hypotheses of the deductive system.

Section 6-4 : Hypotheses.

The hypotheses of the PLTL formal deductive system are tautologies that describe the system of atomic α-RSTs Ξ^n. Hence, the set of hypotheses indirectly describe the validation model and provide part of the axiomatic basis for deriving theorems of the validation model.

Hypotheses have a special ability to express a system of α-RSTs Ξ^n; first, hypotheses must be constructed to express $\Xi_i : 1 \leq i \leq n$. Each α-RST $T =\mathbf{df} (s_1, s_2, \ldots , s_i) \in \Xi_i$ begins with a particular leading tableau s_1 and ends with the trailing tableaux s_i. The terms $\wp$ express the semantic tableaux in the α-RST T. A hypothesis ϕ must be able to specify the leading term ϕ_l, a trailing term ϕ_t, all terms between the leading and trailing terms, and the well founded α-relation ordering the terms. For a member of the well founded α-relation a α a' in the system Ξ^n, there is a NTNF formula ϕ. The hypotheses definition is inductive; it begins with the set of hypotheses X_1 that express Ξ_1. The induction hypothesis assumes a set of hypotheses X_{i-1} that express Ξ_{i-1}, and creates a set of hypotheses X_i that express Ξ_i; in this manner, a set of hypotheses X_n can be constructed that express Ξ_n. The following hypotheses use the X operator to describe the α-relations among the terms to define hypotheses that express the system of semantic tableaux constructions Ξ^n. The following definition uses $\psi_{i(l)}$ to denote the leading term of ψ_i.

Definition 6-4-1 : Next-Time Normal Form Formulae [Fine]

The set X_n of *next-time normal form formulae*, NTNF formulae, of degree n corresponding to Ξ_n is defined by induction on m :

<u>m = 1</u> : The set $X_1 =\mathbf{df} \{ \phi \ni \phi =\mathbf{df} \supset \psi \mathbf{X} \mathbf{T} : \psi \in \wp \}$.

$\underline{m > 1}$: The induction hypothesis posits that X_{m-1} **=df** { $\psi_1, \psi_2, \ldots, \psi_k$ } consists of the next-time normal form formulae of degree m - 1 arranged in their lexicographical order.

X_m **=df** { ϕ ∋

i) ϕ **=df** $\supset$ B $X_1 \vee_{k-1} \pi_1\psi_1 \pi_2\psi_2 \ldots \pi_k\psi_k$;

ii) B **=df** $\phi_1 : B \in \wp$;

iii) π_i is blank iff $\phi_1 \alpha \psi_{i(l)}$[2] holds or π_i is $\neg$ iff $\phi_1 \alpha \psi_{i(l)}$ does not hold : $1 \leq i \leq k$ }.

The next-time normal forms X_n can describe any atomic α-RSTs in Ξ_n. For X_i and $X_j : i > j$, X_j is said to be the *subordinate* of X_i; X_1 does not have a subordinate. The set $X_{i-1} : i > 1$, is called the *immediate subordinate* of X_i. Each element of the set X_i is constructed from X_1, X_2, and X_{i-1}. The disjunction describes the remaining portions of all the α-RSTs that begin with term B and have n - 1 terms. The term B temporally precedes the leading terms of the disjunction having a blank π_i, but does not temporally precede the leading terms of the disjunction with a $\neg$ in π_i. If there exists a NTNF formula ϕ having $\neg$ in every π_i of every disjunct D_i, the tableau B does not temporally precede any other tableau; this does not happen here. The *trailing term* of the NTNF formula ϕ, denoted $\phi_t \in \wp$, is the semantic tableaux formula ψ of the subformula $\supset\psi XT \in X_1$ of ϕ.

The NTNF formulae X^n **=df** $X_1 \cup X_2 \cup \ldots \cup X_n$ are organized to express Ξ^n. The α-relation must be specified among X^n.

---

[2] i.e., $\phi_1 \alpha \psi_{i(l)}$ is the α-relation among the terms ϕ_1 and $\psi_{i(l)}$ which corresponds to the 2-ary predicate $A_2(\phi_1, \psi_{i(l)})$. ϕ_1 corresponds to the next state with respect to the state corresponding to $\psi_{i(l)}$.

Definition 6-4-2 : α-Relation Among NTNF Formulae

The *α-relation among NTNF formulae* $\psi \in X_i$ and $\chi \in X_j$, $i \neq j$, denoted $\psi \Re \chi$, holds iff $\psi_1 =$**df** s' and $\chi_1 =$**df** s, such that s' α s holds.

The α-relation is specified among the leading tableaux of the NTNF formulae.

The following definition uses Ξ^n to denote a structure of NTNF formulae to express the system of α-RSTs Ξ^n.

Definition 6-4-3 : PLTL Deductive System Model

For $n \geq 1$, the *PLTL deductive system model* is the triple $\Xi^n =$**df** $< W, \Re, H >$ such that :

i) $W =$**df** X^n;

ii) $\Re =$**df** $\{ < \psi, \chi > \ni < \psi, \chi > \in W \times W : \psi \Re \chi \}$;

iii) $H =$**df** $\{ < \psi, a_i > \ni < \psi, a_i > \in W \times 2^P : \underline{a_i} =$**df** $\psi_1 \}$[3].

The set W in Ξ^n contains all NTNF formulae of degree not greater than n. The set $\Re$ defines the well founded α-relation of temporal precedence among the leading terms of NTNF formulae. The mapping H associates NTNF formulae with distinct semantic tableaux in the system of α-RSTs. If evaluated, each element of W in Ξ^n would be placed in the left column of some initial tableau of a α-RST. The definition of subordinates also extends to PLTL models. For Ξ^i and $\Xi^j : i > j$, Ξ^j is the *subordinate* of Ξ^i and Ξ^{i-1} is the *immediate subordinate* of Ξ^i.

Since the PLTL model Ξ^n is a collection of hypotheses to express the relation of temporal precedence among the semantic tableaux comprising a system of atomic α-RSTs, the validity of formulae can be denoted with respect to Ξ^n or any of its subordinates. In fact, definitions of evaluation, Definition 5-3-4, or of the semantic tableaux procedure can be made in terms of a PLTL model Ξ^n, because the ordering of semantic tableaux is

[3] i.e., a_i is a distinct semantic tableau corresponding to the leading term of ψ.

implicit in the ordering among NTNF fromulae; this is implicit in what follows. The notation $\Xi^n \models \phi$ indicates that ϕ is valid with respect to the PLTL model. For an atomic theorem ϕ, $\Xi^1 \models \phi$ and $\Xi^n \models \phi$ hold. For any formula $\phi \in W$ of Ξ^n, $\Xi^n \models \phi$.

Definition 6-4-4 : Evaluation of Formulae in Terms of a PLTL Model

For a PLTL model Ξ^n and a formula ϕ, the following hold.

i) $\Xi^n \models \mathbf{T}$ iff $\Xi^n \models \neg\bot$;

ii) $\Xi^n \models \phi \in \boldsymbol{P}$ iff ϕ is an atomic theorem;

iii) $\Xi^n \models \neg\psi$ iff $\Xi^n \not\models \psi$;

iv) $\Xi^n \models \wedge\psi\chi$ iff $\Xi^n \models \psi$ and $\Xi^n \models \chi$;

v) $\Xi^n \models X\psi$ iff $\Xi^n \models \psi$;

vi) $\Xi^n \models U\psi\chi$ iff $\Xi^n \models \neg\wedge\neg\psi\neg\chi$, $\Xi^n \models \neg\wedge\chi\neg XU\psi\chi$, and $\Xi^n \models U\psi T$.

That Definition 6-4-4 is true can be determined by inspection of the evaluation rules and semantic tableaux rules. This fact that is exploited in the forthcoming sections.

Section 6-5 : Theorems.

The process of deduction begins with a PLTL model Ξ^n, an assertion of *hypotheses* with the set W of valid NTNF formulae, and atomic theorems. The process takes the hypotheses W and the PLTL theses, and uses the rules of inference, defined below, to *derive* a theorem.

Definition 6-5-1 : PLTL Derivation

A *PLTL derivation* of a theorem ϕ from a PLTL model Ξ^n **=df** < W, $\Re$, H > [Bri], denoted $\Xi^n \vdash_{PLTL} \phi$, is a finite sequence of formulae Γ **=df** ($\phi_1, \phi_2, \ldots, \phi_p$), such that ϕ **=df** ϕ_p and, for each $i \leq p$, either :

i) ϕ_i is a thesis,

or ii) $\phi_i \in$ W or $\phi_i \in$ ***P*** is an atomic theorem,

or iii) for some j and $k < i$, ϕ_i results from one of the following *rules of inference*:

R1. Modus Ponens - MP

For ϕ_j **=df** $\supset\psi\chi$ and ϕ_k **=df** ψ then ϕ_i **=df** χ.

R2. G Insertion - GI

If ϕ_j **=df** ψ then ϕ_i **=df** $G\psi$.

Definition 6-5-2 : PLTL Theorem

A formulae ϕ is a *PLTL theorem* iff $\Xi^n \vdash_{PLTL} \phi$.

The notation $\Xi^n \vdash_{PLTL} \phi$ is used if ϕ is a conjugate of a PLTL theorem. One can derive from the PLTL model Ξ^n, an infinite set $\Gamma_{\infty\text{-LS}}$ **=df** { $\phi_1, \phi_2, \ldots, \phi_n, \ldots$ } of theorems and an infinite set of their conjugates $\Gamma_{\infty\text{-RS}}$ **=df** $\{\neg\phi_1, \neg\phi_2, \ldots, \neg\phi_n, \ldots\}$; i.e., $\Xi^n \vdash_{PLTL} \Gamma_\infty$. A uniform method of deriving Γ_∞ is given below. A repertoire of derived rules of inference and theorems for PLTL are provided in [MP83]. It will be shown that formulae obtained from derivations are valid.

Section 6-6 : Consistency.

This section shows that any derived theorem ϕ is validated by the PLTL model Ξ^n. That the PLTL deductive system is *consistent* is achieved by showing that Definition 6-5-1 does not produce a contradiction.

Definition 6-6-1 : Soundness

The PLTL deductive system is *sound* if $\Xi^n \mid\!-_{PLTL} \Gamma_{\infty\text{-}LS} \Rightarrow \Xi^n \mid\!= \Gamma_{\infty\text{-}LS}$.

Definition 6-6-1 formally states the objective of this section. The proof of Definition 6-6-1 is obtained by the Theorem 6-6-2. The notation $\mid\!-_{PLTL}$ is simplified to $\mid\!-$.

Theorem 6-6-2 : The PLTL Deductive System is Sound [Kri63].

$\Xi^n \mid\!-_{PLTL} \Gamma_{\infty\text{-}LS} \Rightarrow \Xi^n \mid\!= \Gamma_{\infty\text{-}LS}$.

Proof : By Construction of a Derivation ($\phi_1, \phi_2, \ldots, \phi_{p-1}$).

It must be shown that the set $\Gamma_{\infty\text{-}LS}$ of all formulae derived by Definition 6-5-1 is *consistent*, viz. sound. This proof shows that $\Gamma_{\infty\text{-}LS}$ is consistent by showing that it is not *contradictory*. To prove that $\Gamma_{\infty\text{-}LS}$ is contradictory it is sufficient to show that, for any subset of formulae $\{ \phi_1, \phi_2, \ldots, \phi_{p-1} \} \subset \Gamma_{\infty\text{-}LS}$, a formula ϕ_p can not be derived such that $\wedge\phi_p\neg\phi_p \in \Gamma_{\infty\text{-}LS}$; if $\wedge\phi_p\neg\phi_p$ can be derived, then consistency implies that $\Xi^n \mid\!= \phi_p$ and $\Xi^n \mid\!= \neg\phi_p$, a contradiction. If

$$\supset {}_1\wedge_{p-2} \phi_1\phi_2 \cdots \phi_{p-1} \wedge\phi_p\neg\phi_p \in \Gamma_{\infty\text{-}LS}$$

is valid and $\Gamma_{\infty\text{-}LS}$ is contradictory [Rob77][4]. A derivation of ϕ_p is an ordered sequence of theorems Γ^p **=df** $(\phi_1, \phi_2, \ldots, \phi_{p-1}, \phi_p)$, where theorem ϕ_p is concluded by one the conditions i), ii), or iii) of Definition 6-5-1 from the sequence

$$\Gamma^{p-1} \textbf{=df} (\phi_1, \phi_2, \ldots, \phi_{p-1}).$$

[4] The semantic tableaux procedure applied to $\neg\wedge {}_1\wedge_{p-2} \phi_1\phi_2 \cdots \phi_{p-1} \neg\wedge\phi_p\neg\phi_p$ must not close.

Therefore, a proof that $\Gamma_{\infty\text{-LS}}$ is not contradictory must show that if a derived ϕ_p is valid then its conjugate $\neg\phi_p$ is not valid; this can be achieved using the semantic tableaux procedure as outlined in the appendix. The implications of the semantic tableaux procedure are exploited by the remainer of this dissertation.

Each of the conditions i), ii), and iii) of Definition 6-5-1 must derive a valid formula. Condition iii) can not be invoked in a derivation until any of the conditions i) and ii) have been invoked at least two times in the case of rule of inference R1 or at least one time in the case of rule of inference R2; therefore, conditions i) and ii) are investigated first. It must be shown that theorems added to a derivation by i) and ii) do not create inconsistencies. The set of NTNF formulae W are defined valid; any atomic theorem is valid. The theorems created by the PLTL axiom schemata are shown to be valid, by using a modified semantic tableaux procedure that shows the validation of the theorem conjugate creates an inconsistency. The appendix shows this simple but tedious task. Thus, the hypotheses do not create an inconsistency.

Now, it must be shown that the rules of inference do not derive inconsistencies. Given that a sequence $\Gamma =\mathbf{df} (\phi_1, \phi_2, \ldots , \phi_{p-1})$ of theorems has been derived, $\Xi^n \models \Gamma$ holds.

The rule R1 preserves validity. If $\Xi^n \vdash_{\text{PLTL}} \supset\psi\chi$ and $\Xi^n \vdash_{\text{PLTL}} \psi$ are theorems, then $\Xi^n \models \supset\psi\chi$ and $\Xi^n \models \psi$. For $\Xi^n \vdash_{\text{PLTL}} \chi$ derived by Modus Ponens, it must be shown that $\Xi^n \models \chi$. The formula $\supset\psi\chi$ is an abbreviation for $\neg\wedge\psi\neg\chi$; therefore, $\Xi^n \models \neg\wedge\psi\neg\chi$. Since $\Xi^n \models \neg\wedge\psi\neg\chi$ implies $\Xi^n \not\models \wedge\psi\neg\chi$, and because $\Xi^n \models \psi$ implies $\Xi^n \not\models \neg\chi$, $\Xi^n \models \chi$.

The rule R2 preserves validity. If $\Xi^n \vdash_{\text{PLTL}} \psi$ is a theorem, then $\Xi^n \models \psi$. The theorem $\Xi^n \vdash_{\text{PLTL}} G\psi$ is derived by G Insertion from $\Xi^n \vdash_{\text{PLTL}} \psi$. By a reduction sequence, the formula $G\psi$ is converted to $U\psi\bot$. It must be shown that

$\Xi^n \models U\psi\bot$. It is assumed, reductio ad absurdum, that $\Xi^n \models \neg U\psi\bot$. Therefore, at least one of the following must be true $\Xi^n \models \wedge\neg\psi\neg\bot$, $\Xi^n \models \wedge\bot\neg XU\psi\bot$, and$\Xi^n \models \neg U\psi\mathbf{T}$. $\Xi^n \models \wedge\neg\psi\neg\bot$ contradicts $\Xi^n \models \psi$. $\Xi^n \models \wedge\bot\neg XU\psi\bot$ implies that $\Xi^n \models \bot$, which can not be true. $\Xi^n \models \neg U\psi\mathbf{T}$ implies that $\Xi^n \not\models U\psi\mathbf{T}$. Therefore, for every semantic tableaux s, $U\psi\mathbf{T} \in RC(s)$. But clearly, since $\Xi^n \models \psi$, ψ appears in all left columns. The semantic tableaux procedure applied to any $U\psi\mathbf{T} \in RC(s)$ closes. Therefore, $\Xi^n \models U\psi\mathbf{T}$. Therefore, reductio ad absurdum, $\Xi^n \models U\psi\bot$. **Q.E.D.**

The significance of this proof is that $\Gamma_{\infty\text{-LS}}$ is consistent; therefore, the PLTL deductive system is sound.

Section 6-7 : Maximal Consistent Set.

Given Definition 6-5-1, a set of all formulae derivable from the PLTL model Ξ^n is useful [Rob77][HC]. It is shown that derivation from the PLTL model Ξ^n implies validation with respect to the PLTL model Ξ^n.

Definition 6-7-1 : Consistent Formula

A formulae ϕ is *consistent* with respect to the PLTL model Ξ^n iff $\Xi^n \text{-}|_{PLTL} \neg\phi$.

When ϕ is consistent, $\neg\phi$ is said to be *inconsistent* with respect to ϕ.

Definition 6-7-2 : Consistent Set of Formulae

A set of formulae $\{ \phi_1, \phi_2, \ldots, \phi_p \}$ is *consistent* with respect to the PLTL model Ξ^n iff $\Xi^n \text{-}|_{PLTL} \neg_1\wedge_{p-1} \phi_1 \phi_2 \cdots \phi_p$.

The infinite set of all theorems that can be derived with the PLTL deductive system from the PLTL model Ξ^n is denoted $\Gamma_{\infty\text{-}LS} =$**df** $\{ \phi_1, \phi_2, \ldots, \phi_p, \ldots \}$.

Definition 6-7-3 : Consistent Infinite Set of Formulae

The infinite set of formulae $\Gamma_{\infty\text{-}LS} =$**df** $\{ \phi_1, \phi_2, \ldots, \phi_p, \ldots \}$ is *consistent* with respect to the PLTL model Ξ^n iff it does not contain an inconsistent finite subset of formulae; i.e., for every subset $\{ \phi_1, \phi_2, \ldots, \phi_p \} \subset \Gamma_{\infty\text{-}LS}$,

$$\Xi^n \text{-}|_{PLTL} \neg_1\wedge_{p-1} \phi_1 \phi_2 \cdots \phi_p.$$

Definition 6-7-4 : Contradictory Set $\Gamma_{\infty\text{-}LS}$

The set of derived formulae $\Gamma_{\infty\text{-}LS}$ is *contradictory* if it includes formulae $\phi_1, \phi_2, \ldots, \phi_p$ and a formula $\wedge\psi\neg\psi$ such that the formula $\supset_1\wedge_{p-1} \phi_1\phi_2 \ldots \phi_p \wedge\psi\neg\psi$ is derivable.

This definition of contradictory reflects the definition presented in the PLTL semantics. The formula $\supset_1\wedge_{p-1} \phi_1\phi_2 \ldots \phi_p \wedge\psi\neg\psi$ represents the derivation of $\wedge\psi\neg\psi$ from the derivation of ${}_1\wedge_{p-1} \phi_1\phi_2 \ldots \phi_p$. To derive $\supset_1\wedge_{p-1} \phi_1\phi_2 \ldots \phi_p \wedge\psi\neg\psi$ ultimately implies that $\Xi^n \models \wedge\psi\neg\psi$, a contradiction. The implication is that a set of derived formulae $\Gamma_{\infty\text{-}LS}$ is consistent iff it is not contradictory.

A formal means of generating the infinite set of formulae $\Gamma_{\infty\text{-LS}}$, such that if any formulae were added to $\Gamma_{\infty\text{-LS}}$ would result in an inconsistency, is a tool of this dissertation.

Definition 6-7-5 : Maximal Consistent Set

The *maximal consistent set* $\Gamma_{\infty\text{-LS}}$ derived from a set of hypotheses W of a PLTL model Ξ^n =**df** < W, $\Re$, H > is constructed in the following manner [HC] :

The set of all PLTL formulae, except W and the atomic theorems, are placed in their lexicographical ordering, resulting in the sequence $\phi_1, \phi_2, \ldots, \phi_p, \ldots$ of formulae. The set $\Gamma_{\infty\text{-LS}}$ is constructed by forming a series of sets, superscripted by $\mathbb{Z}^+$, ($\Gamma^0, \Gamma^1, \Gamma^2, \ldots, \Gamma^p, \ldots$) where :

i) Γ^0 is W and the atomic theorems;

ii) given Γ^{p-1}, Γ^p =**df** $\Gamma^{p-1} \cup \{ \phi_p \}$ iff ϕ_p can be added to Γ^{p-1} without creating an inconsistency; otherwise, Γ^p =**df** Γ^{p-1}. The set of all formulae that are included in $\Gamma^0 \cup \Gamma^1 \cup \Gamma^2 \cup \ldots \cup \Gamma^p \cup \ldots$ is denoted $\Gamma_{\infty\text{-LS}}$.

There are a number of consequences of this definition. First, each set in the series $\Gamma^0, \Gamma^1, \Gamma^2, \ldots, \Gamma^p, \ldots$ is consistent. By the definition of hypotheses W, Γ^0 is consistent. Moreover, Γ^p is consistent if Γ^{p-1} is consistent. Second, the set $\Gamma_{\infty\text{-LS}}$ is consistent. If Z is a finite subset of $\Gamma_{\infty\text{-LS}}$, and ϕ_m is the last formula of the lexicographical order in Z, $Z \subseteq \Gamma^m$. The set Γ^m is consistent. Because any subset of a consistent set is consistent, Z is consistent. Therefore, $\Gamma_{\infty\text{-LS}}$ contains no finite inconsistent subset and is consistent. Finally, the set $\Gamma_{\infty\text{-LS}}$ is maximal. A formula ϕ_m of the lexicographical ordering and consistent with $\Gamma_{\infty\text{-LS}}$ can be selected. The formula is consistent with any subset of $\Gamma_{\infty\text{-LS}}$, which includes Γ^{m-1}. In the construction of $\Gamma_{\infty\text{-LS}}$, ϕ_m is added to Γ^{m-1} to form Γ^m and, therefore, is in $\Gamma_{\infty\text{-LS}}$.

Three lemmata are a consequence of Definition 6-7-5. For the remainder of this dissertation $\Gamma_{\infty\text{-LS}}$ is shortened to Γ_∞.

Lemma 6-7-6 :

If Γ_∞ is maximal consistent set derived from W and if ϕ is a formula, it is not true that ϕ and $\neg\phi$ are both in Γ_∞.

Lemma 6-7-7 :

If Γ_∞ is maximal consistent set derived from W, then either

i) $\phi \in \Gamma_\infty$ and $\neg\phi \notin \Gamma_\infty$,

ii) $\phi \notin \Gamma_\infty$ and $\neg\phi \in \Gamma_\infty$, or

iii) $\phi \notin \Gamma_\infty$ and $\neg\phi \notin \Gamma_\infty$.

The lemmata 6-7-6 and 6-7-7 are a result of the fact that the PLTL deductive system does not generate a contradiction.

Lemma 6-7-8 :

If Γ_∞ is maximal consistent set derived from W and if ψ and χ are formulae, then if $\psi \in \Gamma_\infty$ and $\supset\psi\chi \in \Gamma_\infty$, then $\chi \in \Gamma_\infty$.

The previous lemma results from the rule R2. The proofs are found in [HC].

Some additional facts about the maximal consistent set Γ_∞ are required.

Lemma 6-7-9 :

The maximal consistent set Γ_∞ is a set of PLTL formulae such that [MP85][Dal] :

i) $\mathbf{T} \in \Gamma_\infty$ and $\perp \notin \Gamma_\infty$;

ii) $\psi \notin \Gamma_\infty \Rightarrow \neg\psi \in \Gamma_\infty$ or $\neg\psi \notin \Gamma_\infty$;

iii) $\psi \in \Gamma_\infty \Rightarrow \neg\psi \notin \Gamma_\infty$;

iv) $\neg\psi \in \Gamma_\infty \Rightarrow \psi \notin \Gamma_\infty$;

v) $\wedge\psi\chi \in \Gamma_\infty \Leftrightarrow \chi \in \Gamma_\infty$ and $\psi \in \Gamma_\infty$;

vi) $X\psi \in \Gamma_\infty \Leftrightarrow \psi \in \Gamma_\infty$;

vii) $U\psi\chi \in \Gamma_\infty \Leftrightarrow \neg\wedge\neg\psi\neg\chi \in \Gamma_\infty, \neg\wedge\chi\neg XU\psi\chi \in \Gamma_\infty$, and $U\psi\mathbf{T} \in \Gamma_\infty$;

viii) $U\psi\mathbf{T} \in \Gamma_\infty \Rightarrow \neg\psi \notin \Gamma_\infty$.

Proof :

i), ii), iii), iv), and v) A consequence of the definition of the maximal consistent set, which can not produce a contradiction.

vi) For this to be true, $X\psi \in \Gamma_\infty \Rightarrow \psi \in \Gamma_\infty$ and $X\psi \notin \Gamma_\infty \Rightarrow \psi \notin \Gamma_\infty$ must hold. It is assumed, for a proof reductio ad absurdum, that $X\psi \in \Gamma_\infty$ and $\psi \notin \Gamma_\infty$. By ii), $\psi \notin \Gamma_\infty$ has two possible consequences. First, if $\neg\psi \in \Gamma_\infty$ then by rule R2, $G\neg\psi \in \Gamma_\infty$. Now, by axiom A3, $\supset G\neg\psi X\neg\psi \in \Gamma_\infty$. Thus by rule R1, $X\neg\psi \in \Gamma_\infty$; by axiom A7, $\neg X\psi \in \Gamma_\infty$. By iv), $X\psi \notin \Gamma_\infty$ is a contradiction. Therefore, reductio ad absurdum, $\psi \in \Gamma_\infty$. Second by ii), if $\neg\psi \notin \Gamma_\infty$, then either $\neg\neg\psi \in \Gamma_\infty$ or $\neg\neg\psi \notin \Gamma_\infty$. A contradiction results immediately from $\psi \in \Gamma_\infty$. The other case is the original reductio ad absurdum assumption and the case where $X\psi \in \Gamma_\infty$, $\neg\psi \notin \Gamma_\infty$, and $\psi \notin \Gamma_\infty$ must be considered. If $X\psi \in \Gamma_\infty$ then, by Theorem 6-6-2, $\Xi^n \models X\psi$ and, by Definition 6-4-4 $\Xi^n \models \psi$. $\neg\psi \notin \Gamma_\infty$ and $\psi \notin \Gamma_\infty$ imply that there are some

semantic tableaux s with $\neg\psi \in RC(s)$ and some semantic tableaux with $\psi \in RC(s)$. But $\Xi^n \models \psi$ implies that, for every semantic tableaux s, $\psi \in LC(s)$; a contradiction exists. Now it is assumed, for a proof reductio ad absurdum, that $X\psi \notin \Gamma_\infty$ and $\psi \in \Gamma_\infty$. By rule R2, $G\psi \in \Gamma_\infty$. By axiom A3, $\supset G\psi X\psi \in \Gamma_\infty$. Thus rule R1, $X\psi \in \Gamma_\infty$, resulting in a contradiction. Therefore, $X\psi \in \Gamma_\infty \Rightarrow \psi \in \Gamma_\infty$ and $X\psi \notin \Gamma_\infty \Rightarrow \psi \notin \Gamma_\infty$ must hold.

vii) By rule R2, if $U\psi\chi \in \Gamma_\infty$ then $GU\psi\chi \in \Gamma_\infty$. By axiom A3, $\supset GU\psi\chi XU\psi\chi \in \Gamma_\infty$. By Rule R1, $XU\psi\chi \in \Gamma_\infty$. By axiom A9, viz. $\equiv U\psi\chi \vee \psi \wedge \chi XU\psi\chi$, and by the definition of maximal consistent set, $\equiv U\psi\chi \vee \psi \wedge \chi XU\psi\chi \in \Gamma_\infty$. By Rule R1 and $U\psi\chi \in \Gamma_\infty$, $\vee\psi\wedge\chi XU\psi\chi \in \Gamma_\infty$. By the Distributive Law, $\wedge\vee\psi\chi\vee\psi XU\psi\chi \in \Gamma_\infty$. By the definition of the disjunction abbreviation,

$$\wedge\neg\wedge\neg\psi\neg\chi\neg\wedge\neg\psi\neg XU\psi\chi \in \Gamma_\infty,$$

$$\neg\wedge\neg\psi\neg\chi \in \Gamma_\infty, \text{ and}$$

$$\neg\wedge\neg\psi\neg XU\psi\chi \in \Gamma_\infty.$$

By Axiom A10 and Rule R1, $F\psi \in \Gamma_\infty$; thus, $U\psi\mathbf{T} \in \Gamma_\infty$. Now given that

$$\neg\wedge\neg\psi\neg\chi \in \Gamma_\infty,$$

$$\neg\wedge\neg\psi\neg XU\psi\chi \in \Gamma_\infty, \text{ and}$$

$$U\psi\mathbf{T} \in \Gamma_\infty,$$

it follows that $\Xi^n \models \neg\wedge\neg\psi\neg\chi$, $\Xi^n \models \neg\wedge\neg\psi\neg XU\psi\chi$, and $\Xi^n \models U\psi\mathbf{T}$. It is assumed, for a proof reductio ad absurdum, that $U\psi\mathbf{T} \notin \Gamma_\infty$; therefore, either $\Xi^n \models \wedge\neg\psi\neg\chi$, $\Xi^n \models \wedge\neg\psi\neg XU\psi\chi$, and $\Xi^n \models \neg U\psi\mathbf{T}$. Any one of them causes a contradiction. Therefore, reduction ad absurdum, $U\psi\mathbf{T} \in \Gamma_\infty$.

viii) It is assumed, for a proof reductio ad absurdum, that $U\psi\mathbf{T} \in \Gamma_\infty$ and $\neg\psi \in \Gamma_\infty$. $\neg\psi \in \Gamma_\infty$ means that $\psi \notin \Gamma_\infty$. $U\psi\mathbf{T} \in \Gamma_\infty$ implies that there

must be semantic tableaux s with $\psi \in LC(s)$. This clearly contradicts $\neg\psi \in \Gamma_\infty$.

Q.E.D.

Section 6-8 : Completeness.

This section shows the converse of Section 6-6.

Definition 6-8-1 : Sufficiency

The PLTL deductive system is *sufficient* iff $\Xi^n \models \Gamma_\infty \Rightarrow \Xi^n \vdash_{PLTL} \Gamma_\infty$.

A formula ϕ is sufficient iff $\Xi^n \models \phi \Rightarrow \Xi^n \vdash_{PLTL} \phi$. If the PLTL deductive system is sufficient then every tautology is derivable. Theorem 6-6-2 proves that the PLTL deductive system is sound; for the maximal consistent set of formulae Γ_∞, there exists a PLTL model that verifies Γ_∞. A proof of sufficiency is given here, using a *Henkin type proof* [HC][Sac] based on the maximal consistent set Γ_∞, on NTNF formulae, and on some concepts presented in [LP].

Theorem 6-8-2 : PLTL deductive system is sufficient.

Proof : By Induction.

It is assumed that a PLTL model $\Xi^n =$**df** $< W, \Re, H >$ exists. To avoid a contradiction for any $\phi \in \Gamma_\infty$, it is not the case that $\Xi^n \models \phi$ and $\Xi^n \not\models \phi$. Alternatively, $\Xi^n \models \phi$ implies that $\Xi^n \not\models \neg\phi$, and $\Xi^n \models \neg\phi$ implies that $\Xi^n \not\models \phi$. The proof is achieved by i) deriving a maximal consistent set Γ_∞ from Ξ^n and showing that ii) if $\Xi^n \models \phi$ then $\phi \in \Gamma_\infty$. This exploits the advantages accrued by Theorem 6-6-2.

Part i) is not difficult. By Definition 6-6-5, a maximal consistent set Γ_∞ is derived from Ξ^n; but, some further definitions are required. The set denoted Γ_i is derived from W of the subordinate $\Xi^i : 1 \le i \le n$ of Ξ^n; the derivation is denoted $\Xi^i \vdash_{PLTL} \Gamma_i$. Thus, $\Xi^i \vdash_{PLTL} \Gamma_i : i < j$ is the *subordinate* of $\Xi^j \vdash_{PLTL} \Gamma_j$. Also, Γ_{i-1} is the *immediate subordinate* of Γ_i. Thus, $\Gamma_n =$**df** Γ_∞. Because $\Gamma_1 \subset ... \subset \Gamma_n$, if $\phi \in \Gamma_k : 1 \le k \le n$, then $\phi \in \Gamma_j : k \le j \le n$. The characteristics of the maximal consistent set are exploited in the remainder of the proof; if $\phi \in \Gamma_\infty$ then $\Xi^n \models \phi$ and if $\phi \notin \Gamma_\infty$ then $\Xi^n \not\models \phi$.

Part ii) of the Henkin proof requires that every formula ϕ validated by the PLTL model Ξ^n =**df** < W, $\Re$, H > is found in the maximal consistent set Γ_∞. By Definition 6-7-3, it is only necessary to show that there does not exist an inconsistent set of formulae included in Γ_∞; i.e., if there is an inconsistency in any $\Gamma_i \subset \Gamma_\infty$, then there is an inconsistency in Γ_∞. It is shown, with a proof by induction on the length of formula ϕ [HC], vide Definition 5-2-7, that ϕ is sufficient. At each stage of the induction, when $\Xi^n \models \phi$, the proof must show that there is a derivation $\Xi^n \vdash_{PLTL} \phi \in \Gamma_\infty$. Since Γ_∞ is maximal and every formula is checked with respect to Γ_∞ by the induction, the proof shows that all derived formulae are complete.

n = 1 : The induction begins with **T**, $\perp$, and the atomic tableaux. The formula **T** is automatically validated by Ξ^n and is included in Γ_∞, while $\perp$ is not. Each atomic proposition $p \in \boldsymbol{P}$ is checked. That $\Xi^n \models p$ means that p is an atomic theorem. The term p must be included in every leading term of the NTNF formula in Ξ^1. Since $p \in \Gamma_1$, $p \in \Gamma_\infty$.

n > 1 :

$\neg\psi$: The induction hypothesis implies that ψ is sufficient. That formula $\neg\psi$ is sufficient must be proved, therefore $\Xi^n \models \neg\psi$ is assumed. By the induction hypotheses, and since a contradiction occurs when $\Xi^n \models \psi$, $\Xi^n \not\models \psi$. Thus, $\psi \notin \Gamma_\infty$, and $\neg\psi \in \Gamma_\infty$ or $\neg\psi \notin \Gamma_\infty$. If $\neg\psi \in \Gamma_\infty$ then $\neg\psi$ is sufficient. Since Γ_∞ is maximally consistent, $\neg\psi \notin \Gamma_\infty$ implies that there exists at least one semantic tableaux s with $\psi \in$ LC(s); this contradicts $\Xi^n \models \neg\psi$.

$\wedge\psi\chi$: The induction hypothesis implies that ψ and χ are sufficient. That formula $\wedge\psi\chi$ is sufficient must be proved, therefore $\Xi^n \models \wedge\psi\chi$ is assumed. By the induction hypothesis and to avoid any contradiction, $\Xi^n \models \psi$, $\psi \in \Gamma_\infty$, $\Xi^n \models \chi$, and $\chi \in \Gamma_\infty$. By Lemma 6-7-9 v), $\wedge\psi\chi \in \Gamma_\infty$.

$\underline{X\psi}$: The induction hypothesis implies that ψ is sufficient. That formula $X\psi$ is sufficient must be proved, therefore $\Xi^n \models X\psi$ is assumed. Because $\Xi^n \models \psi$, by the induction hypothesis $\psi \in \Gamma_\infty$, $\neg\psi \notin \Gamma_\infty$ and $\Xi^n \not\models \neg\psi$. Lemma 6-7-9 vi), if $\psi \in \Gamma_\infty$ then $X\psi \in \Gamma_\infty$.

$\underline{U\psi\chi}$: The induction hypothesis implies that ψ and χ are sufficient. That formula $U\psi\chi$ is sufficient must be proved, therefore $\Xi^n \models U\psi\chi$ is assumed. $\Xi^n \models U\psi\chi$ implies that $\Xi^n \models \neg\wedge\neg\psi\neg\chi$, $\Xi^n \models \neg\wedge\chi\neg XU\psi\chi$, and $\Xi^n \models U\psi\mathbf{T}$. It is assumed, for a proof reductio ad absurdum, that $U\psi\chi \notin \Gamma_\infty$; hence, the three possibilities

$$\wedge\neg\psi\neg\chi \in \Gamma_\infty,\ \wedge\neg\psi\neg XU\psi\chi \in \Gamma_\infty, \text{ and } \neg U\psi\mathbf{T} \in \Gamma_\infty$$

must be considered. Contradictions result by lemma 6-6-2, because

$\wedge\neg\psi\neg\chi \in \Gamma_\infty$ implies $\Xi^n \models \wedge\neg\psi\neg\chi$ and $\wedge\chi\neg XU\psi\chi \in \Gamma_\infty$ implies

$$\Xi^n \models \wedge\chi\neg XU\psi\chi.$$

The third possibility means that, by lemma 6-6-2, $\Xi^n \models \neg U\psi\mathbf{T}$, which contradicts $\Xi^n \models U\psi\mathbf{T}$. **Q.E.D.**

The definition of soundness and sufficiency lead to the following definition and theorem.

Definition 6-8-3 : Completeness

The PLTL deductive system is *complete* when $\Xi^n \vdash_{PLTL} \Gamma_\infty \Leftrightarrow \Xi^n \models \Gamma_\infty$.

A formula ϕ is complete when $\Xi^n \vdash_{PLTL} \phi \Leftrightarrow \Xi^n \models \phi$.

Theorem 6-8-4 : The PLTL deductive system is complete.

Proof :

By lemmata 6-6-2 and 6-8-2, and the fact that Γ_∞ is maximally consistent.

Section 6-9 : Validation Model Correctness Criteria Formulae Classifications.

The PLTL deductive system is used to derive tautologous formulae that comprise the validation model correctness criteria. If it can be shown that that maximal consistent set Γ_∞ contains the correctness criteria PLTL formulae, then the system design has been validated. The proofs of consistency and completeness are necessary for application of this validation method. The remainder of this chapter defines in detail the types of safety and liveness formulae that could be defined for a validation model using PLTL.

This section defines three classes [MP81i] of formulae that express the correctness criteria Γ for the validation models. The validation model must generate a PLTL model Ξ^n for Γ; i.e., the entire execution of the validation model must adhere to the requirements specified by the correctness criteria formulae Γ.[1] The notation $\underline{s_i}$ identifies a semantic tableaux formula that corresponds to atomic semantic tableaux s_i. The term corresponding to the initial state of a validation model is denoted $\underline{s_0}$ and the term corresponding to a final state is denoted $\underline{s_f}$. Atomic tableaux formulae are not necessarily valid. Axioms of the form $\supset \underline{s_i}\psi$ express a fact ψ about a particular semantic tableaux s_i. If more than one semantic tableau is involved, a conjunction can be used. To express the same fact about all semantic tableaux, the formula ψ suffices.

The first classification of formulae express *safety properties* that are invariant over the PLTL model Ξ^n. These properties are expressed by formulae having the schema $\supset \psi G \chi$.

Definition 6-9-1 : Partial Correctness

---

[1] i.e., the formulae must be valid when interpreted in terms of the corresponding system Ξ^n of semantic tableaux.

The PLTL model must have the property of *partial correctness*. If the subformula ψ is a *pre-condition* that holds at the initial state of the model and χ is a *post-condition* that holds for any particular state $\underline{s}_i$, then the partial correctness property is expressed by the formula $\supset \supset \underline{s}_0 \psi \; G \supset \underline{s}_i \chi$.

But for a cyclic validation model, the correctness criteria might require that the certain properties are maintained throughout the execution cycle, viz. with the infinity set.

Definition 6-9-2 : Clean Behaviour

The property of *clean behaviour* maintained over all semantic tableaux $\{ s_1, s_2, \ldots, s_k \}$ is expressed by the formula

$$\supset T \; G \wedge_{k-1} \supset \underline{s}_1 \chi \supset \underline{s}_2 \chi \supset \underline{s}_3 \chi \ldots \supset \underline{s}_k \chi .$$

In general, clean behaviour is defined for the strongly connected component of the reachability graph. Partial correctness and clean behaviour can be combined into a property that must hold in all semantic tableaux of Ξ^n.

Definition 6-9-3 : Global Invariant

A formula that expresses a *global invariant* uses a subformula β to describe conditions that must hold for all semantic tableaux of Ξ^n. Given a subformula β true at the start state s_0 of a validation model, the global invariant formulae has the form $\supset \supset \underline{s}_0 \beta \; G\beta$.

A cyclic validation model does not terminate. However, certain error conditions, called *deadlocks*, prevent a validation model from continuing execution; it must be ensured that such an error condition does not occur.

Definition 6-9-4 : Deadlock Freedom

Given a formula ε that expresses an error condition resulting in a deadlock, a formula that expresses *deadlock freedom* has the form $\supset \supset \underline{t}_0 \psi \; G \neg \varepsilon$; it states that from an initial state, the error condition does not occur.

In general, ε could be any error condition. For example, to preserve the consistency of a resource, distributed mutual exclusion must be provided. Only one system component at a time may be in the critical section. When a system component i accesses the resource, the formula α_i expresses the fact that the system component is in a *critical section*. If α_i is true then no other α_j can be true.

Definition 6-9-5 : Mutual Exclusion

A formula that expresses the property that two system components i and j are *mutually excluded* from a critical section is $\supset \supset \underline{s}_0 \psi \, G \neg \wedge \alpha_i \, \alpha_j$.

Here $\wedge \alpha_i \, \alpha_j$ expresses the fact that both system components are simultaneously in the same critical section, an error condition.

Hence, safety properties express the fact that nothing bad ever happens [MP81i].

The second classification of formulae express *liveness properties*, where, given a state, some property must eventually be true at some future state held by the validation model. Liveness properties have the general schema $\supset \psi F \chi$. Partial correctness ensures that a state s_i exhibits the desired property, but does not guarantee that the state is ever held.

Definition 6-9-6 : Total Correctness

The property of *total correctness* ensures that any state s_i, where the property χ holds, is eventually held. Formulae to express this property have the form $\supset \supset \underline{s}_0 \psi \, F \supset \underline{s}_i \chi$, with ψ and χ the same as partial correctness.

In cyclic validation models certain properties must occur periodically; for example, the P / C system validation model is cyclic.

Definition 6-9-7 : Intermittent Assertion

An *intermittent assertions* of a property ψ is expressed by the formula $\supset \psi F \psi$.

Of course, that an occurrence of a property implies that it will occur periodically does not imply that the property is ever asserted in the first place. Indeed, it is often required that a particular property is asserted at some time.

Definition 6-9-8 : Accessibility

A formula to express the fact that a property ψ is *accessible* has the schema $\supset TF\psi$.

For example, a mutual exclusion mechanism must guarantee that if a system component requests access to a resource then it is eventually given access to the resource. There is a liveness analogue to the safety property of deadlock freedom. A validation model holding a particular state s_i must not hold that state forever.

Definition 6-9-9 : Deadlock Freedom

A liveness formula to express the property of deadlock freedom has the form $\supset \underline{s_i} F \neg \underline{s_i}$.

Hence, liveness properties state that something good eventually happens to the validation model.

Servers generate validation models that are cyclic; that is, a server begins in an idle state, a *request* is made for a service provided by the server, the server *responds* to the request, and once the request has been granted the server normally returns to the idle state. *Responsiveness* is a class of properties required of cyclic validation models, concerning requests for services and the resulting responses. In the following definitions, ψ is a formula expressing the fact that a request has been made and χ is a formula expressing the fact that a response has been granted.

Definition 6-9-10 : Response to Insistence

Given some, possibly permanent, issue of a request ψ, the property of *response to insistence* requires that eventually a response χ will be returned; this is expressed by $\supset G\psi F\chi$.

When the response χ is made and the request ψ is satisfied, ψ must not hold forever and χ must begin to hold at some future marking; therefore, the formula $\neg G \wedge \psi \neg \chi$ must be included with response to insistence. *Polling* is a common method of issuing a request, where a request ψ is issued repeatedly until a response χ is returned. By persistently submitting a request for a resource that is currently in use, eventually the resource will become free and the request can be honoured.

Definition 6-9-11 : Response to Persistence

A formula that expresses the property of *response to persistence* is $\supset GF\psi F\chi$, or alternatively $\neg G \wedge F\psi \neg \chi$.

If a request is stored in a buffer, only one request is sufficient to guarantee a response.

Definition 6-9-12 : Response to an Impulse

A formula that expresses a *response to an impulse* has the form $\supset \psi F\chi$ or $G \supset \psi F\chi$.

Up to this point, the responsiveness formulae employ the temporal operators F and G.

However, there are responsiveness properties that can not be expressed without the until operator [GPSS]. Normally, a response occurs if and only if a request has been made.

Definition 6-9-13 : Absense of Unsolicited Response

A formula to express the *absence of unsolicited responses* has the form $\supset F\chi U \neg \chi \psi$ and states that, for a response χ to occur, it can not happen until the request ψ has been issued or ψ and χ happen concurrently.

If multiple requests are made by a number of different system components, to prevent some system components from being denied access to the resources, it is sometimes necessary to impose an order on the responses. A simple method is to place requests in a queue, which imposes a *First In First Out*, FIFO, discipline on the order requests are considered and thus an order on the responses.

Definition 6-9-14 : Precede Function

The *precede function* $Pr(\psi_1, \psi_2)$ is an abbreviation of the formula $\supset F\psi_2 U\psi_1 \neg\psi_2$, which expresses the FIFO discipline.

The precede function is a formula that expresses the fact that a request ψ_1 is issued before request ψ_2. The precede function is building block from which ordering can be described for requests and responses.

Definition 6-9-15 : Strict Fairness

A formula to express *strict fairness*, that responses occur in the order their requests are made, for two requests ψ_1 and ψ_2 and two associated responses χ_1 and χ_2 is $\supset Pr(\psi_1, \psi_2)Pr(\chi_1, \chi_2)$.

The strict fairness states that if ψ_1 and ψ_2 are issued in that order, then the responses must follow in the order χ_1 and χ_2. Unfortunately, this formula allows a single ψ to be followed by any number of responses χ; what is really required is that at least one ψ is interleaved between each χ. The formula $\wedge Pr(\psi, \chi)G\supset\chi Pr(\psi, \chi)$ describes the required property.

Chapter 7 : CONCLUSIONS.

Section 7-1 : Summary.

This dissertation presents a methodology to derive, in a heuristic manner, valid multiprocessor system designs. The methodology is restricted to producing abstract designs at the architectural level and does not consider implementation of the designs. An iterative process of defining a system design, constructing a functionally equivalent validation model of the design, and analysing the validation model for safety and liveness correctness criteria continues until a valid system design is derived. It is claimed that this methodology guides and assists the very complex task of designing multiprocessor systems.

A subclass of Petri nets is used to model the concurrent nature of multiprocessor system designs. A validation model consists of a collection of n component models. Each component model is a finite state machine state-based transition model, $< \Pi , \Delta , A , \Omega , H >$, but is presented as a graph. The graphs are coupled to represent the interaction among the system components. The coupled graphs result in a Petri net subclass, $< \Pi , \Sigma , I , O , m_0 , M_f , H >$, which permits the application of Petri net reachability analysis. The result of reachability analysis is a reachability graph, $< S , T , m_0 , M_f , H >$, a finite state machine state-based transition model that represents the validation model global states and their transitions. The reachability graph is a simplification of the Petri net without loss of relevant information necessary for temporal analysis purposes.

Temporal analysis converts the strongly connected, directed reachability graph to the validation model α-relation structure Ξ conforming to the linear, discrete, and absolute

view of time. The execution over time of a validation model is represented by constructing the set of distinct finite and infinite walks that begin with a particular vertex in the reachability graph, where each walk represents one possible sequence of global states. Standards of conduct define the α-relation, which select a subset of all countably infinite walks $\Delta_{\omega-\alpha}(RG)$. This set of infinite walks is condensed into a collection of finite walks, that contains sufficient information to specify the maximal strongly connected reachability graph component, SCC-$\Delta_{\omega-\alpha}(RG)$, specified by the α-relation. The information can be expressed as a family of predicates, $\{ A_i : 1 \leq i \leq n \}$, whose sentences correspond to the finite walks. The validation model α-relation structure Ξ,

$$< S, \{ A_i : 1 \leq i \leq n \}, \text{SCC-}\Delta_{\omega-\alpha}(RG), \{ H(s) \ni \forall s \in S \} >,$$

is defined from the predicates and forms a structure of finite sequences of propositions providing the basis for the PLTL definitions.

The safety and liveness criteria are expressed in the language of PLTL. The rules of syntax select words from the PLTL language which can be assigned meaning. The temporal semantics determine whether a word asserts something true in a given sequence of propositions. The semantic tableaux procedure is a proof, reductio ad absurdum, demonstrating that a word is valid in the relation structure.

The PLTL deductive system is a syntactic method of selecting tautologies with respect to the validation model α-relation structure. The terms of the deductive system correspond to the validation model global states. A set of axiom schemata define the theses of the deductive system. The validation model α-relation structure Ξ is expressed with a set of hypotheses. Together, the terms, theses, and hypotheses define the axiomatic basis for the PLTL deductive system. The process of deduction derives a maximal consistent set Γ_∞ of theorems from the axiomatic basis. If the maximal consistent set Γ_∞ contains the correctness criteria formulae, the system design is valid. The proofs of consitency and completeness ensure that this is the case.

Section 7-2 : Epilogue.

The result of this disseration is a technique of validating that safety and liveness correctness criteria hold for abstract multiprocessor system designs. The techniques are intended to be part of an overall heuristic system design methodology. The technique is useful to solve multiprocessor design problems, but can be extended to improve its usefulness. Furthermore, the methodology has some weaknesses that should be understood.

Computer Assisted Design System. While all of the techniques presented here could be performed manually, automation would be a practical step towards dealing with the complexity of designing a multiprocessor system. This methodology provides the theoretical foundation for a *Computer Assisted Design*, CAD, system. A CAD system would consist of three parts : a Petri net model *specification and storage system* using an *interactive graphics capability* to specify validation models and an *automatic theorem prover* to automate the validation technique.

Since a validation model is an abstract mathematical representation, it lends itself to automated specification and storage. First, an internal representation for Petri nets is required; vector addition systems and vector replacement systems are likely candidates. Given an internal representation of a Petri net, algorithms to convert the Petri net representation to the reachability graph to the system of semantic tableaux are necessary; the validation model α-relation structure Ξ would be the final form of the models' internal representation.

Furthermore, some means to ease the system designer's task when entering the model into the CAD system is necessary. An interactive graphics capability is useful, since illustrations are an effective means for humans to conceptualize the system design. The interactive graphics capability would have to be integrated with the CAD system's

internal representations and conversion algorithms. The interactive graphics capability must overcome the difficulty of illustrating interaction among system components. If a validation model were to be constructed with more than two interacting components, interaction would be difficult to illustrate in two dimensions.

An automatic theorem prover requires that the problems of determining truth, satisfiability, and validity of safety and liveness criteria PLTL formulae are *computable* [Dav], that is, that the problems can be solved by an algorithm. The semantic tableaux procedure is used to show validity of safety and liveness criteria formulae, but results in some very complex semantic tableaux constructions that would consume considerable processing time and space. No attempt has been made here to simplify the process, though simplification is possible; for example, if the procedure has been applied to a formula, it is not necessary to repeat the process when the formula appears as a subformula of another formula.

But, the problem of state space explosion must be addressed before the automatic theorem prover can be implemented. The state space explosion affects the processing time and space required by the automatic theorem prover, since the measurement of complexity is based on the number of vertexes in the reachability graph and the length of the correctness criteria formulae [LP]. Indeed, research has shown that the problems of determining truth, satisfiability, and validity for logics similar to PLTL are PSPACE-complete [SC][Lad]. Thus, the automatic theorem prover might not be usable for complex system design models with a large number of states. However, the authors of [LP] argue that, since many correctness criteria can be expressed with formulae of relatively small length, a practical automatic theorem prover is possible. The decomposition provided by phases is illusionary, since many assertions must be proved across all phases; but, phases may reduce complexity in some situations.

Methodology Extensions. The validation model is a subclass of Petri nets that generates regular languages. Though it is claimed that this model is sufficient for modelling purposes, a study of how this theory could be extended to Petri nets defining bounded context-free languages may be of some use.

Since the automatic theorem prover would encounter the state space explosion problem, it might be useful to develop a repertoire of common, basic validation model constructs with validated correctness criteria. Validation for the constructs would be peformed once; and once performed, the constructs would serve as building blocks for more complex systems. For example, a common building block is the Producer / Consumer system. Any criteria validated for the Producer / Consumer system could be transferred to systems incorporating that building block. This notion conforms to the idea of deriving a system design by composition.

Using the PLTL language to define safety and liveness criteria is more precise and concise than a prose description; but, the PLTL language could be improved with a number of extensions.[1] Although the fundamental next-time and until temporal operators can be used to express any relevant safety and liveness criteria, it would be useful to have additional abbreviations for common formulae incorporating these operators, such as in [SM-S]. The existential quantifiers could be added to PLTL [CES]. Branching time versions of the logic might be more appropriate for the reachability tree [CE][EH82][EH83][EL]. It would, of course, be useful to find the most efficient means of verifying formulae; this would require an in-depth study of inherent complexity issues.

Abandoning Petri nets, to avoid the state space explosion, in favour of hybrid modelling techniques, such as LOTOS, ESTL, SDL, and FAPL, might be a prudent

[1] The prefix notation is chosen to avoid cumbersome bracketing conventions, but is not commonly employed. For some people, the infix notation is an improvement.

course of action. There could be two approaches. First, PLTL could express the correctness criteria for the validation model specified with the techniques. This would require methods of validating the formulae with respect to the models. Second, PLTL could serve as a low level language from which statements in the modelling techniques could be constructed. Proof that the modelling technique statements are correct could be achieved by investigating the underlying PLTL formulae.

Limitations. This methodology is developed strictly for design purposes, therefore it does not address the problem of implementing a system design. It is analogous to data flow diagrams and structure diagrams of software engineering. Like these software engineering techniques, the methodology provides a framework from which hardware and software designs can be conceptualized. Why use a Petri net when CSP, or even an actual programming language can be used as an model? The Petri net is a fundamental structure that can represent concurrent states; thus, at the very least, the Petri net provides a foundation from which integration of PLTL with other techniques could be pursued.

This methodology, as was stated in Chapter 1, ignores the issue of predicting the potential performance of a system design. Performance is measured in terms of some explicit definition of real-time periods. Because of the implicit time definition within PLTL, there is no concept of a time measurement. Indeed, the concept of safety and liveness criteria over time is concerned with whether something happens, how it happens, whether something does not happen, but not when something happens. Just because the design is correct as far as this dissertation is concerned, does not mean that the design is acceptable. Fortunately, the methodology presented here does not preclude the implementation of some means of measuring performance. For example, the validation model could be modified into a timed Petri net to model logical time or include real-time measurements.

APPENDIX

This appendix supplies the portion of Theorem 6-6-2 showing that the PLTL axiom schemata are valid. The proof of validity for each axiom schema is considered separately. Each proof begins by listing the axiom schema whose validity is to be proved on the left side of the page. Because the axiom schemata of Definition 6-3-2 are presented as abbreviated PLTL formulae, the abbreviated axioms are converted with a reduction sequence, Definition 5-2-9, to an unabbreviated PLTL formulae ϕ. The reduction sequences are notated as per the example on page 5-95. The proof of validity is achieved using the semantic tableaux procedure, Definition 5-4-1. The semantic tableaux procedure begins with an arbitrary atomic semantic tableau $T_1 =$**df** $(s) \in \Xi_1$. The PLTL axiom ϕ is placed in the right column of T_1 and a semantic tableaux construction T_m is created by applying the semantic tableaux rules. If a contradiction occurs in all attempts to verify $\phi \in RC(s)$, i.e. the conjugate of ϕ, then it may be concluded that the axiom is valid. A proof of validity is given for each axiom schema.

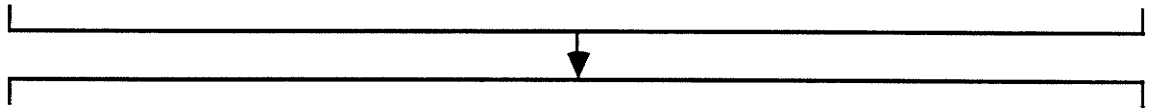
However, there are some typographical difficulties encountered when showing the proofs; therefore, the following conventions are used. The proofs are organized so that reading from the top of the page down follows closely with the semantic tableaux procedure. The proofs use the following conventions.

i) The semantic tableaux, including T_1, are indicated by the symbol

indicating the top of a semantic tableau and the symbol

indicating the bottom of a semantic tableau. The left column of the semantic tableaux appears on the left side of the page and the right column appears on the right side of the page. The converted PLTL axioms are placed at the top in the right column.

ii) If one of the XL and XR rules is applied for the first time in a semantic tableau, a new semantic tableau is created; the following symbol separates the two semantic tableaux, for example, in the proofs for A3, A6, A7.



Any additional applications of XL and XR place the consequent formulae in the new tableau. The proofs, here, do not require the creation of more than one additional semantic tableau.

iii) The semantic tableaux rules, Definition 5-4-1, are applied to the formulae in a semantic tableaux, usually one formula at a time. The application of a rule is not explicitly noted; in addition, the *target* formulae, the formulae the rule are applied to, and the *consequent* formulae, the formulae resulting from the application the rule, are not explicitly identified. However when either the $\neg$L or $\neg$R rule is applied, the consequent formula is always placed on the same line as the target formula, but on the opposite column. In the other cases, except $\wedge$R, the consequent formulae are placed directly below the target formula and are indented.

iv) The $\wedge$R rule is a special case. This rule is usually not applied in a tableau until all other possible rules are applied; also, the rule is applied to every conjunction at once. Instead of replicating the tableau, very cumbersome on paper, the following notation is used.

$$\neg\phi \text{ or } \neg\perp^{\ddagger} \text{ and } \psi^{\ddagger} \text{ or } \neg\perp \text{ and } \ldots \text{ and } \neg\phi^{\ddagger} \text{ or } \neg\perp^{\ddagger}$$

The zigzags separate a list of consequent possibilities resulting from the application of the $\wedge$R rule from the formulae in the semantic tableau. Reading the list left to right, the consequent possibilities are listed as they appear in the semantic tableau, reading the tableau from top to bottom. For every conjunction $\wedge\psi\chi$, the rule results in either ψ or χ on the right side; for multiple target conjunctions, the consequent possibilities are separated by the word "and". The application of the rule requires that one consequent possibility of each multiple target conjunctions is placed in the right column. If the choice of a possibility immediately results in a contradiction, a closed tableau, the symbol $\ddagger$ is placed by the formulae.[1] Those possibilities so indicated need not be considered. The other possibility, not creating a contradiction, must be selected and may cause a contradiction with a member of another pair of possibilities. Such contradictions are indicated by a footnote. If both possibilities produce a contradiction, the semantic tableaux is automatically closed; vide A2. Every attempt is made to show closure without replicating a semantic tableaux. When each case must be considered separately they are clearly labelled, for example, the proofs for axiom schemata A7 and A9.

[1] e.g., formula **T** in the right column and formula $\perp$ in the left column immediately result in a contradiction.

v) A closed semantic tableau is indicated by placing the symbol

within the tableau. If all semantic tableaux are closed, the construction for the axiom's conjugate is closed, and the axiom is valid

v) Each proof of validity is terminated with **Q.E.D.**

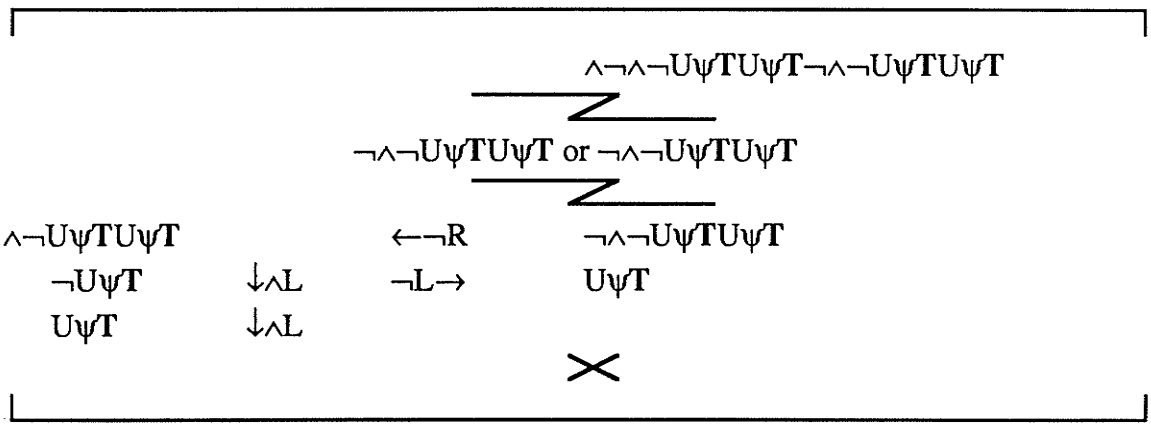
The first three proofs explicitly notate the application of the semantic tableaux.

Proof of validity for axiom A1. $\equiv\neg F\psi G\neg\psi$

Reduction sequence :

$\equiv\neg F\psi G\neg\psi$	
$\wedge\supset\neg F\psi G\neg\psi\supset G\neg\psi\ \neg F\psi$	; iii)
$\wedge\supset\neg F\psi\neg F\neg\neg\psi\supset\neg F\neg\neg\psi\neg F\psi$	; v) b)
$\wedge\supset\neg F\psi\neg F\psi\supset\neg F\psi\neg F\psi$	; vi)
$\wedge\neg\wedge\neg F\psi\neg\neg F\psi\neg\wedge\neg F\psi\neg\neg F\psi$	; ii)
$\wedge\neg\wedge\neg F\psi F\psi\neg\wedge\neg F\psi F\psi$	; vi)
$\wedge\neg\wedge\neg U\psi TU\psi T\neg\wedge\neg U\psi TU\psi T$	; iv) a)

Semantic tableaux procedure :



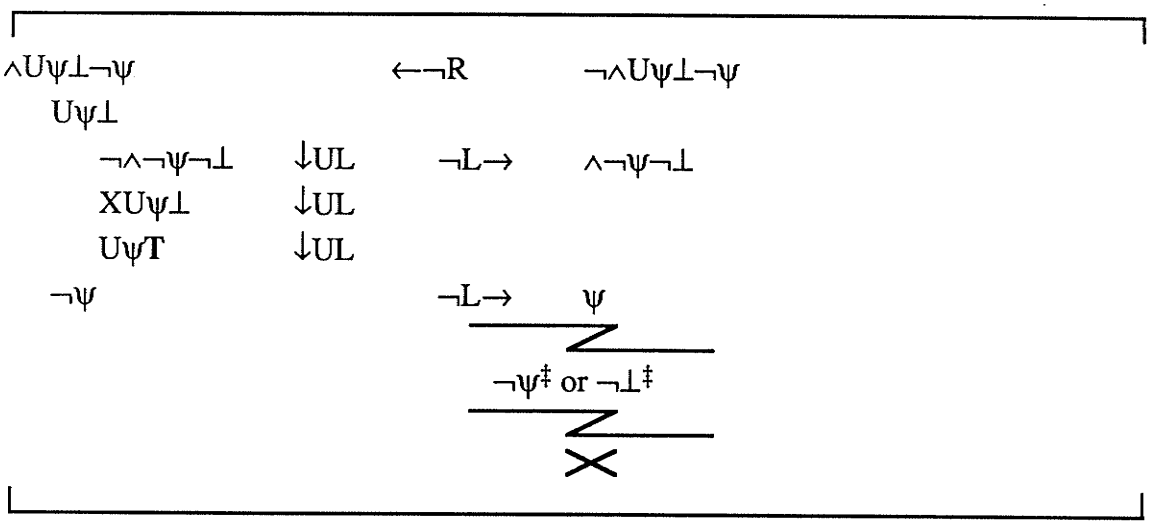
Q.E.D.

Proof of validity for axiom A2. $\supset G\psi\vee\psi$

Reduction sequence :

$\supset G\psi\vee\psi$	;
$\neg\wedge G\psi\neg\psi$	; ii)
$\neg\wedge U\psi\bot\neg\psi$	; v) a)

Semantic tableaux procedure :



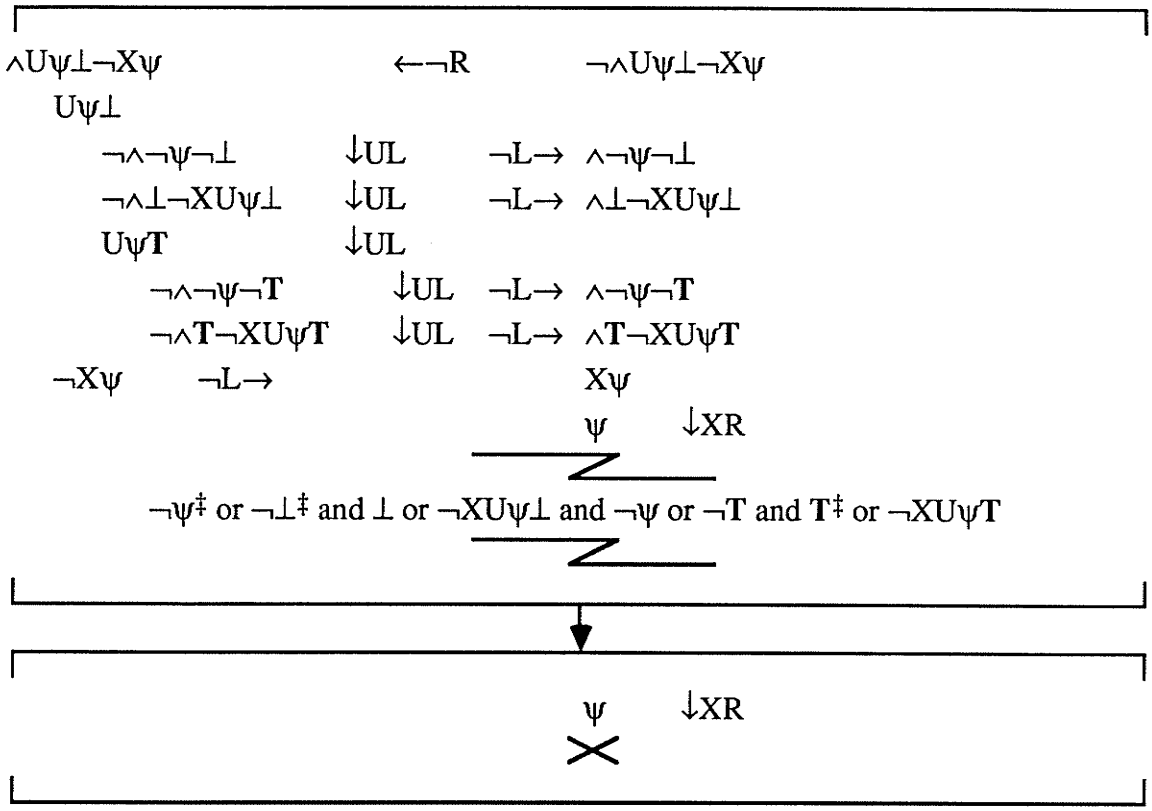
Q.E.D.

Proof of validity for axiom A3. $\supset G\psi X\psi$

Reduction sequence :

$\supset G\psi X\psi$	;
$\neg\wedge G\psi\neg X\psi$	; ii)
$\neg\wedge U\psi\perp\neg X\psi$	; v) a)

Semantic tableaux procedure :



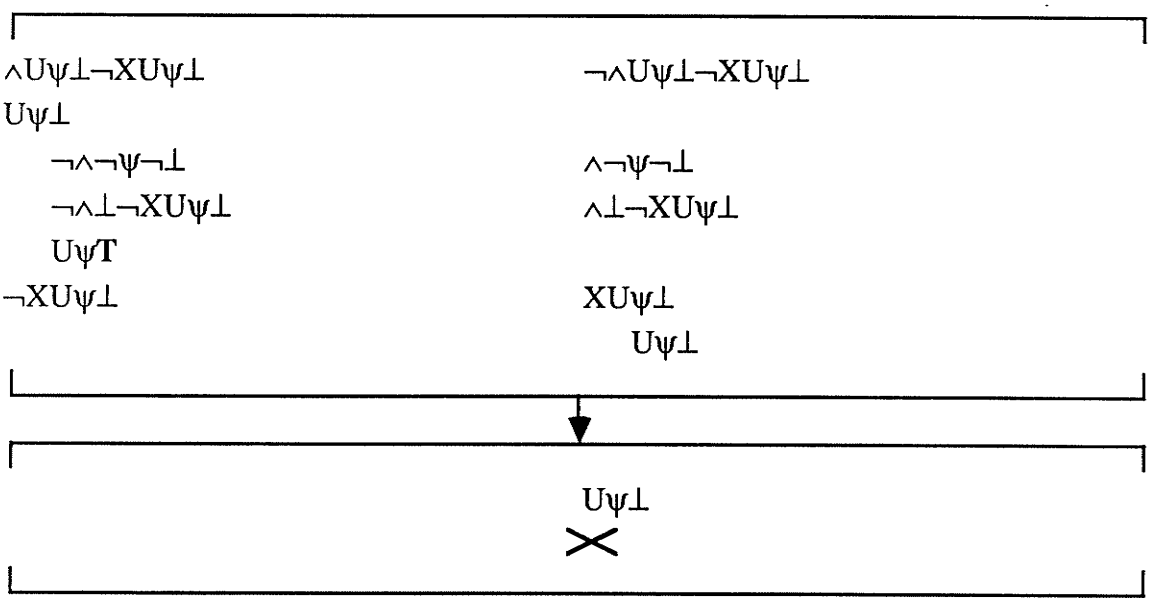
Q.E.D.

Proof of validity for axiom A4. $\supset G\psi XG\psi$

Reduction sequence :

$\supset G\psi XG\psi$	;
$\neg\wedge G\psi\neg XG\psi$	; ii)
$\neg\wedge U\psi\perp\neg XU\psi\perp$	; v) a)

Semantic tableaux procedure :



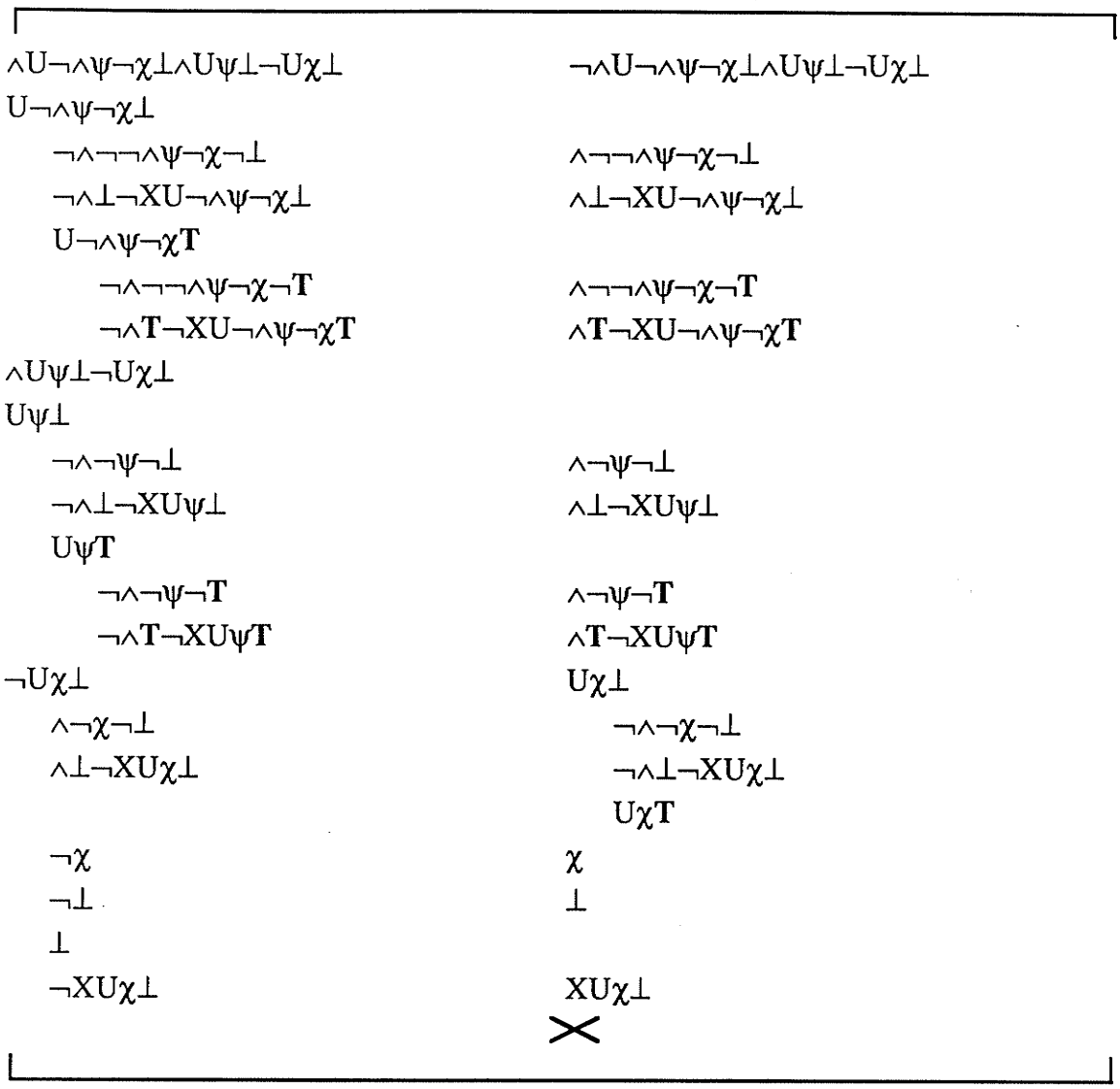
Q.E.D.

Proof of validity for axiom A5. $\supset G \supset \psi\chi \supset G\psi G\chi$

Reduction sequence :

$\supset G \supset \psi\chi \supset G\psi G\chi$	;
$\neg \wedge G \supset \psi\chi \neg \supset G\psi G\chi$	; ii)
$\neg \wedge G \neg \wedge \psi \neg \chi \neg \neg \wedge G\psi \neg G\chi$	; ii)
$\neg \wedge G \neg \wedge \psi \neg \chi \wedge G\psi \neg G\chi$	; vi)
$\neg \wedge U \neg \wedge \psi \neg \chi \perp \wedge U\psi \perp \neg U\chi \perp$	; v) a)

Semantic tableaux procedure :



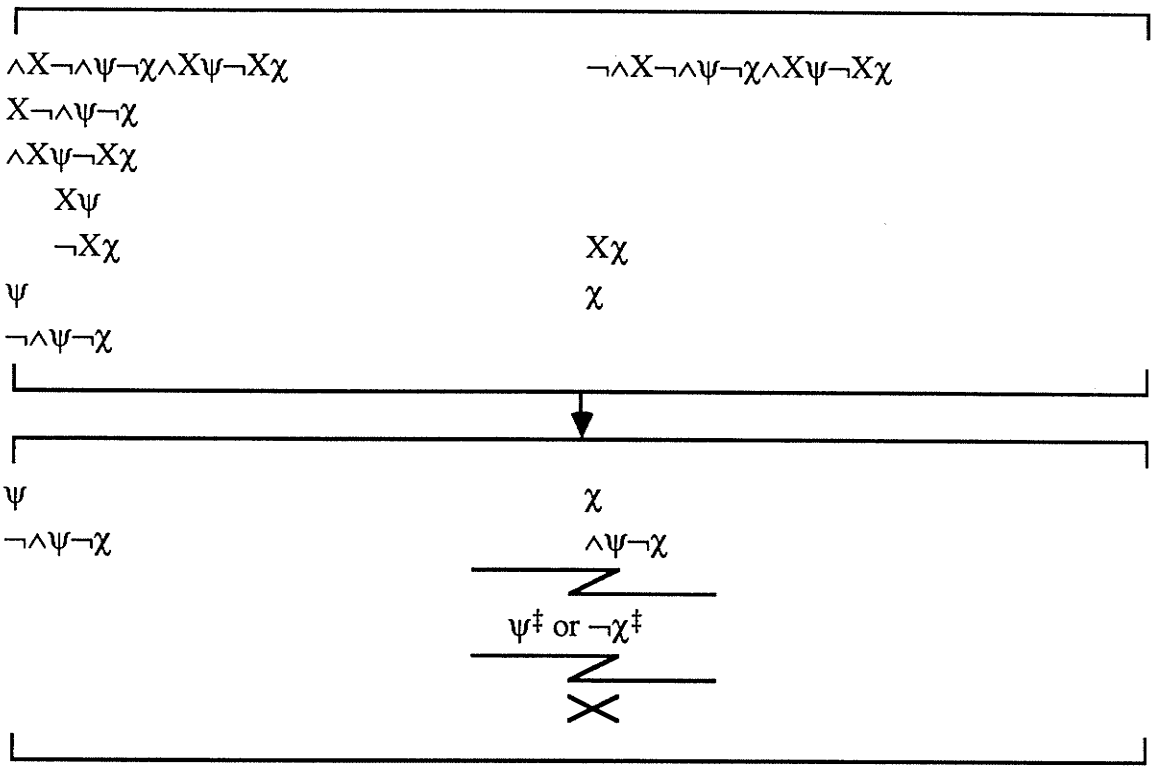
Q.E.D.

Proof of validity for axiom A6. $\supset X \supset \psi\chi \supset X\psi X\chi$

Reduction sequence :

$\supset X \supset \psi\chi \supset X\psi X\chi$	;
$\neg\wedge X \supset \psi\chi \neg\supset X\psi X\chi$	; ii)
$\neg\wedge X \neg\wedge\psi\neg\chi\neg\neg\wedge X\psi\neg X\chi$	; ii)
$\neg\wedge X \neg\wedge\psi\neg\chi\wedge X\psi\neg X\chi$	; vi)

Semantic tableaux procedure :



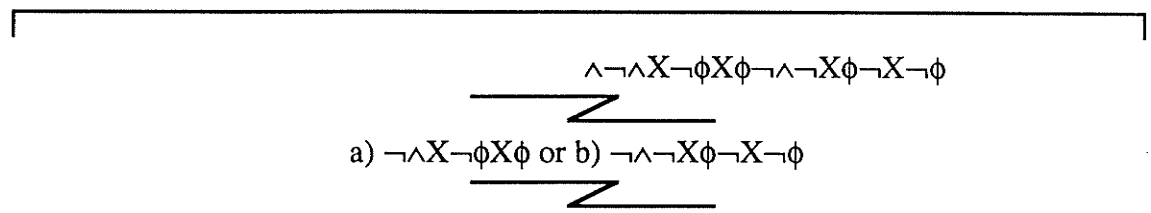
Q.E.D.

Proof of validity for axiom A7. ≡X¬ϕ¬Xϕ

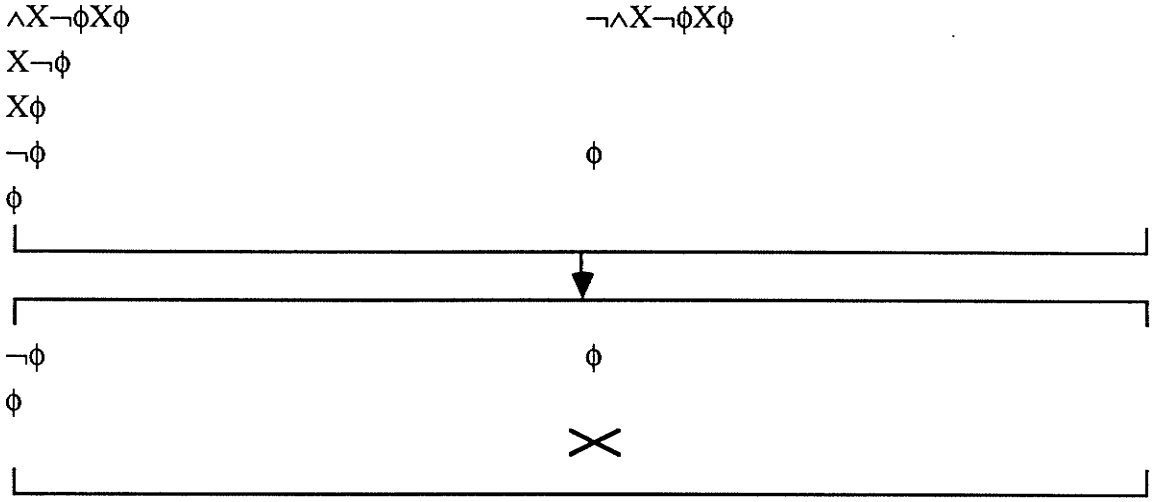
Reduction sequence :

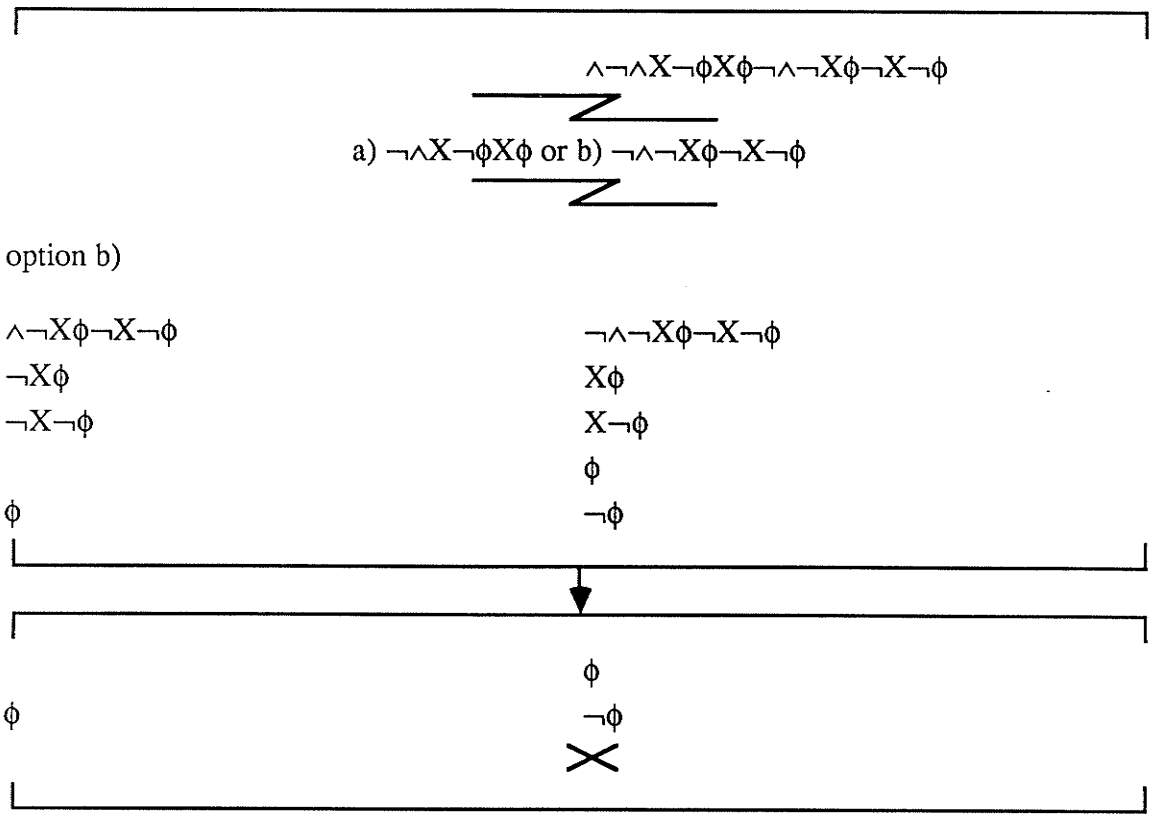
≡X¬ϕ¬Xϕ	;
∧⊃X¬ϕ¬Xϕ⊃¬XϕX¬ϕ	; iii)
∧¬∧X¬ϕ¬¬Xϕ¬∧¬Xϕ¬X¬ϕ	; ii)
∧¬∧X¬ϕXϕ¬∧¬Xϕ¬X¬ϕ	; vi)

Semantic tableaux procedure :



option a)





option b)

∧¬Xϕ¬X¬ϕ	¬∧¬Xϕ¬X¬ϕ
¬Xϕ	Xϕ
¬X¬ϕ	X¬ϕ
	ϕ
ϕ	¬ϕ

	ϕ
ϕ	¬ϕ

Q.E.D.

Proof of validity for axiom A8. $\supset G \supset \phi X \phi \supset \phi G \phi$

Reduction sequence :

$\supset G \supset \phi X \phi \supset \phi G$	;
$\neg \wedge G \supset \phi X \phi \neg \supset \phi G \phi$	; ii)
$\neg \wedge G \neg \wedge \phi \neg X \phi \neg \neg \wedge \phi \neg G \phi$	; ii)
$\neg \wedge G \neg \wedge \phi \neg X \phi \wedge \phi \neg G \phi$	; vi)
$\neg \wedge U \neg \wedge \phi \neg X \phi \bot \wedge \phi \neg U \phi \bot$	; v) a)

Semantic tableaux procedure :

$\wedge U \neg \wedge \phi \neg X \phi \bot \wedge \phi \neg U \phi \bot$	$\neg \wedge U \neg \wedge \phi \neg X \phi \bot \wedge \phi \neg U \phi \bot$
$U \neg \wedge \phi \neg X \phi \bot$	
$\neg \wedge \neg \neg \wedge \phi \neg X \phi \neg \bot$	$\wedge \neg \neg \wedge \phi \neg X \phi \neg \bot$
$\neg \wedge \bot \neg XU \neg \wedge \phi \neg X \phi \bot$	$\wedge \bot \neg XU \neg \wedge \phi \neg X \phi \bot$
$U \neg \wedge \phi \neg X \phi$ **T**	
$\wedge \phi \neg U \phi \bot$	
ϕ	
$\neg U \phi \bot$	$U \phi \bot$
$\wedge \neg \phi \neg \bot$	$\neg \wedge \neg \phi \neg \bot$
$\wedge \bot \neg XU \phi \bot$	$\neg \wedge \bot \neg XU \phi \bot$
	$U \phi$ **T**
$\neg \phi$	ϕ
$\neg \bot$	$\bot$
$\bot$	
$\neg XU \phi \bot$	$XU \phi \bot$
	✕

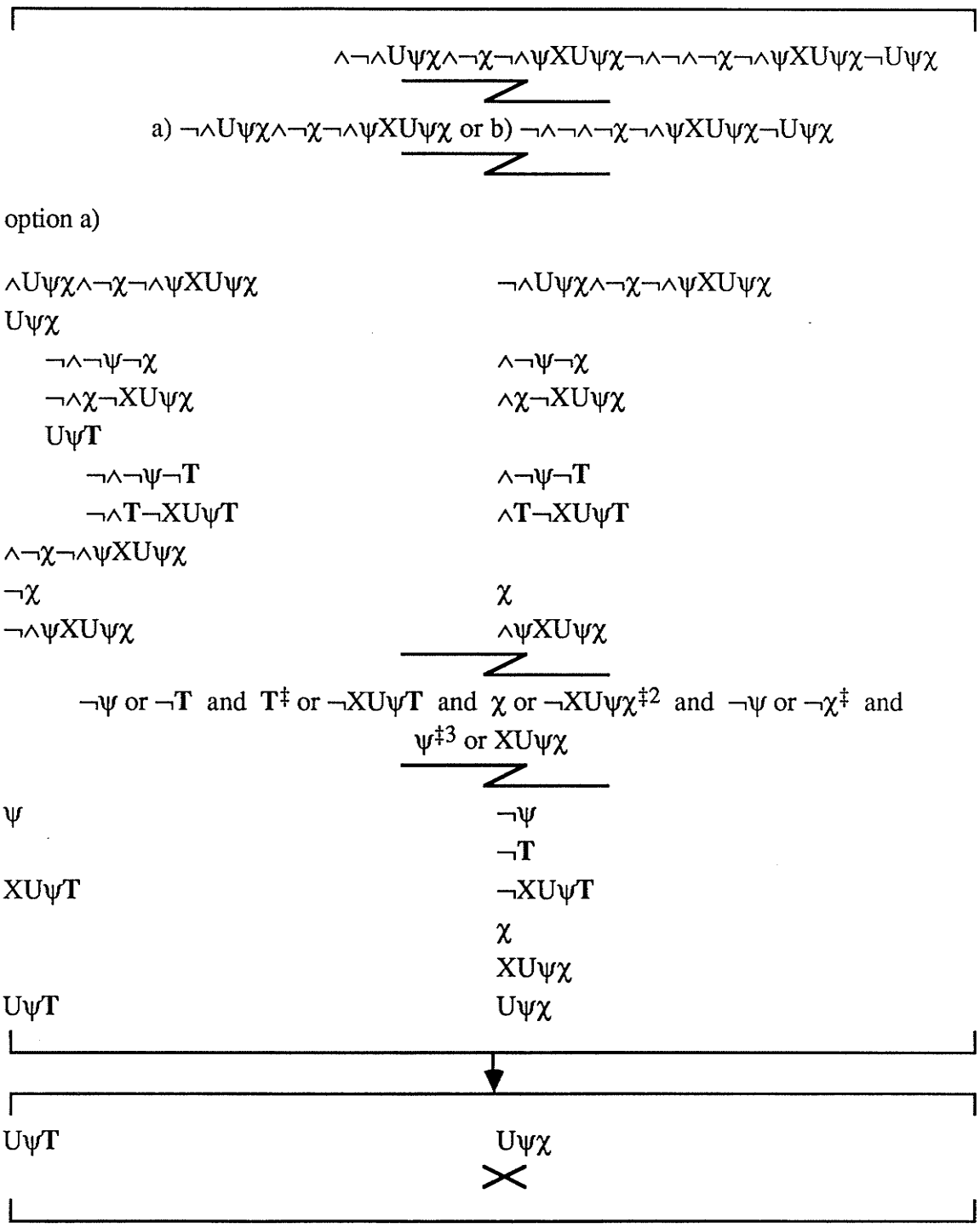
Q.E.D.

Proof of validity for axiom A9. ≡Uψχ∨χ∧ψXUψχ

Reduction sequence :

≡Uψχ∨χ∧ψXUψχ	;
∧⊃Uψχ∨χ∧ψXUψχ⊃∨χ∧ψXUψχUψχ	; iii)
∧¬∧Uψχ¬∨χ∧ψXUψχ¬∧∨χ∧ψXUψχ¬Uψχ	; ii)
∧¬∧Uψχ¬¬∧¬χ¬∧ψXUψχ¬∧¬∧¬χ¬∧ψXUψχ¬Uψχ	; i)
∧¬∧Uψχ∧¬χ¬∧ψXUψχ¬∧¬∧¬χ¬∧ψXUψχ¬Uψχ	; vi)

Semantic tableaux procedure :



2 because XUψχ must be on the right side due to foot note three.

3 because ¬ψ must be on the right side of the tableau.

$\wedge\neg\wedge U\psi\chi\wedge\neg\chi\neg\wedge\psi XU\psi\chi\neg\wedge\neg\wedge\neg\chi\neg\wedge\psi XU\psi\chi\neg U\psi\chi$

a) $\neg\wedge U\psi\chi\wedge\neg\chi\neg\wedge\psi XU\psi\chi$ or b) $\neg\wedge\neg\wedge\neg\chi\neg\wedge\psi XU\psi\chi\neg U\psi\chi$

option b)

$\wedge\neg\wedge\neg\chi\neg\wedge\psi XU\psi\chi\neg U\psi\chi$	$\neg\wedge\neg\wedge\neg\chi\neg\wedge\psi XU\psi\chi\neg U\psi\chi$
$\neg\wedge\neg\chi\neg\wedge\psi XU\psi\chi$	$\wedge\neg\chi\neg\wedge\psi XU\psi\chi$
$\neg U\psi\chi$	$U\psi\chi$
$\wedge\neg\psi\neg\chi$	$\neg\wedge\neg\psi\neg\chi$
$\wedge\chi\neg XU\psi\chi$	$\neg\wedge\chi\neg XU\psi\chi$
	$U\psi\mathbf{T}$
$\neg\psi$	ψ
$\neg\chi$	χ
χ	
$\neg XU\psi\chi$	$XU\psi\chi$

Q.E.D.

Proof of validity for axiom A10. $\supset U\psi\chi F\psi$

Reduction sequence :

$\supset U\psi\chi F\psi$	;
$\neg\wedge U\psi\chi\neg F\psi$	; ii)
$\neg\wedge U\psi\chi\neg U\psi\mathbf{T}$	; iv) a)

Semantic tableaux procedure :

$\wedge U\psi\chi\neg U\psi\mathbf{T}$	$\neg\wedge U\psi\chi\neg U\psi\mathbf{T}$
$U\psi\chi$	
$\neg\wedge\neg\psi\neg\chi$	$\wedge\neg\psi\neg\chi$
$\neg\wedge\chi\neg XU\psi\chi$	$\wedge\chi\neg XU\psi\chi$
$U\psi\mathbf{T}$	
$\neg U\psi\mathbf{T}$	$U\psi\mathbf{T}$
	$\times$

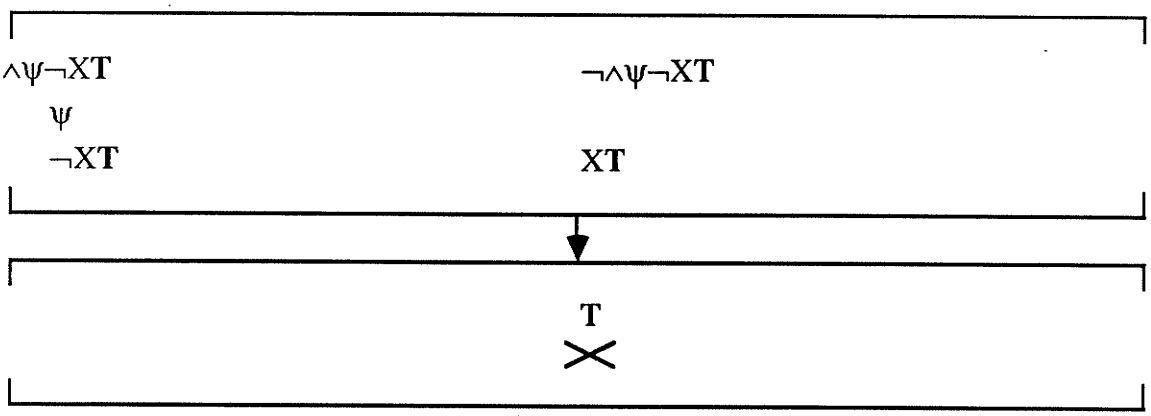
Q.E.D.

Proof of validity for A11. $\supset\psi\mathbf{XT}$

Reduction sequence :

$\supset\psi\mathbf{XT}$	;
$\neg\wedge\psi\neg\mathbf{XT}$	; ii)

Semantic tableaux procedure :



Q.E.D.

REFERENCES.

[AEIKOT] Anttila M., H. Eriksson, J. Ikonen, R. Kujansuu, L. Ojala H. Tuominen, "Tools and Studies of Formal Techniques - Petri Nets and Temporal Logic Protocol Specification, Testing, and Verification, III (H. Rudin and C.H. West eds.), Elsevier Science Publishers B.V. (North Holland), 1983.

[BB] Bolognesi T. and E. Brinksma, "Introduction to the ISO Specification Language LOTOS", not yet published.

[Ber] Berge C., Graphs, 2nd ed., North Holland : Amsterdam, New York, Oxford, 1985.

[BL] Bologna S. and N.G. Leveson, "Foreward : Reliability and Safety in Real-Time Systems", I.E.E.E. Transactions on Software Engineering, Vol. SE-12, No.9, September 1986.

[BMP] Ben-Ari M., Z. Manna, and A. Pnueli, "The Temporal Logic of Branching Time" Conference Record of the 12th Annual A.C.M. Symposium on Principles of Programming Languages, Williamsburg, V.A., January 26-28, 1981.

[BS] Bochmann G.V. and C.A. Sunshine, "Formal Methods in Communication Protocol Design", I.E.E.E. Transactions on Communications, Vol. Com-28, No. 4, April 1980.

[Bra] Brainerd W.S., "The Minimalization of Tree Automata", Information and Control 13, 1986.

[Bri] Bridge J., Beginning Model Theory, The Completeness Theorem and Some Consequences, Clarendon Press : Oxford, 1977.

[Bur] Burgess J. P., "Axioms for Tense Logic 1. 'Since' and 'Until' " Notre Dame Journal of Formal Logic, Vol. 23, No. 4, October 1982.

[CE] Clarke E.M. and E.A. Emerson, "Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic" Science of Computer Programming, Vol. 2, 1982.

[CES] Clarke E.M., E.A. Emerson, and A.P. Sistla, "Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach" Conference Record of the 10th Annual A.C.M. Symposium on Principles of Programming Languages, Austin, Texas, January 24-26, 1983.

[Cho] Choueka Y., "Theories of Automata on ω-Tapes: A Simplified Approach" Journal of Computer and System Sciences 8, 1974.

[CGL] Chow C.H., M. G. Gouda, and S.S. Lam, "A Discipline for Constructing Multiphase Communication Protocols" A.C.M. Transactions on Computer Systems, Vol. 3, No. 4, November 1985.

[CPW] Cohen B., D.H. Pitt, and J.C.P. Woodcock, "The Importance of Time in the Specification of O.S.I. Protocols: An Overview and Brief Survey of the Formalisms", National Physical Laboratory Report DITC 86 / 78, November 1986.

[Cop] Copi I.M., Symbolic Logic, 3rd ed., The MacMillan Co.: New York, Toronto, 1967.

[Cr-R] Crespi-Reghizzi S., "Petri Nets and Szilard Languages" Information and Control 33, 1977.

[Cur] Curry H. B., Foundations of Mathematical Logic, McGraw-Hill Book Co., 1963.

[CV] Carstensen H. and R. Valk, "Infinite Behaviour and Fairness in Petri Nets" Advances in Petri Nets 1984, Lecture Notes in Computer Science No. 188 (G. Rozenberg ed.), Springer Verlag : New York, 1985.

[Cyp] Cypser R.J., Communication Architecture for Distributed Systems, Addison-Wesley Publishing Co. : Reading, Menlo Park, London, Amsterdam, Don Mills, Sydney, 1978.

[Dal] Dalen D. van, Logic and Structure 2nd ed., Springer-Verlag : Berlin, 1983.

[Davis] Davis M., Computability and Unsolvability, Dover Publications, Inc. : New York, 1985.

[DGU] Diaz M., and G. Guidacci Da Silveira. "Specification and Validation of Protocols by Temporal Logic and Nets." Information Processing 83, (R.E.A. Mason ed.) Elsevier Science Publishers B.V. (North Holland), 1983.

[DHJKLLTW] Davies D.W., et al, Distributed Systems, Architecture and Implementation (Lampson D.W., M. Paul, and H.J. Siegart eds.), Springer Verlag : New York, 1981.

[DSB] Dubois M., Scheurich C., and F.A. Briggs, "Synchronization, Coherence, and Event Ordering in Multiprocessors", Computer, Vol.21, No.2, February 1988.

[EH82] Emerson E.A. and J.Y. Halpern, "Decision Procedures and Expressiveness in the Temporal Logic of Branching Time" Proceedings of the 14th Annual A.C.M. Symposium on the Theory of Computing, San Francisco, California, May 5-7, 1982.

[EH83] _____, "'Sometimes' and 'Not Never' Revisted: On Branching Versus Linear Time" Conference Record of the 10th Annual A.C.M. Symposium on Principles of Programming Languages, Austin, Texas, January 24-26, 1983.

[EL] Emerson E.A. and C.L. Lei, "Modalities for Model Checking: Branching Time Strikes Back" Conference Record of the 12th Annual A.C.M. Symposium on Principles of Programming Languages, New Orleans, Louisiana, June 14-16, 1985.

[Eme] Emerson E.A., "Automata, Tableaux and Temporal Logics", Conference Proceedings of 9th Annual Conference on Logics of Programs, Lecture Notes in Computer Science No. 193 (Goos G. and J. Hartmanis, eds.), Springer Verlag : New York, 1982.

[ES] Emerson E.A. and A.P. Sistla, "Deciding Branching Time Logic: A Triple Exponential Decision Procedure for CTL* " Logics of Programs, Lecture Notes in Computer Science No. 164 , Springer Verlag : New York, 19.

[Fine] Fine K., "Normal Forms in Modal Logic" Notre Dame Journal of Formal Logic, Vol. 16, No. 2, April 1975.

[GH] Guttag J. and J.J. Horning, "Formal Specification as a Design Tool" Conference Record of the Seventh Annual A.C.M. Symposium on Principles of Programming Languages, Los Vegas, Nevada, January 28-30, 1980.

[GP] Gajski G.D. and J-K Pier, "Essential Issues in Multiprocessor Systems", Computer, Vol. 18, NO. 6, June 1985.

[GPSS] Gabbay D., A. Pnueli, S. Shelah, and J. Stavi, "On the Temporal Analysis of Fairness" Conference Record of the Seventh Annual A.C.M. Symposium on Principles of Programming Languages, Los Vegas, Nevada, January 28-30, 1980.

[GS] Gurevich Y. and S. Shelah, "The Decision Problem for Branching Time Logic" The Journal of Symbolic Logic, Vol. 50, No. 3, September 1985.

[Hack] Hack M., Petri Net Languages, M.I.T Technical Report 159, 1976.

[Hai] Hailpern B. T., "Verifying Concurrent Processes Using Temporal Logic" Lecture Notes in Computer Science No. 129, Springer-Verlag: New York, 1982.

[Hal] Halmos P. R., Naive Set Theory, D. Van. Nostrand Co., Inc., 1960.

[HC] Hughes G.E. and M.J. Cresswell, An Introduction to Modal Logic, Methuen and Co. Ltd. : London, 1968.

[HO] Hailpern B.T. and S.S. Owicki, "Modular Verification of Computer Communication Protocols" I.E.E.E. Transactions on Communications, Vol. Com-31, No. 1, January 1983.

[Hoa69] Hoare C. A. R., "An Axiomatic Basis for Computer Programming" Communications of the A.C.M., Vol. 12 No. 10, October 1969.

[Hoa74] Hoare C. A. R., "Monitors : An Operating System Structuring Concept", Communications of the A.C.M., Vol. 17 No. 10, October 1974.

[Hoa78] _____, "Communicating Sequential Processes" Communications of the A.C.M., Vol. 21 No. 8, August 1978.

[HU] Hopcroft J. E., and J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley Publishing Co.: Sidney, Ontario, 1979.

[ISO7498] Information Processing Systems - Open Systems Interconnection - Basic Reference Model, 1st ed., ISO/TC 97/SC 21/7498-1984 (E), 1984.

[Kos] Kosaraju S.R., "Decidability of Reachability in Vector Addition Systems" Proceedings of the 14th Annual A.C.M. Symposuim on the Theory of Computing, San Francisco, California, May 5-7, 1982.

[Kri59] Kripke S.A., "A Completeness Theorem in Modal Logic" The Journal of Symbolic Logic, Vol. 24, No. 1, March 1959.

[Kri63] _____, "Semantical Analysis of Modal Logic I Normal Modal Propositional Calculi" Zeitschrift fur Mathematische Logik und Grundlagen dur Mathematik, Vol. 9, 1963.

[Lad] Ladner R.E., "The Computational Complexity of Provability in Systems of Modal Propositional Logic" Society for Industrial and Applied Mathematics Journal of Computing, Vol. 6, No. 3, September 1977.

[Lam78] Lamport L., "Time, Clocks, and the Ordering of Events in a Distributed System" Communications of the A.C.M., Vol. 21, No. 7, July 1978.

[Lam80] _____, "'Sometime' is Sometimes 'Not Never' On the Temporal Logic of Programs" Conference Record of the 7th Annual A.C.M. Symposium on Principles of Programming Languages, Los Vegas, Nevada, January 28-30, 1983.

[Lam83i] _____, "Reasoning About Nonatomic Operations" Conference Record of the 10th Annual A.C.M. Symposium on Principles of Programming Languages, Austin, Texas, January 24-26, 1983.

[Lam83ii] _____, "Specifying Concurrent Program Modules" A.C.M. Transactions on Programming Languages and Systems, Vol. 5, No., 2, April 1983.

[Lam83iii] _____, "What Good is Temporal Logic?" Information Processing 83 (R.E.A. Mason ed.), Elsevier Science Publishers B.V. (North Holland), 1983.

[Lam85i] _____, "What It Means for a Concurrent Program to Satisfy a Specification: Why No One Has Specified Priority" Conference Record of the 12th Annual A.C.M. Symposium on Principles of Programming Languages, New Orleans, Louisiana, January 14-16, 1985.

[Lam85ii] _____, "A Fast Mutual Exclusion Algorithm" Digital Systems Research Centre Report 7, November 14, 1985.

[Lam85iii] _____, "On Interprocess Communication" Digital Systems Research Centre Report 8, December 25, 1985.

[Lam86i] _____, "Control Predicates are Better Than Dummy Variables for Reasoning About Program Control" Digital Systems Research Centre Report 11, May 5, 1986.

[Lam86ii] _____, "A Simple Approach to Specifying Concurrent Systems" Digital Systems Research Centre Report 15, December 25, 1986.

[Lau] Laucht C.M. "Reachability Determination for a Non-Syntactic Subclass of Vector Replacement Systems (Petri Nets)" Ph.D. Thesis, University of Manitoba, 1982.

[LPS] Lehmann D., A. Pnueli, and J. Stavi, "Impartiality, Justice, and Fairness: the Ethics of Concurrent Termination", Automata, Languages, and Programming, Lecture Notes in Computer Science No. 115 (Even S. and O. Kavir eds.), Springer Verlag: New York, 1981.

[LP] Lichtenstein O., and A. Pnueli, "Checking that Finite State Concurrent Programs Satisfy Their Linear Specifications", Conference Procedings of the 12th Annual Symposium on the Principles of Programming Languages, New Orleans, Louisiana, Jan. 14-16, 1985.

[Mayr] Mayr E.W., "An Algorithm for the General Petri Net Reachability Problem" Society for Industrial and Applied Mathematics Journal of Computing, Vol. 13, No. 3, August 1984.

[McA] McArthur R. P., Tense Logic, D. Reidal Publishing Co. : Boston, 1976.

[McN] McNaughton R., "Testing and Generating Infinite Sequences by a Finite Automaton" Information and Control 9, 1966.

[Mil] Milner R., "A Calculus of Communicating Systems", Lecture Notes in Computer Science No. 92, Springer Verlag : New York , 1980.

[MP79] Manna Z., and A. Pnueli, "The Modal Logic of Programs" Proceedings of the 6th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science No. 71 , Springer Verlag: New York, 1979.

[MP81i] Manna Z., and A. Pnueli, "Verification of Concurrent Programs: the Temporal Framework", The Correctness Problem in Computer Science (Boyer R.S. and J.S. Moore, eds.), International Lecture Series in Computer Science, Academic Press : London, 1981.

[MP81ii] Manna Z., and A. Pnueli, "Verification of Concurrent Programs: Temporal Proof Principles" Lecture Notes in Computer Science No. 131, Springer Verlag: New York, Berlin, Heidelberg, 1981.

[MP83] Manna Z., and A. Pnueli, "Verification of Concurrent Programs: A Temporal Proof System" Foundations of Computer Science IV, Distributed Systems : Part 2, Semantics and Logic, (De Bakker J.W. and J. Van Leeuwen eds.) Mathematical Centre Tracts 159, Mathematical Centre : Amsterdam, 1983.

[Mou] Mountain M., The Zen Environment : The Impact of Zen, Bantam Books, Inc., 1983.

[O'Do] O'Donnell M. J., "A Critique of the Foundations of Hoare-Style Programming Logic" Communications of the A.C.M., Vol. 25, No. 12, Dec. 1982.

[Per] Perrin D., "An Introduction to Finite Automata on Infinite Words" Automata on Infinite Words, Lecture Notes in Computer Science No. 192, Springer Verlag: New York, 19.

[Peter77] Peterson J. L., "Petri Nets" Computing Surveys, Vol. 9, No. 3, September, 1977.

[Peter81] Peterson J. L., Petri Net Theory and the Modelling of Systems, Prentice-Hall, Inc. : Engelwood Cliffs, N. J., 1981.

[Petri77] Petri C.A., "Modelling as a Communication Discipline" Measuring, Modelling and Evaluating Computer Systems (Beilner H. and E. Gelenbe eds.), North-Holland Publishing Company, 1977.

[Piat] Piatkowski T.F., "The State of Art in Protocol Engineering", Proceedings from the SIGCOMM 1986 Symposium on Communications, Architectures, and Protocols, Stowe, Vermont, August 5, 6, and 7, 1986.

[Pnu] Pnueli A., "The Temporal Semantics on Concurrent Programs" Proceedings of the Symposium on Semantics of Concurrent Computations, Lecture Notes in Computer Science No. 70 (Even S. and O. Kavir eds.), Springer Verlag: New York, 1979.

[Pra] Prather R.E., Discrete Mathematical Structures for Computer Science, Houghton Mifflin Co. : Boston 1976.

[PZ] Pouzin L. and H. Zimmermann, "A Tutorial on Protocols" Proceedings of the I.E.E.E., Vol. 68, No. 11, November 1978.

[Rab] Rabin M.O., "Decidability of Second-Order Theories and Automata on Infinite Trees" Transactions of the American Mathematical Society, Vol. 141, 1969.

[Rob73] Robinson A., "Model Theory as a Framework for Algebra", Studies in Model Theory (Morley M. D. ed.), Studies in Mathematical Vol. 8, The Mathematical Association of America, 1973.

[Rob77] Robinson A., Complete Theories, 2nd. edition, North Holland : New York, 1977.

[RS] Rabin M.O. and D. Scott, "Finite Automata and Their Decision Problems" I.B.M. Journal of Research and Development, No. 3, 1959.

[RU] Rescher N. and A. Urquhart, Temporal Logic, Springer Verlag : New York, 1971.

[Rudin] Rudin H., "An Informal Overview of Formal Protocol Specification" I.E.E.E. Communications Magazine, Vol. 23, No. 3, March 1985.

[Sac] Sacks G.E., Saturated Model Theory, W.A. Benjamin, Inc. : Reading, Massachusetts, 1972.

[SC] Sistla A.P. and E.M. Clark, "The Complexity of Propositional Linear Temporal Logics" Proceedings of the 14th Annual A.C.M. Symposium on the Theory of Computing, San Francisco, California, May 5-7, 1982.

[Sis] Sistla A.P., "Theoretical Issues in the Design and Verification of Distributed Systems", Ph.D. Thesis., Carnegie-Mellon University, August 1983.

[SM-S] Schwartz R. L. and P.M. Melliar-Smith. "Temporal Logic Specification of Distributed Systems", Conference Proceedings of the Second Annual International Conference on Distributed Systems, Paris, France, April 8-10,1981.

[Smu] Smullyan, First Order Logic, Springer Verlag : New York, 1968.

[Sta] Stallings W., Data and Computer Communications, MacMillan Publishing Co. : New York, 1985.

[Tan] Tanenbaum A.S., Computer Networks Prentice-Hall Inc. : Englewood Cliffs, New Jersey, 1981.

[Tap] Tapscott B. L., "Correcting the Tableau Procedure for S4" Notre Dame Journal of Formal Logic, Vol. 25, No. 3, July 1984.

[Tar] Tarski A., Ordinal Algebras, North-Holland Publishing Company : Amsterdam, 1956.

[Wec] Wecker S., "Computer Network Architectures" Computer, September 1979.

[Wes] West C.H., "Protocol Validation by Random State Exploration" Protocol Specification, Testing, and Verification, VI (Sarikaya B. and G.V. Bochmann), North-Holland Publishing Company : Amsterdam, New York, Oxford, and Tokyo, 1987.

[Wol] Wolper P. "Specification and Synthesis of Communicating Processes Using An Extended Temporal Logic" Conference Record of the 9th Annual A.C.M. Symposium on Principles of Programming Languages, Albuquerque, New Mexico, January 25-27, 1982.