National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

NRS APPROACH TO SEMANTICS OF SPECIFICATION AND TESTING OF DISTRIBUTED SYSTEMS

By

Linsheng Wei

Thesis

Submitted to the School of Graduate Studies
and Research

in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy
in
Computer Science
under the auspices of the Ottawa-Carleton Institute
for Computer Science

University of Ottawa

May 1995

ISBN 0-612-07811-6

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Acknowledgements

I would like first of all to express my deepest gratitude to my supervisor Professor Robert L. Probert, for his guidance, encouragement, multiple suggestions, and moral support in my difficult moments. Whatever research methods and results I may have acquired in these years, I feel I owe it in large part to him.

I am very grateful to Dr. L. Logrippo for his willingness to listen and comment on my work and for giving me helpful and inspiring advice.

I would like to thank Professor M. Hennessy for his book "Algebraic Theory of Processes" and very helpful comments on the preliminary version of this thesis.

I would also like to thank many people for their help and suggestions in various stages of my study: Dr. H. Ural, Dr. To-yat Cheung and Dr. S. Das among others. Many thanks to Mrs. Louise Desrochers for her regular help in my everyday student work.

My deepest respect and love to my father Tianlei Wei who always inspires me with courage in my study and life, and was eager to see but could not wait until the finish of my education. I owe my mother, sister, and brothers a great deal for not requiring me to take care of them while pursuing my studies. Special thanks go to my wife Lanying Wei for her support and patience which were very valuable for finishing this work, and to my daughter Jiaqi who missed many entertaining programs to provide me a quiet environment and enough time for my study.

Finally, I gratefully acknowledge support from the Telecommunications Research Institute of Ontario (TRIO), the Natural Sciences and Engineering Research Council (NSERC) of Canada, and the University of Ottawa.

Abstract

In this thesis, we study a new semantic technique called *NRS approach* for modeling the behaviors of nondeterministic processes in distributed systems. The *basic principle* of this approach takes into account the abilities of each of the process and its environment to control execution behaviors of the whole system. System behaviors are therefore seen as the result of mutual influences of the process control and its environment control. The original idea comes from our experience in analyzing specifications for test generation and observing behaviors of processes under test, particularly telecommunications systems. The main objective of capturing this idea in formal semantics is to try to improve the applicability of formal semantics to the practical development of distributed systems.

First, based on labelled transition systems, the process and environment controls are formalized as two formal objects: the *nondeterministic rippling effect* inside the process and the *choice pattern* enforced by the environment. By considering the mutual influences of these two objects, a semantic object denoted *nondeterministic ripple set* (NRS), is defined. Then, by means of synchronization trees, NRS is applied to a subset of CCS. As a semantic object, NRS can be used to describe various properties of processes.

Next, based on NRS, we study a new equivalence between processes called *nondeterministic ripple equivalence* . Two processes are nondeterministic-ripple equivalent if they have the same set of nondeterministic ripple sets. An axiom system is provided for this equivalence, and its soundness and completeness are proved. Two auxiliary operators are introduced to model both process control and environment control algebraically. The algebraic representation of NRS is derived. This work provides a calculus for reasoning about process properties based on NRS.

We then develop an application of NRS to process testing by formalizing a theory of testing. Distinct from other testing theories, we define tests based on the process under test such that each test can be related to a test purpose by means of the environment control A testing preorder/equivalence called *nondeterministic ripple acceptance testing* is defined. It is proved that this testing preorder/equivalence possesses the same distinguishing power between processes as failure equivalence.

Comparison of the NRS approach with other major semantic theories is discussed. It appears that NRS semantic interpretation is intuitively simple and practical for interpreting process behaviours in a distributed system. As for the distinguishing power of the nondeterministic ripple equivalence, it is proved that nondeterministic ripple equivalence is different from bisimulation equivalence, ready trace equivalence, refusal testing equivalence and readiness equivalence. Nondeterministic ripple equivalence is also proved to be different from failure equvalence, but we conjecture it implies failure equivalence. Extension of the NRS approach to recursive processes with internal actions is also briefly outlined. Finally, the contributions and practical applications of the NRS approach, in particular to conformance testing, are discussed.

The application of the NRS semantic approach to conformance testing appears promising. This brings us back to our original inspiration from testing for NRS semantics, and lends support to the further development of a pragmatic formal semantics for testing processes in distributed systems.

Table of Contents

Symbols and Notations

(frequently used)

Abbreviations

LTS	Labelled Transition System
ST	Synchronization Tree
NRS	Nondeterministic Ripple Sets, or characterization set of a process

Symbols

$\in$	is a member of
$\cup$	set union
$\subseteq$	set containment
$\neg$	logical negation
$\Rightarrow$	logical implication
$\mathfrak{U}$	pointwise union of sets
$\sum$	summation of synchronization trees
$\|$	interaction of process and test
ε	empty string
.	action prefixing operator of processes, or action prefix to a set
+	summation operator of processes
$\oplus$	operator of processes for representing process control
$\pm$	operator of processes for representing environment control
rec x._	operator for defining recursive processes
τ	the invisible action
σ	mark of ending of a complete trace and report of success in testing

Notations

Act	a predefined set of actions ranged over by a
NRS(P)	nondeterministic ripple sets of a process P
NRT(P)	deterministic tree representation of NRS(P)
nrs	a single nondeterministic ripple set in NRS(P)
nrt	an element in NRT(P)
CS(P)	the set of choice patterns of a process P

NP(a)	nondeterministic part of a ST with root branches labelled with a
$T(P)$	the set of tests defined for a process P
e	an experiment or a test in $T(P)$
ST	domain of synchronization trees
$=_{NR}$	nondeterministic ripple equivalence
$=_{nra}$	nondeterministic ripple acceptance testing equivalence

(following notations defined in chapters 3 and 5, respectively)

$\text{---}a\rightarrow$	performing an a action
$\text{---}s\rightarrow$	performing a sequence of actions s
$=s\Rightarrow$	performing a sequence of actions, possibly interspersed by internal actions
$L(p) = \{s \mid p \text{---}s\rightarrow\}$	the *language* or the *set of traces* of p
$D(p, s) = \{q \mid p \text{---}s\rightarrow q\}$	the *s-ripples* or *derivatives of p.*
$S(p) = \{a \mid p \text{---}a\rightarrow\}$	the *successors* of p, if p is a leaf node, $S(p) = \{\sigma\}$
$A(p, s) = \{S(q) \mid q \in D(p, s)\}$	the *acceptance set of p after s*
$XA(p, s) = \times_{S_i \in A(p, s)} S_i,$	the *cartesian product of elements in $A(p, s)$*
$CA(p, s) = \{G(x) \mid x \in XA(p, s)\}$	*transform each tuple in $XA(p, s)$ into a set*
$S(p, s) = \cup_{q \in D(p, s)} S(q),$	the *successors* of p *after s*
$CT(P) = \{s \mid r \text{---}s\rightarrow q$ and $\neg(\exists a \in Act, q \text{---}a\rightarrow)\}$	the set of *complele traces* of P
$S(dt(s)) = \{a \mid dt(s) \text{---}a\rightarrow\}$	the *successors* of a deterministic process dt at the node labelled by s

Chapter 1

Introduction

This chapter sets the context for the research of this thesis. We first state the problem to be solved. Then, we review previous studies on formal semantics of distributed systems. Finally, we outline the contributions and organization of this thesis.

1.1 Statement of Problem

The development of distributed computing systems can be viewed as a series of transformations of system descriptions, called *specifications*, from an initial specification of requirements through successively more complete designs to a final executable specification called the implementation. During this process, the intermediate system descriptions are specifications of different conceptual levels of the same system, and are visually manipulated to serve as transitional interfaces between different development phases or contract parties. In this scenario, the question arises whether two specifications produced by two different development parties describe the same system:

(1) *Do the two descriptions specify the same system* ?

Or, within the same development phase, when one of the descriptions, S_1, is viewed as an initial specification and the other, S_2, is viewed as an refinement of S_1, the question becomes

(1.1) Is S_2 a correct (or conforming) refinement of the specification S_1?

One specific example of the above general problem (1) occurs in the development of communicating systems. In this context, the natural question is:

(2) *Can two communicating systems based on the same (standard) specification communicate with each other?*

In order to communicate with each other in such systems, it is required that conforming implementations of the same communicating requirement(the same communicating software specification or protocol) are placed into each communicating entity. Thus, two communicating systems can communicate with each other if they have conforming implementations of the same requirements specification (protocol). If we denote a requirements specification as S_1 and its implementation in each communicating system as S_2, Question (2) becomes:

(2.1) Is S_2 a conforming implementation of S_1?

Questions (1) and (2) are directly derived from the real world. In general, they can be abstracted as a formal equivalence problem:

(3) *Given two specifications S_1 and S_2, decide if S_1 and S_2 are equivalent, written as $S_1 \approx S_2$.*

In most cases, the exact equivalence between specifications is difficult or undesirable due to physical restrictions. The equivalence problem is often weakened as extension or reduction between specifications. This can be formalized by a partial order(or preorder) relation between specifications as:

(3.1) Given two specifications S_1 and S_2, decide if S_2 is a reduction(or extension) of S_1, written as $S_1 \leq S_2$.

In a development environment, if a *precise semantics*: f is available as an underlying mathematical interpretation of the *specification language*, the equivalence problem can be stated as:

(4) $S_1 \approx (\leq) S_2$ if $f(S_1) = (\leq) f(S_2)$.

where $f(S)$ stands for the semantic interpretation of specification S.

In software theory and engineering, solving question (4) during a development process is called *validation*. Furthermore, the determination of question (1) is referred to as *verification* where two written specifications are involved. Finally, the determination of question (2) is referred to as *testing* where the implementation is seen as a black box. Test cases are derived from its specification and applied to the black box through a interface. Then, the observed behaviors of the implementation are compared with the expected ones from the specification to decide conformance. For validation of communicating systems, establishing a verdict for question (2) is also called *conformance testing* [CTFM, Ray 87].

Formal methods [BS 80, BS 83, Mcl 84, San 88] have been suggested as an effective way to model the process of development of distributed systems. This approach consists of a formal description technique/language and a formal semantics for interpreting the specifications described by the language.

Significant improvements have been made to description techniques of distributed/communicating systems. FSM [Lee 78, Koh 78], CCS [Mil 80] and CSP [Hoa 85] description models have been studied from various aspects, and based on them, several description languages: SDL [CCITT 87, ST 87], Estelle [BD 87], Lotos [BB 87, EVD 89] and TTCN [MoPr 92] have been standardized by ISO and CCITT as formal description techniques.

However, improvements on studies of formal semantics of distributed systems are not so great. A satisfactory semantics should assist in the practical activities such as property reasoning, design, verification and testing in development of systems . Especially in the case of testing, the semantics should support *test generation, test selection, and test*

specification. This is still not the case in the *practical* application of formal semantics to development of distributed systems.

The reason for this situation is that due to particular features of distributed systems such as *reactivity, nondeterminism and concurrency*, unlike sequential systems, many variations exist for the understanding and definition of system behaviors. Studies have been active in recent years to search for a solution to this problem. The main objective of this thesis is to address the semantic and validation problems described above by designing and studying a new semantic technique: *NRS approach* for modelling process behaviors of distributed systems.

1.2 Review of Semantic Techniques for Distributed Systems

Previous studies on the semantics of distributed systems are reviewed in this section. First, we briefly outline some semantic knowledge in general. Then we list and evaluate major semantic techniques used for modeling processes in distributed systems.

1.2.1 Semantics Background

Given a description language, its semantics concerns the interpretation (meaning) of specifications (programs) written in this language. This is usually done by translating the program of the language into another notation(semantic object) which abstracts away the *pragmatic aspects* of the language while keeping the *information* one thinks to be *important* about the system specified in the language. A semantics is *formal* if it is designed based on a precisely defined mathematical concept [Sto 77, Gue 81, Hen 88]. A specific semantics(or a semantic model) induces equivalences or orders among programs. Ideally, a properly designed semantics of a description language would produce a method which

assists in specifying, designing and validating the system described by the language. The recent trend of developments in this field has been toward applying formal specifications, formal semantics, and formal validation techniques to system development, i.e., *formal methods* in general [San 88].

Semantics of traditional programming languages was defined by means of mathematical functions, which describe the behavior of a program as a function from its input to output states [Gor 79]. However, it has become common understanding that this approach is not adequate for communicating programs(reactive systems, protocols). Because of the distinct features in these systems such as *reactive(interactive), nondeterminism and concurrency* the intermediate states of a program/system cannot be overlooked, since they are directly related to the behaviors of the surrounding environment [Cas 88]. Properties like deadlock and liveness cannot be analyzed by considering only two snapshots of the program. Moreover, the final or output state in a communicating system is not precise, since any state may be considered as an output state due to its interactive nature.

Therefore, much effort has been devoted to finding new theories to define semantics of distributed systems. *Powerdomains* [Plo 76, Smy 78, Abr 83] and *Labelled Transition Systems* [Plo 81, Mil 88] are the recent developments in this respect. Actually, they serve as extensions of the theories of traditional (functional) denotational and (finite state machine) operational semantics in the cases of distributed systems. *Process algebra* [GTW 78, Mil 80, Mil 83, AB 84, Mil 88, Hen 88] is another major development. This approach involves the use of algebraic methods to specify processes and define relationships between processes. Many semantic models have been proposed (as we surveyed below) based on process algebra.

1.2.2 Semantic Techniques for Distributed Systems.

Many semantic models have appeared in the literature based on different *semantic techniques* or *behavioral views* of system operations. In this section, we review the major semantic techniques for distributed systems by which most of the semantic models are formalized to provide a better coverage of this development. These semantic techniques might be classified as: Labelled Transition Systems, Bisimulation, and Refusal/Acceptance of actions. We also classify the testing approach as a technique, because it is a new view to define system behaviors and directly addresses validation problems in communicating systems. Related issues in semantic definitions such as internal actions, divergence, interleaving, and true concurrency are also discussed.

(1) Labelled Transition Systems

Labelled Transition Systems(LTS) were first proposed as a general model of computation by Keller [Kel 76] and then, have been extensively used for describing operational behaviors of concurrent programs [Plo 81, Mil 88]. LTS describes processes as evolving through states via successive transitions. Each transition is labelled by an action which may represent an interaction with the environment or an internal computational step. Formally, a LTS is defined as: a quadruple $<P, r, Act, \longrightarrow>$, where P is a set of states; $r \in P$ is the initial state; Act is a set of actions (or labels); and $\longrightarrow \subseteq P \times Act \times P$ is the transition relation. A transition $(p, a, p') \in \longrightarrow$ is usually represented as $p \xrightarrow{a} p'$.

LTS is a convenient basic model for characterizing on-going behaviors of concurrent programs. Actually, most recent studies in concurrency, starting with Milner's Calculus of Communicating Systems(CCS) [Mil 80] have been based on LTS as an operational interpretation of concurrent programs. Because of its simple and general definition, LTS is

widely used in semantics of distributed systems. They provide an operational *semantic universe* for defining logic and denotational semantics. By varying the structures of transitions and states, LTS can be used for different levels of abstract description. Examples include Milner's weak transitions for CCS [Mil 88], distributed transitions [Cas 88], as well as pomset transitions for so-called true concurrent models of processes [BC 86, Cas 88, Win 87].

However, labelled transition systems as an operational model of distributed systems are not entirely adequate for describing system properties. It has been recognized early that this approach does not abstract away sufficiently from unwanted details in system specifications and yield intentional accounts of processes [BN 92]. Moreover, LTS alone does not induce equivalence relations between specifications, and needs to be augmented by other semantic approaches such as bisimulation [Mil 80] and failure equivalence [BHR 84]. As such, labelled transition systems cannot directly provide methods for verification and testing in development of distributed systems. They are used as basic operational semantic definitions to assist in studies of process properties in other semantic approaches.

(2) Bisimulation

Bisimulation has been used as a major semantic technique to define semantics of processes specified by algebraic languages such as CCS [Mil 80]. Pioneering work of using an algebraic approach to the theory of communicating systems was due to R. Milner. His work led directly to CCS which consists of an algebraic language for processes, a *behavioral equivalence* based on an operational semantics(LTS), and a calculus for reasoning about processes based on a set of equations. This behavioral equivalence was first called Milner's *observational equivalence* [Mil 80, HM 80, HM85] and then replaced by *bisimulation equivalence* defined by Park [Par 81] simply because of its better

mathematical properties. The two equivalences coincide on a large class of processes and now, the two names are often used interchangeably.

Bisimulation equates two processes if they can bi-simulate each other's transitions at all corresponding pairs of states based on the LTS semantics of processes. The theory of bisimulation is well investigated from many aspects such as algebra [Mil 88, HM85, Jon 82], logics [Bro 83, HM85, Str 87, Chr 88], equation axiomatization [Str 85, Dar 82] so that it has been considered as a fundamental notion of concurrent systems. Process description languages such as CCS and many of its variants have been thoroughly studied using equivalence notions based on bisimulation. It may be said that approaches used in this theory have given much guidance to other semantic studies of distributed systems.

However, in spite of the well developed theory of bisimulation, its application for verification and testing of distributed systems appears limited. This is mainly because its distinguishing power between processes is too strong to be realized practically, that is, it goes beyond those distinctions that can really be made by an observer [Abr87, CH 93]. Bisimulation was studied from testing point of view in [Abr87, CH 93]. It was proved that testing for bisimulation needs copying and global testing mechanism. This means that an inside state of a process should be copied for repeated testing, which is not possible at the state of art. To some extent, bisimulation technique is still an intentional approach. It is difficult to design a denotational semantics for bisimulation based on a model-theoretic approach [Hen 85], although an algebraic characterization is given in [HM 85].

(3) Refusal and Acceptance of Actions

The technique for modelling process behaviors in this approach is based on the interactions between a process and its environment. The process and its environment (observer) are

imagined to interact through an interaction point. The observer offers actions to the process and records his observations from the process as the process's behaviors [RB 81, OH 86]. In general, a process behavior is recorded as:

A sequence of actions accepted by the process (s)

+ process behavioral expectation after the sequence (X)

This is usually represented as <*s*, X>. The process behavioral expectation X is formalized as a set which represents the set of actions which may be refused or accepted at the present state of the process. Refusal set [BHR 84], Ready set [OH 86], and Acceptance set [Hen 88] are instances of expectation X, for example.

Failure equivalence, where X is called failure sets [Hoa 85, BHR 84], and testing equivalence/acceptance trees, where X is called acceptance sets [Hen 85, Hen 88] are well-known models in this class.

By assigning the set X different contents, different semantic interpretations of processes can be formed. The traditional trace equivalence for automata theory [HU 79] may be classified into this class by making X empty. These models consider a trace and the expected behaviors after the trace. By extension, one may consider the refusals or ready-set after each single action of a trace of the process. This results in stronger models such as failure-trace [Lan 89] and ready-trace [BBK 87] models. Other similar models in this class include [BMO 84, GB 87, TV 87, DG 87].

Merits of the semantic models in this class are that they admit set-theoretic model for denotational interpretations of a process, and most of the semantic denotations have intuitive representations such as acceptance trees. As a, models here are considered more extensional or observational in describing process properties than models based on

bisimulation and LTS. Failure equivalence lies strictly between trace and bisimulation equivalences and it appears to be the weakest tractable equivalence which respects the possibility of deadlock. Application of failure equivalence to validation of distributed systems has been studied by means of testing [Bri 88] and the results are shown to be better than those of other semantic models.

However, the distinguishing ability of failure equivalence may be weak because some processes with intuitively non-equivalent behaviors can be equated by failure equivalence [Lan 89].

(4) Testing Approach

This semantic technique defines equivalence between processes by testing. The idea is that two processes are equivalent if they exhibit the same pattern of success when subjected to all possible tests. A process here is understood as a black box(implementation of the process) with interface on which its environment(observer) performs experiments with tests.

The notion of *Testing Equivalence* was first developed by Hennessy and de Nicola [NH 83, Hen 85]. In their work, tests are described by CCS processes with a distinguished action ω to report success. During a testing experiment, the process under test and a testing process are synchronized to progress until first deadlock occurs. This testing run successes or fails depending on whether or not the action ω is encountered. A denotational model has been derived from the testing equivalence called Acceptance Trees [Hen 85, Hen 88]. It was proved that the distinguishing ability of this equivalence coincides with the failure equivalence for strongly convergent processes [Nic 87]. A strongly convergent process is a process which does not have infinite internal computations.

Another work [Phi 85] slightly modifies Hennessy and de Nicola framework by defining a set of different tests and allowing the course of an experiment to progress beyond the deadlock. It results in a new equivalence called *Refusal Testing* whose distinguishing ability is stronger than *Testing Equivalence*. Furthermore, in [Abr 87], much more complicated testing processes are allowed such as copying and global testing, and the bisimulation equivalence is shown to be characterized by this notion of testing. Recently, the relation between testing equivalence and bisimulation equivalence is further discussed in [CH 93] from the point of view of transformations of transition systems.

Other developments have tried to directly apply the notion of testing to conformance testing of distributed systems in conjunction with the study of specification language Lotos(CCS-based) [Bri 88, BSS 87, GuLo 88, BALL 89] and Estelle(FSM-based) [Pit 87, CVI 89]. A major representative of these methods is based on experiments on Finite State Machines(FSM) [Lee 78, ADLU 88, SL 86, SL 88, SD 85]. For example, *Distinguishing Sequence* is used to verify each state and transition in a FSM. The semantics used in this method is the trace equivalence for deterministic FSM. Another major representative is *Canonical tester* for nondeterministic processes which was proposed by Brinksma [Bri 88, Wez 89]. This work tries to derive a canonical test suite from a specification to test its implementations. The equivalence/preorder in the underlying semantics of this method is called **red** which is a similar version of failure equivalence [Bri 88]. However, the conformance relation enforced by the canonical tester is **conf** which is not a pre-order.

Defining equivalence by testing is a totally different view to semantic approaches of distributed systems. Studies of this approach have contributed to defining many critical concepts of testing. However, the whole development of this research is still in its early stage. Some important testing concepts proposed in this approach appear theoretical-

oriented. For example, the set of tests (test suite) is specified by an algebraic language, and tests applied to the process under test are considered to be all terms generated by this language. It seems unpractical.

Methods oriented towards conformance testing of communicating systems do provide ways to generate tests from a given specification. However, the approach represented by experiments on FSM only handles the deterministic machines. Canonical tester is a better example of applying formal semantic theory to conformance testing. But a canonical tester is a single process. Testing effectiveness in its application is questioned [Hog 90]. Furthermore, practical testing issues such as testing purposes and test specifications are not formally addressed by any of these methods. As a whole, studies on application of formal semantic theory to practical validation problems of distributed systems are still far from satisfactory.

We next review some related issues in semantic formalization of distributed systems.

We have reviewed four main semantic techniques on which most of the semantic models in literature are based. However, the specific formalization details of the models in each approach are not discussed. Actually, by considering or emphasizing on different factors of system operations, many different models or variants of the same model can be(or have been) defined under each technique. We summarize the two factors which are often treated differently in formalizing a model.

i) Internal Actions and Divergence

Internal actions are actions which represent internal computation steps and whose behaviors cannot be observed from the environment. As a result, system behaviors induced by

internal actions cannot be controlled by environments. When a system is engaged in an infinite number of internal actions, it is said to be divergent. Many models are different only in their different ways of treating this factor. Examples include both bisimulation-based models [Mil 81, Mil 83, HM 85, Hen 81] and failure-set based models [Nic 87, OH 86, RR 86, DG 87, KH 80, LBDG92]. In addition, Hennessy [Hen 85, Hen 88] also replaces the internal action τ with a nondeterministic choice operator $\oplus$ for his acceptance tree model.

ii) Interleaving and True Concurrency

Two major classes of semantic models are distinguished, that is *interleaving* semantics and *true concurrency* semantics. Interleaving semantics interprets parallelism between independent processes by all possible interleaving of parallel actions, such as $a \mid b$ is interpreted as $ab + ba$. Interpreting concurrency as nondeterministic interleaving is a useful simplifying method. It allows construction of elegant semantic models in which the specification and validation of large systems are possible. However, some comments against interleaving semantics are that they do not consider the causal relations between actions. Thus, another class of models based on "*true concurrency*" approach is developed The most representative theory of the true concurrency are *Labelled Event Structures*(LES) [Win 82, Win 87] which are based on Petri-net [Pet 73, NPW 81]. LES plays the role of LTS as the basic operational interpretations of processes in true concurrency approach. Almost all semantic techniques mentioned above have been applied to LES for modeling formalizations [BC 86, Cas 88, ADF 86, DNM 88, Bes 87]. But, these true concurrency models are mainly concerned with representations of processes, and in general, do not provide constructs(a language) for defining processes. As a result, they only contribute to theoretical research of process behaviors at present.

Other Related Semantic Research

Other approaches such as Algebra of Communication Processes(ACP) [AB 84, BK 84], Modal, temporal Logics [Boud 85, BKP 85, Pnu 85, Pnu 86, Bau 88, Ben 89], axiomatization and so on [Hoa 69, MP 74, LG 81, Dar 82, BR 83, Sou 84, KSR+ 85, Sou 86, AR 87] have also been used for investigating the properties of distributed systems. However, this is beyond the scope of this thesis.

1.2.3 Discussion

As a summary to the development of semantic research surveyed above, we might say that theoretical studies on formal semantics of distributed systems have been well developed. The major representative semantic approaches, LTS, bisimulation, refusal/acceptance of actions, and testing, provided solid contributions to formal studies in specification and validation of distributed systems, and resulted in mathematically elegant theories of processes such as bisimulation equivalence, testing equivalence and failure equivalence. Furthermore, these approaches and related modelling methods play a significant role in guiding other and future semantic studies of distributed systems. However, regarding practical applications of these approaches, at the state of art, the existing models are still being experimented on with respect to their suitability to be used for development of distributed systems.

We understand that the reason for this situation is that considerable variation is possible in understanding and modelling the behaviors of processes in a distributed system due to its distinct interactive and concurrent features. Designing a suitable semantic approach which can closely capture the practical operations of a system is a difficult task. Thus, we expect

that the existing behavioral views of distributed systems can still be improved in some aspects.

The following aspects are observed which may not be or less addressed in the present techniques:

i) In general, these techniques do not *directly* capture the dynamic operational relation of process functions and their relations. For example, bisimulation tries to recursively match the transitions at each corresponding pair of states between two processes, while failure semantics investigates refusals after each trace in a process. Both of them do not reflect a direct relation between process functions(complete traces, defined later) in their execution.

ii) These techniques were not intended to model explicitly the notion of *environment* or *user intentions*, or in the case of validation, verification goals or *test purposes*. This point is extremely useful during test case design, for example.

iii) Most importantly, both environments/users of a process and the process itself can have influential powers to the whole system behavior. Users may have a particular ranking of preferences in offering an action to the process; process may be designed not to perform all possible actions, but only those necessary for each particular configuration in its operation. The *actual behaviors* of a system will be determined by the interplay of these two factors. Current semantic models do not consider this user/process interplay.

We are going to address these points and design a new behavioral view to capture a kind of practical interactive operations in a distributed system.

1.3 Contributions and Organization of the Thesis

This thesis develops a new semantic technique for modeling the behaviors of distributed systems. Our main objective was to contribute towards the practical applicability of formal semantics to the development of distributed systems, particularly in the aspect of conformance testing. The scope of the study was similar to other semantic theories like bisimulation [Mil 80] or failure equivalence [BHR 84].

The basic methodology of this approach yielded a new behavioral view of system operations based on *mutual influences* between a process and its environment during their interactions. By taking into account the nondeterministic rippling of the process and the control enforced by its environment, this behavioral view was formalized as a new semantic object denoted *Nondeterministic Ripple Set*(NRS) which naturally denotes the process functions and their behaviors. Then, as a formal semantic object, the NRS was used to establish a formal semantic model for distributed systems.

An equivalence entitled *Nondeterministic Ripple equivalence* ($=_{NR}$) was defined between processes by means of the NRS. Based on this equivalence, we provided an axiom system for a process calculus based on our NRS approach. The soundness and completeness of the axiom system with respect to $=_{NR}$ were proved.

Application of the NRS semantic approach to process testing was studied. A testing theory was formalized where a testing preorder/equivalence called *Nondeterministic Ripple Acceptance testing* ($\geq_{nra}/=_{nra}$) was defined. The major contributions of this testing theory include 1) tests are generated from each process under test; 2) each test can be defined from a test purpose. Thus, this testing theory can be used to produce a set of meaningful and manageable tests which contributes to solving the issue of test selection from infinite

possible tests. The equivalence $=_{nra}$ was compared with $=_{NR}$ and failure/acceptance tree equivalence. We proved that $=_{NR}$ is different from $=_{nra}$, and $=_{nra}$ has the same distinguishing power as failure/acceptance tree equivalence.

Comparison of NRS approach with other theories, extension of NRS definitions, and some practical applications of NRS semantics were also discussed.

The organization of the thesis is roughly divided into three parts. The first part, consisting of chapters 2 - 4, is dedicated to the studies of the basic behavioral view of the NRS approach. This includes the formalization and algebraic characterization of NRS. The second part, including chapters 4 and 8, mainly discusses the applications of the NRS approach with respect to testing. This includes a testing theory in chapter 4 and some consideration of practical applications of the NRS approach in chapter 8. The third part consists of chapters 6 and 7. It provides an evaluation of the NRS approach and its extensions. Finally, in chapter 9, we conclude the thesis and point out the future studies of the NRS approach.

♠ ♠ ♠ ♠

In this chapter, we have set up our goals and research environment. Next, we will present the detailed development of our new semantic technique. In chapter 2, we motivate most of our ideas which will be formalized in later chapters.

Chapter 2

Motivation

In this chapter, we *intuitively* present a method of observing system behaviors by utilizing a type of canonical example, *vending machines*, to motivate most of the concepts precisely formalized in later chapters.

2.1 Practical Experience

One of the main characteristics in a distributed system is communications between its distributed entities. All distributed entities operate in parallel and independently to some extent, and coordinate their actions by communication or synchronization. Due to the parallel and independent nature of the entities, nondeterminism in a distributed system is a difficult phenomenon to model. Considerable variation is possible in viewing and modelling the nondeterministic behaviors of processes as reviewed in section 1.2. We shall now motivate another view to help model nondeterministic behaviors in a distributed system.

This view comes from our experience of observing process behaviors in testing distributed telecommunication systems, specifically in activities of test case generation and observing system behaviors in test execution. The presentation of this view is abstracted into the CCS [Mil 80, Mil 88] or CSP-like [Hoa 85] setting with references to testing as motivation.

Behaviors of a distributed system are often specified, observed or tested at an interaction point where the system is divided into two entities denoted a *process* and its *environment*,

respectively. Interactions between the process and its environment take place at the interaction point, and at any moment, both the process and its environment can offer a certain (or set of) action(s) to each other for communication or synchronization [OH 86, BHR 84]. In this setting, for testing processes with *nondeterministic* reactions or transitions, three aspects should be considered.

(I) Each time an observer or tester (environment) interacts with a process at an interaction point in a distributed system, he or she has a specific purpose in mind, namely, a *verification goal* or *test purpose*, to validate conformance of the system's implementation to its specification. Each system's service or capability is called a *function*. The correctness of a function can usually be tested by making the process travel through the *path* which corresponds to the implementation of this function. From the environment(user) point of view, the role of the tester(or test case) is to interact with the process at the interaction point, try to ***induce*** or *guide* the process to follow that path, and observe the sequence of actions(trace) interacted to decide on the correctness of the function implemented. This determines that at every state of the process encountered during a test case design or test execution, the tester knows which action he should offer.

(II) The *nondeterminism* in the process always interferes with the purpose of the tester. For example, the tester tries to guide the process to follow the specific path determined by his purpose, but the nondeterministic transition at a state may *steer* the process into another path which fails the test purpose. A practical scenario is that a tester tries to establish a connection between two communicating entities to transfer a data file. But, after his *connection-request* action, instead of a *connection-confirmation* action, he obtains a *disconnection* action. In this case, he is not successful in realizing his test purpose.

(III) When a path determined by the test purpose cannot be followed because of the process's nondeterminism, the test execution cannot be stopped right away. The process has to be guided further until it reaches a proper *stop/exit* point. This is usually the starting point of the process for next execution, or the exit point of the path being followed by the process. For finite processes, the exit point of the path being followed should be reached; while for infinite processes, an executing session should be completed. For example, a communication service provider or operating system executes continuously; however, a reasonable *session* for working with it is a complete data transfer from connection request to disconnection confirmation, or a complete duration from login to logout. This consideration always leaves the process at a known stable state for other operations.

According to these aspects in testing, we know that the behavior of a system cannot be decided by either a process or its environment alone. It actually depends on the two factors, the *test purpose* of the environment and the *nondeterministic transitions* inside the process. These two factors cooperate with and exert some control upon each other, and determine the functions or traces performed by the system. Assume that the environment's test purpose (or control) is a ***known condition*** which can be determined in advance. Then, in a testing execution, ***a set of traces*** has the potential to be followed due to nondeterminism inside the process. This set forms a semantic denotation, denoted ***nondeterministic ripple set***, to characterize behaviors of the process.

These general analyses are illustrated in the following sections by using a vending machine example. This is done from two directions. (1) Predicting process behaviors from its specification, i.e. specification-oriented which corresponds to activities of test case generation. (2) Observing process behaviors from its implementation which corresponds to activities in test execution. Analyses from both directions reach the same result of observations. In this illustration, we use *synchronization trees* [Mil 80] or labelled rooted

trees [Hen 88] to represent processes, and base their semantic operations on labelled transition systems (defined in chapter 3) [Mil 80, Hen 88].

2.2 Predicting Process Behaviors from its Specification

Consider this scenario. In the initial stages of surveying the market need for a particular vending machine product, a company has determined that a machine could be well accepted in the market place provided:

i) the machine accepts a coin from a customer, and subsequently dispenses coffee, tea, or juice(This is a generic specification of functions, denoted as a static property).

ii) because the demand is highest for coffee, the machine must always be able to dispense coffee(this is an operational constraint on machine behavior and is a dynamic property).

iii) since tea and juice are merely alternatives for coffee(in this market), and demand for tea or juice is very low, it is acceptable for the machine to be unable to dispense tea or juice(or both), depending on the quantity on hand(again, operational constraints on machine behavior, i.e. dynamic properties).

Given these requirements, many acceptable formal specifications can be produced. Two formal specifications are shown in Figure 2.1 as labelled rooted trees *P1* and *P2* representing nondeterministic processes, where α, β, γ denote the possible machine states after the transition(labelled with action *coin*) is taken. We also call *P1* and *P2* processes or machines in this example when it is convenient.

Obviously, *P1* and *P2* have the same set of traces or static property (a trace is a sequence of actions along a path in a tree). This means that they have a similar ability *in principle* to provide functions or services. Now, consider the following questions regarding behaviors of *P1* and *P2*:

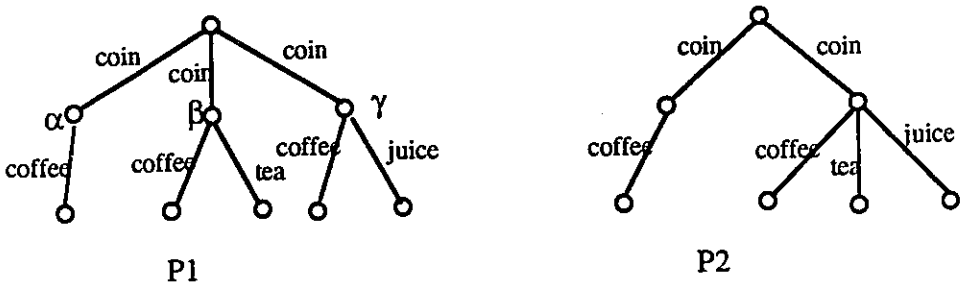


Figure 2.1. Two vending machine specifications

1) How will the machines specified by *P1* and *P2* behave during operation ?
2) Will their (nondeterministic) dynamic behaviors be equivalent ?

To answer these questions, let us analyze the operational behaviors of *P1* and *P2* by considering possible interactions between the environment(a tester) and each process.

First of all, two factors in the system operation should be kept in mind.

i) When interacting with the process, the tester has a test purpose in mind. This determines his reaction at each state of the process encountered.
ii) The process has nondeterministic transitions in its execution which is beyond the control of its environment.

Now, consider interactions between the tester and Process *P1*. Assume that the test purpose is to "***test if P1 can provide coffee***".

In the first interaction, the machine *P1* accepts a coin, executes one of the three possible transitions nondeterministically and reaches a new state as a result. The new state may be any one of α, β or γ depending on *P1*'s configuration. Which new state results is beyond the control of the tester.

At this point, the tester may choose a different action depending on the different present state. However, his purpose is to test if the machine can provide coffee, then, he insists on a cup of coffee. Thus, he will press the coffee button and will always be able to receive coffee regardless which state the machine went to after his coin was inserted. The entire history of the provision of this service, also called a **trace** of interactions between the tester and *P1*, is:

i) {*coin.coffee*}.

In this interaction session, the action offered to *P1* by the tester at each state encountered is

{(*r*, *coin*), (α, coffee), (β, coffee), (γ, coffee)}

Note that we label the root state of *P1* and *P2* as *r*. This set of action selections is determined by the test purpose, and can be understood as the *control* exerted to the process by its environment. In Chapter 3, we call this kind of action selections as a choice pattern.

In another case, the tester may set *to test if P1 can provide a cup of tea or juice* as a test purpose instead. After a coin is inserted, the machine may end up in state β, indicating that coffee or tea is available. This may happen because juice may be temporarily out of stock, or because even though both tea and juice are available, the machine nondeterministically chose to execute the middle transition of *P1*. Then, the tester can press the tea button and get a cup of tea. Alternatively, if the state γ is reached, then the tester can enjoy a cup of juice. But if the state α is reached nondeterministically, the tester has to settle for coffee to

continue the interaction. Thus, the potential traces which may occur in this interaction session are members of the set:

ii) {*coin.tea, coin.juice, coin.coffee*}.

Action selections in this test session are

{(*r, coin*), (α, *coffee*), (β, *tea*), (γ, *juice*)}

Other cases can be derived from the same analysis for other test purposes on machine *P1*, namely those which test for provision of tea, juice, tea or coffee, juice or coffee, or tea, juice or coffee. The additional sets of potential traces which may be accepted by P1 for these other cases are the sets:

iii) {*coin.tea, coin.coffee*} and iv) {*coin.juice, coin.coffee*}

Thus, for all possible test purposes (note that a test purpose here is to test provision of one or more drinks or services), we obtain a collection of four sets of traces (NRS(*P1*) in Figure 2.2, where each of the trace sets consists of all traces which may be potentially involved in a test interaction session between a tester and machine *P1* for a specific test purpose.

Note that more than one trace may be included in a trace set ii), iii), or iv). It is because of the process's ability to make nondeterministic transitions at its current state, and the rippling effect of such transitions upon continuing interactions. This can be understood as the *control* (or behavioral influence) exerted to the environment by the process in interactions.

We call each set obtained in the above analysis a ***nondeterministic ripple set***(*nrs*). The set of all nondeterministic ripple sets of a process *P* is denoted as NRS(*P*).

{{*coin.coffee*}
{*coin.tea, coin.coffee*}
{*coin.juice, coin.coffee*}
{*coin.tea, coin.juice, coin.coffee*}}

NRS(P1)

{{*coin.coffee*}
{*coin.tea, coin.coffee*}
{*coin.juice, coin.coffee*}}

NRS(P2)

Figure 2.2. Nondeterministic ripple sets of specifications P1 and P2.

By applying a similar analysis to machine *P2* in Figure 2.1, the nondeterministic ripple sets of *P2*, denoted NRS(*P2*), can be obtained and are shown in Figure 2.2. Note that there are only three nondeterministic ripple sets in NRS(*P2*). The set {*coin.tea, coin.juice, coin.coffee*} which is in NRS(*P1*) cannot be derived for *P2*. This is because there are only two states(three for *P1*) which can be potentially reached after a coin is inserted into *P2*. A tester can only have two potential states to make synchronizations at this point in *P2*.

By the above analysis, we understand that the sets NRS(*P1*) and NRS(*P2*) form a kind of behavioral representation of processes *P1* and *P2*. This representation reflects the interactions between the two factors in a system operation, i.e., environment control over the process by the test purpose, and process control over the environment by its nondeterministic transition. It tells us that a system behavior is the outcome of mutual influences of two contributing factors from the system's two participating entities. This answers question 1 above. Question 2, namely whether or not (nondeterministic) behaviors of *P1* and *P2* are equivalent, can be answered in the negation. The difference between specifications *P1* and *P2* in Figure 2.1 can be intuitively observed in a number of ways, such as the following:

i) Assume that the probability of choosing each of the transitions labelled by *coin* at the root nodes of *P1* and P2(Figure 2.1) is identical. Then, the probability of getting tea(or

juice) in machine *P1* is 1/3, while in *P2*, it is 1/2. This means that the operation of *P1* is more *flexible* than *P2*.

ii) Alternatively, consider an interaction scenario(also a testing scenario) in which the ability to dispense coffee is (possibly temporarily) taken away from both *P1* and *P2*. This means that the left transitions labelled by *coin* after the root nodes can not be chosen by both machines. In this scenario, whether or not tea(or juice) can be obtained successfully during interactions is nondeterministic in *P1*(actually, probability is 1/2), while in *P2*, the request for tea or juice can be always satisfied(chances are 1).

The same analysis can be done from another point of view as discussed in the next section.

2.3 Observing Process Behaviors from its Implementation

In this subsection, we analyze the behavior of a process from an observational point of view. Here, our start point is an implementation of the process specification. The same result about the behavioral observations is obtained, but with a more intuitive tree representation.

A user or tester interacts with an implementation of the process(i.e., conceive the vending machine as a black box) through its interface. We may imagine that a frequent user of the vending machine can form the behavior of the machine by repeated use of its services, or for a professional tester of the vending machine, the specification of the machine is *available* to him, and the tester tries to formalize a set of behaviors of the process which should be observed from the interface of its implementation for testing the implementation against its specification.

Let us imagine that the vending machine designed from the specification *P1* or *P2* has an interaction interface as shown in Figure 2.3.

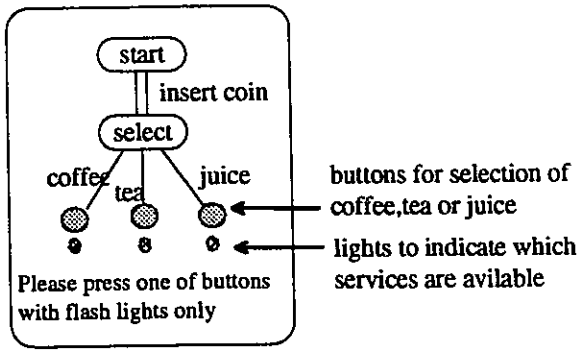


Figure 2.3. Interaction interface for machines of *P1* or *P2*.

The machine is at *start* state in idle situation. When a coin is inserted in, it will reach *select* state, flash any of the three lights to show which drinks are available at the moment, and wait for a button with flash light to be pressed. Then the machine provides the corresponding drink.

With this setting, we first consider the machine which is based on specification *P1*. A user who only likes coffee inserts a coin into the machine and then press the coffee button. He is always satisfied with a cup of coffee. The actions he takes for this service are insert_*coin*.and press_*coffee*_button, which make him deduce that the machine has a kind of behavior which can be represented as a labelled rooted tree in Figure 2.4(a). However, inside the machine, from specification *P1*, we know that one of the three potential paths(Figure 2.4(b)) may be followed nondeterministically, which is beyond the knowledge of the user. What he sees is only the tree in Figure 2.4(a).

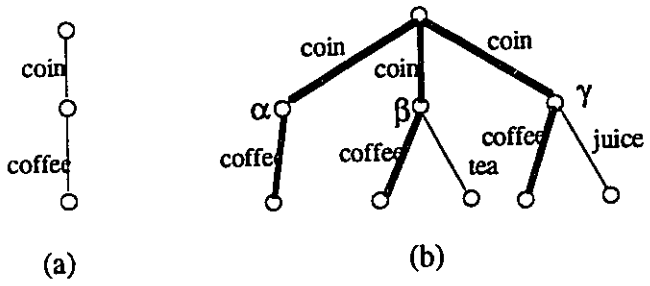


Figure 2.4. Observable behavior for getting coffee and potentially traversable paths

Now, let us see how a user who prefers tea or juice feels the behavior of the machine. After a coin is inserted, the lights for tea and coffee may flash, then the user has a satisfiable choice of tea. Alternatively, when the lights for juice and coffee are flashing, he may take juice. One more possibility is that only light for coffee is flashing, then he is forced to have a cup of coffee. So, the customer(assume he is a frequent user of the machine) deduces that the machine has a behavior, that is, one of the three sequence of actions may be taken during an interactive session. this happens due to the effect of mutual influences between the machine's then configuration and his or her habit of prefering tea and juice. This behavior can be represented as a deterministic labelled rooted tree as in Figure 2.5(a). Although the possible paths followed inside the machine may be bold ones as shown in Figure 2.5(b).

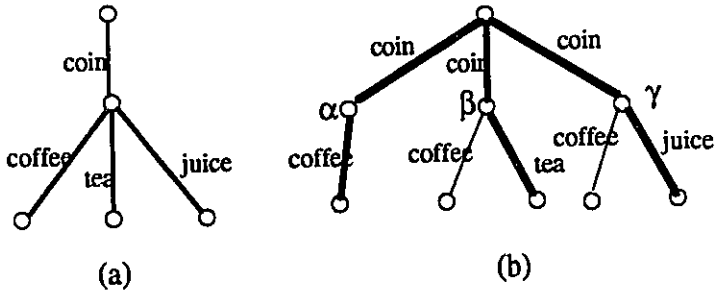


Figure 2.5. A behavior of the machine with preferred service tea or juice.

Two more behaviors observed from the interface of the machine are trees as in Figure 2.6(b) and (c), which are deduced from users with other preference in their choice of drinks, namely those who like tea but not juice,or juice but not tea, with tolerance of coffee if both tea and juice are not available. More user preferences can be formed, but all the behaviors deduced this way are four trees in Figure 2.6, i.e., trees (a),(b),(c), and (d).

Note that the user preference here plays the role of a test purpose which determines the action choice at each state encountered during interactions.

For the machine designed based on specification *P2* with the same interface as in Figure 2.3, a similar analysis shows that the machine's behaviors can be represented by trees (a),(b), and (c) in Figure 2.6. (d) is not considered as a behaviour of *P2*, because there are only two potential states after a coin is inserted. Thus, in each execution, at most two traces are potential to be followed.

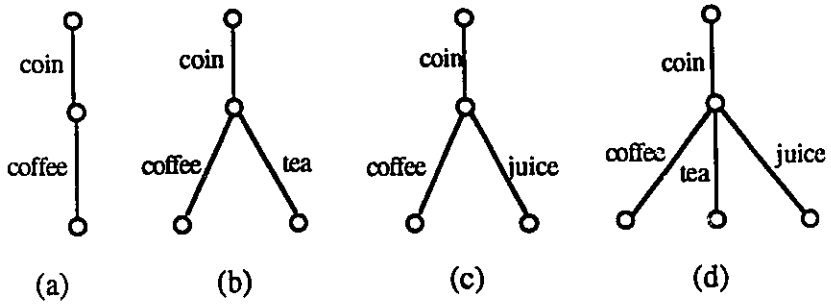


Figure 2.6. Behavioral observations from the machines based on *P1* and *P2*.

This analysis of observable behaviors of processes also distinguishes between processes *P1* and *P2* due to their different behavioral tree sets.

From the procedure of how these process behaviors are derived and their representation structures, it is easy to understand that this procedure is actually a test generation process

from a given specification for its implementation, and the set of behaviors forms a **test suite** for test selection and test specification.

Furthermore, an inherent relation exists between the nondeterministic ripple sets of a process and the intuitive tree representation of its behaviors. These two representations reflect the same behavioral view of a system from two different aspects of interactions between a process and its environment. The relation between them will be proved in section 3.4.

♠ ♠ ♠ ♠

We have intuitively analyzed the behaviors of two vending machine processes, motivated a semantic object: nondeterministic ripple set, and indicated some possible applications of this behavioral view to interpretation and testing of distributed systems. In the illustration, some *philosophical* and *formal attributes* of distributed systems have been used to illustrate this view of observing processes.

In a distributed system, at some observation/testing point, a process and its environment constitute a system to perform specified functions by interacting with each other. Both the process and environment have their own behaviors which impose controls on the system behavior. The system behavior is actually the result of mutual influences of its two sub-behaviors in the process and its environment.

In the circumstances of testing, the environment behavior is reflected as the action selections at the states encountered during interactions which form environmental controls to the process. The environment behavior can be determined in advance based on a test purpose or similar consideration. The process behavior is reflected by its nondeterministic

transitions during execution which form control to the environment. In a system operation, these two behaviors cooperate with and control to each other to form a system behavior.

To represent these attributes in a system, by recording the actions interacted between a process and its environment, the system behavior can be represented as a sequence of actions or a trace which corresponds to a service or function of the system. However, only assume that the environment behavior is known, the process behavior can be predicted as a set of traces which may be potentially followed in an interaction session. This is because the nondeterminism inside the process and the mutual influences between behaviors of the process and its environment. We denote this set of traces nondeterministic ripple set which is the main topic of this study.

All the concepts motivated in this chapter will be precisely discussed in the following chapters.

Chapter 3

Characterization of Semantic Behaviors of Distributed Systems by NRS

In this chapter, we *precisely* develop the semantic object, *Nondeterministic Ripple Set* (NRS) motivated in chapter 2 by representing processes as *labelled transition systems* and its alternative representation: *synchronization trees.*

Labelled transition systems (LTS's) [Plo 81, Mil 80] have been generally recognized as an appropriate operational model/semantics for describing nondeterministic processes in a distributed system. LTS's interpret processes as evolving through states via successive transitions. Each transition is labelled by an action which may represent an interaction with the environment or an internal computation step. We shall here concentrate on a class of finite *acyclic* labelled transition systems without internal actions for convenience of reasoning at the moment. Processes with recursive behaviors and internal actions will be treated in chapter 7.

We show that based on labelled transition systems, the controls exerted to the system by both a process and its environment can be defined as two concepts: *nondeterministic rippling* and *environment choice pattern.* Then, by considering the mutual influences of these two controls, we define the semantic object, nondeterministic ripple set, for a process.

Labelled transition systems are good at interpreting system operations, but they have a *loose* nature for defining processes (as a program, for example) and studying their algebraic properties. Thus, we also use an alternative notation of LTS's, Synchronization

Trees (ST's), as an intuitive, concise representation of processes for reasoning algebraic properties of processes. ST's were first proposed by Milner for CCS processes and his observation equivalence [Mil 80]. Then, they have been used for studying algebraic properties of systems specified by CCS or CSP [Hoa 85, Bro 83, BR 83]. By adapting ST's, our work can be directly applied to a class of processes with basic operators in CCS, and an easy correspondence can be established with other work in literature for comparison. In this and later chapters, we often use ST's for graphical representation of processes in examples.

This chapter is organized as follows. In section 3.1, we focus on the basic definitions for labelled transition systems and nondeterministic ripple sets. Then, in section 3.2, we define an alternative process notation, ST's and derive a formula for calculating NRS for processes represented as ST's. Finally, in section 3.4, we prove an alternative deterministic tree representation of NRS(P) for a process P.

3.1 Labelled Transition Systems

From now on, all the precise formulation of concepts and reasoning are based on processes interpreted operationally as labelled transition systems. We define the notion of *Labelled Transition Systems*(LTS's) in this section.

Definition 3.1. A labelled transition system(LTS) is a quadruple $(P, Act, r, \longrightarrow)$, where

i) P is a set of states(processes)

ii) Act is a set of actions

iii) r is the initial state, root

iv) $\longrightarrow$ is a transition relation $\subseteq P \times Act \times P$

Intuitively, $\longrightarrow$ is the set of transitions which may progress from one state to another when an action is interacted in the state. Unless otherwise specified, we assume that Act does not include internal actions such as τ. We write a transition $(p, a, p') \in \longrightarrow$ as $p \text{—}a\text{→} p'$ for convenience. We use $p, q, p' \cdots$ to range over P, and $a, b \cdots$over Act. Furthermore, the relations $\text{—}\, a \,\text{→}$ are extended to sequences of actions(*traces*) $p \text{—}\, s \,\text{→}\, p'$, for every $s \in Act^*$, as follows,

i) $p \text{—}\, \varepsilon \,\text{→} p'$ if p' is p

ii) $p \text{—}\, as \,\text{→} p'$ if $p \text{—}\, a \,\text{→} p''$ for some p'' such that $p'' \text{—}\, s \,\text{→} p'$.

This means that $p \text{—}\, s \,\text{→} p''$ if p can evolve to p' by performing the sequence of actions s. We also use $p \text{—}\, s \,\text{→}$ to mean that there exists a p' such that $p \text{—}\, s \,\text{→} p'$. For a LTS, a sequence of transitions is called a derivation or an execution of the LTS, and the sequence of actions labelling the sequence of transitions in an execution is called the trace followed in this execution. For technical convenience, we shall only consider LTSs with *finite* branching at their states, that is, for each p in P, the set $\{p' \mid \exists\, a, p \text{—}a\text{→} p'\}$ is finite, and *finite* sequence of transitions, that is, in each derivation, $p \text{—}s\text{→} p'$, s is finite, in chapters 1 - 6. Processes with recursive behaviors and internal actions are discussed in chapter 7.

Note that in an LTS, it is not always necessary to have an initial state. However, we are primarily interested in systems with an initial state, *root*, such that all other states can be accessible from the root. This condition holds for all LTS's used in rest of this thesis.

Given an LTS = $(P, Act, r, \longrightarrow)$, each state $q \in P$ may be thought of as an initial state and define some other LTS = $(Q, Act, q, \longrightarrow_q)$, where q is the root, $Q \subseteq P$ consists of the states reachable from q via some finite sequence of transitions and $\longrightarrow_q$ is the restriction of $\longrightarrow$ to Q.

According to our definition, an LTS is a process starting from some initial state and evolving through successive states by means of elementary transitions. By understanding that each state of the LTS also defines some other LTS, we may then regard the LTS as giving rise to new processes by transitions, rather than going through successive states. Accordingly, we shall also refer to *states in P* as *processes*. Usually, when we intend to refer to a process in general as an LTS, we use *capitals* such as T, P, Q to represent it, but when we want to emphasize on a state of a process, we use small letters for it. Given an LTS, it is easy to understand that a correspondence can be established between its states and the processes defined by the states. After we introduce the synchronization trec(ST) notation of LTS's, a state will be often labelled by its corresponding ST description.

3.2 Nondeterministic Ripple Sets

In this section, we define the fundamental semantic object, Nondeterministic Ripple Set(NRS). First, we discuss some operational characteristics of a process and its environment, then, we formalize the concepts, nondeterministic rippling, choice pattern and nondeterministic ripple set.

3.2.1 Operational Characteristics of an LTS.

Now, we discuss some operational characteristics in a labelled transition system.

1) *Environment and process controls over the system operations.*

In a distributed system, a process and its environment constitute a system to perform specified functions by interacting with each other. According to the LTS interpretation of process operations, both the process and its environment have control over the system

behavior. This is reflected by the deterministic and nondeterministic transitions in the process. Deterministic transitions at a state (i.e., transitions labelled with different actions) reflect the environment's control over the system, i.e., the choice between transitions is made by the environment, while nondeterministic transitions at a state (i.e., transitions labeled with a same action) reflect the process's control of the system, i.e., the choice between transitions has to be made internally by the process without knowledge of its environment. The similar observation about this phenomenon between a process and its environment has been mentioned in [Hoa 85]. We illustrate this with a state transition step in Figure 3.1.

Assume that in an LTS, transitions at state p_1 are defined as in (a) or (b) of Figure 3.1. The two transitions (branches) in (a) are deterministic, then, the choice between these two transitions is made by the environment. If the environment chooses action a, the transition $p_1 — a \rightarrow p_2$ is executed, and if it chooses b, the transition $p_1 — b \rightarrow p_3$ is followed. However, if the transitions at the state are defined as in (b) where the two transitions are nondeterministic, then, even if the environment offers action a at state p_1, it is unclear which state between p_2 and p_3 will be reached during this execution, because this choice has to be made by the process. Actually, both transitions have the *potential* to be executed. We say ***potential*** because it is beyond the user's control to know which branch will be traversed.

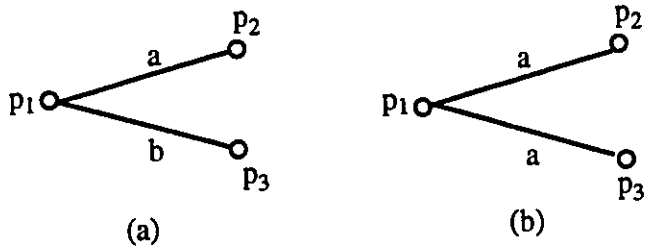


Figure 3.1. Environment and process controls over the system operations

2) *Mutual influences between a process and its environment.*

During an execution course, because of the possible deterministic and nondeterministic transitions at states encountered, environment and process controls can exert influences on each other's behavior (transition choice), and they must cooperate with each other to form controls of the system to keep an execution going. We call this the ***Mutual Influences*** between a process and its environment. This mutual influence can be shown at the state in Figure 3.2 by the interplay between the process and environment behaviors.

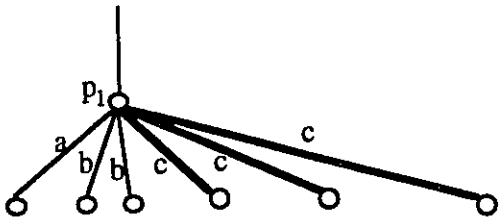


Figure 3.2. Mutual influence between a process and its environment

At state p_1 of Figure 3.2, there are three groups of transitions labelled with a, b and c respectively. Choices among these groups of transitions are deterministic. Thus, the environment must make a choice and tell the process which action(a, b or c) should be performed. This determines a group of potential transitions to be performed. We mean that this is the *influence* exerted on the process by the environment control. For a specific action chosen by the environment, the process can make a choice among all possible transitions provided (i.e., branches emanating from state p_1 and labeled with that specific action) without the knowledge of the environment. For example, if the environment chooses action c to perform, the process can choose any one of the three branches labeled with c, or, if action b is chosen, the process has two possible choices between branches labeled with b. For this interaction step at p_1 , if the environment chooses action c, then, the potential branches which may be followed are the three bold branches labeled with c.

At the next transition step, the environment has to consider an interaction with the process at any of the three potential states reached by performing action *c*, if the execution of the system can be continued. We mean that this is the *influence* exerted on the environment by the process control.

3) *Process behavioral characterization*

According to this understanding of interactions between a process and its environment, it is not difficult to see that a specific transition or transition path(successive transitions from the initial state) traversed in an execution of the process is determined by the ***mutual influence*** or ***coordination*** between nondeterministic and deterministic choices made independently by the process and its environment at each state encountered during execution. Thus a path traversed during execution is decided by two factors as a point in a plane is decided by the point's two coordinates. This path cannot be determined ***in advance***, and it depends on the dynamic coordinating activities of both sides in an actual execution of the process. Neither process nor environment alone can select a specific path for execution in a nondeterministic system.

Let us analyze a process behavior from its environment side. The environment can be a user or a tester who has a *specific purpose* in mind during his interactions with the process. Thus, it knows what coordinate activities he should choose at states encountered. In the case of testing, the tester is always biased towards a set of particular choices of actions determined by its *test purpose* to control the process in order to ***induce*** the process to show some expected behavior. This occurs for example when the tester is attempting to test if a normal or abnormal behavior appears in the process implementation under test. This practice, so called *conformance testing*, is often used to verify that the actual behavior of the process conforms to its specified or expected behavior [MoPr 92]. Importantly, note

that testing execution is always applied to an implementation of the process which is a blackbox with the interface for interactions. However, the setting we are discussing now is similar to the ***one for test generation*** where we can work with the process specification, *imagine* its potential execution cases based on its operational semantics (LTS), and characterize its behaviors. A reader needs not worry about how to look into the process.

Due to nondeterministic transitions of the process (i.e., process control), the tester cannot know in advance which path will be traversed during an execution. Thus, he has to cooperate with the process by choosing proper actions at different potential states to keep the execution going according to his test purpose. This means that in each execution, a specific, biased set of paths has the potential to be traversed. For example, in Figure 3.2, starting from state p_1, at least three paths with initial transitions labeled by c have the potential to be followed. The environment (or tester) can determine the action selections (i.e., environment control) at all encountered states based on the test purpose. In realistic situations, the correspondence between a test purpose and the relevant action selections can be often decided easily. Given a set of action selections, the corresponding set of potential paths can be determined *in advance* according to the process's specification by enumerating all possible paths which may be followed while *satisfying* the set of action selections. This set of the potential paths and the set of the traces labeling these paths can be used as process behavioral characterization.

3.2.2 Formalization of Nondeterministic Ripple Set

We now formally define the ideas discussed above. First, we define some notations and formalize the process control.

Definition 3.2 (*Ripples or Derivatives at a state for process control*)

For an LTS = $(P, Act, r, \longrightarrow)$ and $p, p' \in P$, $a \in Act$, we define

$$D(p, a) = \{p' \mid p \text{—}a\rightarrow p'\},$$

$D(p, a)$ is called the *Ripples,* or *Derivatives* at p with respect to a.

At any state p of an LTS, let $D(p, a) \neq \varnothing$. Then, during a transition with action a at p, the process has the potential to reach any p' in $D(p, a)$ after performing action a. Which p' is reached may not be under the control of the environment. We call this ***nondeterministic rippling effect of the process with respect to action a at state p,*** or simply *nondeterministic rippling*. *Nondeterministic rippling effect* intuitively denotes the spreading of possible execution paths from a state, and formalize the concept: **process control** on the system behavior. For convenience of describing definitions, we also define the following notations.

Definition 3.3 For an LTS = $(P, Act, r, \longrightarrow)$ and $p, p' \in P$, $a \in Act$, $s \in Act^*$, respectively,

i) $D(p, s) = \{p' \mid p \text{—}s\rightarrow p'\}$, the *s -Ripples,* or *Derivatives* of p.

ii) $S(p) = \{a \mid p \text{—}a\rightarrow\}$, the *Successors* of p.

Now, we represent the environment control. All action selections made by the environment at all potential states encountered during an execution can be represented as a set C which is a subset of the set of pairs: $\{(p, a) \mid p \in P \text{ and } a \in S(p)\}$, i.e.,

$$C \subseteq \{(p, a) \mid p \in P \text{ and } a \in S(p)\}.$$

Each (p, a) in C stands for the action a chosen by the environment at state p. We use C to describe an environment control over a process in an execution, called a choice pattern, as defined below.

Definition 3.4 (*Choice pattern for environment control*). C is called a *choice pattern* of process P, if C satisfies the following conditions.

1) $(r, a) \in C$, r is the initial (root) state of process P

2) If $(p, a) \in C$, then for all $q \in D(p, a)$ where $S(q) \neq \emptyset$,

there exists exactly one $a \in S(q)$, such that $(q, a) \in C$.

Note that in a choice pattern, exactly one action can be selected at each state. We represent the set of all choice patterns of a process P as $CS(P)$.

A convenient "intuition" about why we use a set of pairs to represent *environment control* is this: imagine a user starts an interactive session with a process/machine in an interactive system. At the start of the session, the machine is at its initial state, the user has several choices to perform an action to gain some control over the machine. Alternatively, as in many existing systems, the machine may present the user a menu of all possible actions the user may perform at this state. The user can only perform one action out of these choices to control the machine at this state. We can represent this control-directed choice by a pair such as (r, a), where r represents the initial state and a the action chosen by the user (environment) at state r. Then, the machine goes to the resulting next state, say p. Exactly which next state is reached depends on the machine's internal behaviour and may appear to the user to be nondeterministic for different executions. This perceived nondeterminism may be real, due perhaps to concurrency and race conditions, or only virtual, due to information hiding or delay design decisions. At state p, the user again has choices and choose one action to interact with the machine. The user's action selection at state p is recorded as a pair (p, a). Thus, it is easy to understand that in this interactive session, the user's (environment's) attempts to control the machine can be recorded as a set of pairs

which represent the environmental action selections at all states encountered during this session. This desire to control the machine's behaviour is most obviours during testing.

Definition 3.5 (*Nondeterministic ripple set(nrs)*)

A ***nondeterministic ripple set*** of a process is a non-empty set of maximal traces followed potentially during an execution which is under the constraint of the nondeterministic rippling effect of the process and action selections specified by a choice pattern C at all states encountered during the execution.

Operationally, given a process P, and a choice pattern C, a nondeterministic ripple set of P, denoted *nrs*, can be calculated inductively as follows, where auxiliary set PS represents all the potential states encountered in an execution, i.e., $PS = \{ p \mid (p, a) \in C\}$, and nrs_n is an auxiliary variable denoting a set of traces:

$nrs \leftarrow \{\varepsilon\}$.
repeat $nrs_n \leftarrow \varnothing$.
  for each $s \in nrs$, and $p \in D(r, s) \cap PS$
    if there exists a $(p, a) \in C$,
      then $nrs_n \leftarrow nrs_n \cup \{s.a\}$, $C \leftarrow C - \{(p, a)\}$
      else $nrs_n \leftarrow nrs_n \cup \{s\}$.
  $nrs \leftarrow nrs_n$.
until $C = \varnothing$.

Note that for a trace s, not all states in $D(r, s)$ are potentially reached during an execution because of the action selections specified in the choice pattern C. For an empty choice pattern C, its corresponding *nrs* is $\{\varepsilon\}$, while for a process with infinite or recursive behaviors, the computation in this definition may not terminate unless some exit point in the process is made. This concerns practical considerations. The name *nondeterministic ripple*

set is used because it is the result of nondeterministic rippling effect of the process among its traces.

In Figure 3.3, an example is given to show the definition of a *nrs*.

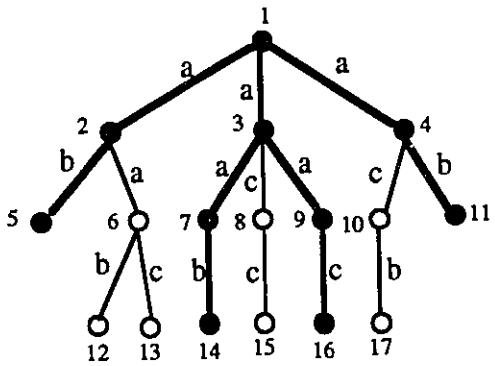


Figure 3.3. Specification of process P.

Here, suppose that the choice pattern used is:

C = {(1,a), (2,b), (3,a), (4,b), (7,b), (9,c)}.

The *nrs* defined by this choice pattern C is

nrs = {ab, aab, aac}.

Note that the potential states reached in this execution is PS = {1, 2, 3, 4, 5, 7, 9, 11, 14, 16}, Intuitively, the interactions between the process in Figure 3.3 and its environment E (assume E is a tester who is controlling and observing P) can be explained as: At the first interaction, E provides action a to P at state 1(root). P may choose one of the three branches labelled with a to follow, then after a, P may be in one of the three states{2,3,4}. At this point of the execution, E must coordinate with P to make a choice within the choice possibilities provided at each state because which state is reached is beyond the control of E. Suppose at state 2, E chooses b, then path (1,2,5) is followed; or, at state 3, E selects a, then state 7 or 9 is reached where only possible choice is b or c, path (1,3,7,14) or

(1,3,7,16) is followed; or at state 4, E chooses b, then path (1,4,11) is traversed. All the potential paths traversed are bold in Figure 3.3.

Each nondeterministic ripple set characterizes a specific nondeterministic type of behavior of a process during an execution. All possible nondeterministic ripple sets generated by all different choice patterns during all possible executions characterize all possible nondeterministic rippling behaviors of the process. Thus, we can define the following item to characterize the behaviors of the process.

Definition 3.6 (Characterization set NRS of a process)
For a process P, the ***characterization set of the behaviors*** **of** P, represented as *NRS(P)*, is the set of all nondeterministic ripple sets of P, i.e.

$$\mathrm{NRS}(P) = \{\, nrs \mid nrs \text{ is a nondeterministic ripple set for some choice pattern } C \text{ of } P\}.$$

Thus, the *NRS(P)* is the set of all *nrs* sets derived from all choice patterns of P. We also call *NRS(P)* the ***characterization*** of processes in general, which plays the same role as failure sets or acceptance trees in the work [BHR 84, Hen 88]. In the following, we always use lower case *nrs* for a single nondeterministic ripple set derived from a choice pattern, and upper case NRS to mean the set of all *nrs*'s of a process. Examples of the characterization sets are illustrated in Figure 3.4 where the processes in (a) are represented by their synchronization trees.

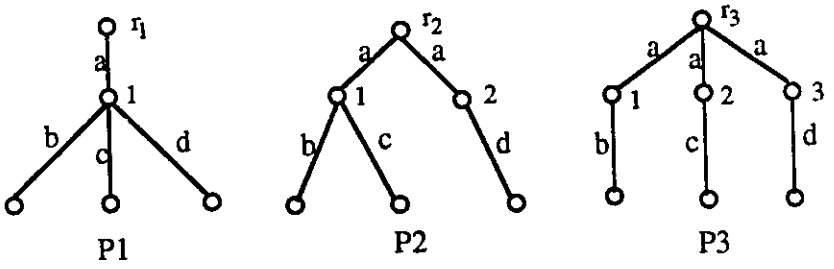
The sets of choice patterns for these processes are

$$\mathrm{CS}(P1) = \{\{(r_1, a), (1, b)\}, \{(r_1, a), (1, c)\}, \{(r_1, a), (1, d)\}\},$$

$$\mathrm{CS}(P2) = \{\{(r_2, a), (1, b), (2, d)\}, \{(r_2, \mathrm{a}), (1, \mathrm{c}), (2, \mathrm{d})\}\}, \text{ and}$$

$$\mathrm{CS}(P3) = \{\{(r_3, a), (1, b), (2, c), (3, d)\}\},$$

respectively. Their corresponding characterization sets are shown in (b).



P1 P2 P3

(a)

{{ab},{ac},{ad}} {{ab, ad},{ac, ad}} {{ab, ac, ad }}

NRS(P1) NRS(P2) NRS(P3)

(b)

Figure 3.4 Characterization sets of processes

Intuitively, that any of the three *nrs* 's in NRS(P1) contains only one trace implies that a tester can control *P1* to follow any of its three traces *ab*, *ac*, and *ad* in an execution depending on his purpose. Note that *P1* is deterministic. For process *P2*, no matter which trace the tester wants to follow, there is another trace which has the potential to be followed due to the nondeterministic rippling progress at the root state. This is why each *nrs* in NRS(*P2*) contains two traces. *P3* is fully nondeterministic, no matter which trace in it is intended to be followed, the other two traces has the potential to be traversed. Thus, *P3* has only one *nrs* including all its three traces *ab*, *ac*, and *ad*.

Definition 3.7 Trace s is a ***complete trace***, if $r \xrightarrow{s} q$ and $\neg(\exists a \in Act,\ q \xrightarrow{a})$. The set of all complete traces of a process P is denoted as $CT(P)$, i.e., $CT(P) = \{s \mid s \text{ is a complete trace}\}$.

It is easy to see that in the domain of finite processes, every trace in a $nrs \in \mathrm{NRS}(P)$ is a complete trace. It is also interesting to see that according to definition 3.6 of NRS, for a

deterministic process, its NRS consists of all those sets each of which contains exactly one complete trace of the process, that is, if P is a deterministic process,

$$\mathrm{NRS}(P) = \{\{s\} \mid s \in CT(P)\}.$$

P1 and *NRS(P1)* in Figure 3.4 shows such a process and its NRS.

Note that definition 3.6 does not yield an easy way to derive all *nrs* 's for a process in the sense that it appears to require all possible choice patterns to be enumerated so that all possible *nrs*'s can be derived. That this is not required is shown in the next section, where a more workable formula is given to derive NRS(P) directly based on synchronization trees.

3.3 Nondeterministic Ripple Sets for Synchronization Trees

In this section, we define an alternative notation for labelled transition systems, denoted synchronization trees (ST's) [Mil 80]. Then, we derive a formula to calculate the characterization set for processes represented by ST's. Synchronization trees give rise to a concise way to represent and reason about processes, and they also make our work applicable to and comparable with other approaches.

3.3.1 Synchronization Trees

Given a labelled transition system, it can always be "unrolled" into a tree in the usual way. The initial state labels the root, states label the nodes, and the actions in *Act* label the branches. The resulting tree is called a synchronization tree (ST). To be consistent with LTS, we only consider ST's with finite branches at each node. Formally, we have:

Definition 3.8. (*Synchronization Tree ST*)

A ***synchronization tree*** (ST) is a rooted, unordered, finitely branching tree all of whose branches are labelled with an action in *Act*.

Let $T, P, T' \cdots$ range over trees. We use $a.T$ for the synchronization tree with an initial branch labelled by $a \in Act$ and at the end of the branch, subtree T attached. The trivial tree with a single node and no branches is written as *NIL*. If $\{T_i \mid 1 \le i \le n\}$ is a family of trees, we denote by

$$\sum_{i=1,n} T_i$$

the tree obtained by gluing the root nodes of all the T_i. For convenience, we write $a_i.T_i$ as $a_i T_i$.

The structures of $a_i.T_i$ and $\sum_{i=1,n} a_i.T_i$ in a synchronization tree corresponds to the basic operators: *Action Prefix* $a.$ and *Summation* $+$ in CCS [Mil 88]. The processes described by ST's can also be defined by BNF-like notation as:

(3.3.0) $\qquad T ::= NIL \mid a.T \mid T + T$

According to the definitions of ST, if T_1, T_2, and T_3 are ST's, the following axioms are obvious, because both sides of each equation represent the same tree.

Proposition 3.1. $(T_1 + T_2) + T_3 = T_1 + (T_2 + T_3)$ (+1)

$T_1 + T_2 = T_2 + T_1$ (+2)

$T_1 + NIL = T_1$ (+3)

Given a ST, its corresponding LTS can be obtained as the $(P, Act, r, \longrightarrow)$, where the transition relation is defined as (the same transition rules for *Action Prefix* and *Summation* in CCS)

(3.3.1) $a.T \xrightarrow{a} T$

$T_1 + T_2 \xrightarrow{a} T'_1$ if $T_1 \xrightarrow{a} T'_1$

$T_1 + T_2 \xrightarrow{a} T'_2$ if $T_2 \xrightarrow{a} T'_2$

Each time a transition is made, a new name is given to represent the present state which is labelled by the sub-ST derived. The root is the state r labelled with the *ST;* The P consists of all states labelled with the corresponding sub-ST's derived from the transitions.

From now on, we do not distinguish between a ST and an LTS for representing a process, and use either of them when it is convenient. Furthermore, we use names "state" and "node" interchangeably. In most cases, we name a state with a ST which both labels the state and represents the process rooted at the state when it is clear from the context.

3.3.2 Derivation of NRS for Synchronization Trees

In this subsection, we derive a formula for calculating nondeterministic ripple sets for a process with synchronization tree representation. We begin with some definitions and notations.

Each synchronization tree (ST) can be reformulated as

(3.2) $\sum_{i=1,n} a_i T_i = \sum_{k=1,n_1} b_1 T_{1k} + \cdots + \sum_{k=1,n_j} b_j T_{jk} + \cdots + \sum_{k=1,n_m} b_m T_{mk}$

where $b_i \neq b_j$ if i ≠j.

Definition 3.9 (*Branch tree and Nondeterministic part*)

For the ST represented by (3.2), each $a_i T_i$ is called a ***branch tree*** of the ST. Each $\sum_{k=1,n_i} b_i T_{ik}$ is called a ***nondeterministic part*** of the ST, denoted $NP(b_i)$.

The name, *branch tree*, comes in that such a tree has only one branch emitted from its root. Note that all root branches of a nondeterministic part are labelled by a same action, and different nondeterministic parts of a ST have different actions for labelling their root branches. The name *nondeterministic part* is used because as long as the first action is chosen in a process (say it is b_i), a *nrs* of the process or the interactions between the process and its environment will be determined by this nondeterministic part and the environment choice pattern, and has nothing to do with other nondeterministic parts. This is proved in proposition 3.2. Obviously, $D(p, a)$'s, $D(p, s)$'s, and $S(p)$'s of a nondeterministic part are equal to the corresponding ones in the process from which the nondeterministic part is defined.

Examples of the ST's, nondeterministic parts, and branch trees are shown in Figure 3.5, where the ST T in (a) is

$$T = \sum_{k=1,3} b_1 T_{1_k} + \sum_{k=1,2} b_2 T_{2_k} + b_3 T_{3_1}.$$

It has three nondeterministic parts, and one of them, $NP(b_1) = \sum_{k=1,3} b_1 T_{1_k}$ is shown in (b); the branch trees of $NP(b_1)$ are listed in (c).

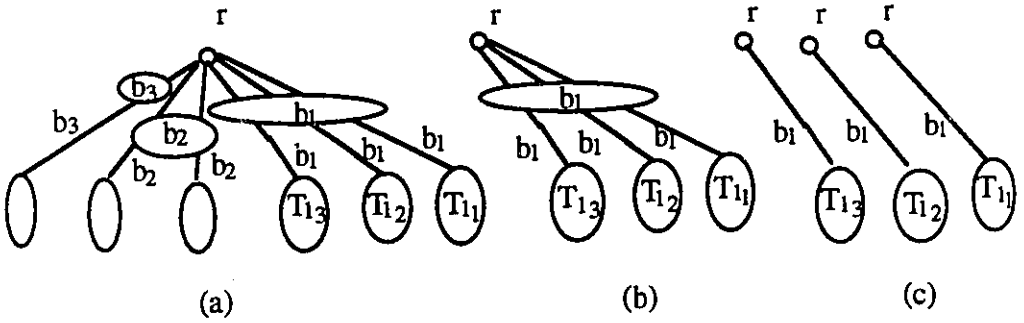


Figure 3.5. (a) a ST, (b) a nondeterministic part, (c) three branch trees

Proposition 3.2 For $T = \sum_{i=1,m} NP(b_i)$, $\mathrm{NRS}(T) = \cup_{i=1,m} \mathrm{NRS}(NP(b_i))$.

Proof: First, we prove NRS($NP(b_i)$) $\subseteq$ NRS(T). Assume that both the root of each $NP(b_i)$ and the root of T are labelled by r. Then, we have that a choice pattern for the process denoted by $NP(b_i)$ is also a choice pattern of T, i.e., $CS(NP(b_i)) \subseteq CS(T)$. This is because by definition 3.4, each $C \in CS(NP(b_i))$ is determined by the parameters r, $S(p)$, $D(p, a)$ and PS based on $NP(b_i)$, and given the $S(p)$'s, $D(p, a)$'s and PS used in the definition of C, it is obvious that these parameters are exactly the same in T and can be used to define a choice pattern of T which is the same as C for $NP(b_i)$, since $NP(b_i)$ is a nondeterministic part of T. Note that for each $C \in CS(NP(b_i))$, $(r, b_i) \in C$. Since for each $s \in Act^*$, $D(r, b_i s)$ of $NP(b_i) = D(r, b_i s)$ of T, then, given a $C \in CS(NP(b_i))$, by definition 3.5, the *nrs* of $NP(b_i)$ defined by the C is also a *nrs* of T. This proves that

$$\text{NRS}(NP(b_i)) \subseteq \text{NRS}(T).$$

On the other hand, for each $C \in CS(T)$, by the control condition of definition 3.4, we have for some $b_i \in S(r)$, $(r, b_i) \in C$. This means that for some nondeterministic part $NP(b_i)$, $C \in CS(NP(b_i))$. This is because with $(r, b_i) \in C$, the same parameters used in defining this C for T can also be obtained in the process denoted by $NP(b_i)$, and they can be used to define a choice pattern for $NP(b_i)$ which is the same as C for T. This tells us

$$CS(T) = \cup_{i=1,m} CS(NP(b_i)).$$

Finally, NRS(T) = $\{u \mid u$ is *a nrs* determined by a $C \in CS(T)\}$

$= \{u \mid u$ is *a nrs* determined by a $C \in \cup_{i=1,m} CS(NP(b_i))\}$

$= \cup_{i=1,m} \text{NRS}(NP(b_i))$. ♦

Two operations *action prefix* "." and *pointwise union* "$\mathfrak{U}$" are helpful in the following derivation. They are also used in chapter 4 for algebraic characterization of processes.

(1) Action prefix . : $a.S = \{a.s \mid s \in S\}$ where $a \in Act$, $S \subseteq Act^*$, and $a.s = as$ is the action concatenation as usual. If $S = \{S_i \mid S_i \subseteq Act^*\}$, then $a.S = \{a.S_i \mid S_i \in S\}$.

(2) Pointwise union $\mathfrak{U}$: If B_1 and B_2 are sets of sets, we define that

$$B_1 \,\mathfrak{U}\, B_2 = \{\, \beta_1 \cup \beta_2 \mid \beta_1,\ \beta_2 \in B_1,\ B_2 \text{ respectively}\}.$$

It is trivial to show that

$$(B_1 \,\mathfrak{U}\, B_2) \,\mathfrak{U}\, B_3 = B_1 \,\mathfrak{U}\, (B_2 \,\mathfrak{U}\, B_3),$$

$$B_1 \,\mathfrak{U}\, B_2 = B_2 \,\mathfrak{U}\, B_1,$$

$$a.(B_1 \,\mathfrak{U}\, B_2) = a.B_1 \,\mathfrak{U}\, a.B_2, \text{ and}$$

$$B_1 \,\mathfrak{U}\, (B_2 \cup B_3) = (B_1 \,\mathfrak{U}\, B_2) \cup (B_1 \,\mathfrak{U}\, B_3).$$

We now compute the nondeterministic ripple sets NRS(T) for a synchronization tree T based on the definitions 3.5 and 3.6.

First, we compute NRS(a_iT_i) for a branch tree a_iT_i.

Suppose that for the subtree T_i, its characterization set is NRS(T_i). Now, we compute NRS(a_iT_i). Let $nrs \in$ NRS(T_i), and the corresponding choice pattern for the nrs be C_0.(which is a choice pattern of T_i). Then, for the process represented by the branch tree a_iT_i, one of its choice patterns is $C_1 = C_0 \cup \{a_iT_i, a_i\}$, where a_iT_i stands for the node labeled by itself. This means that during execution of a_iT_i, the first action interacted is a_i, then, the process reaches node T_i. *From this point on*, the interaction will be controlled by C_0. That is, any trace in the nrs prefixed by a_i is a potential trace followed in this execution, and any trace followed in this execution must be a_i followed by a trace in the nrs. All such traces form a nondeterministic ripple set for process a_iT_i with choice pattern C_1, i.e.,

$a_i.nrs \in$ NRS(a_iT_i).

Obviously, the set of all choice patterns for a_iT_i is

$$CS(a_iT_i) = \{C_0 \cup \{a_iT_i, a_i\} \mid C_0 \in CS(T_i)\},$$

then, for a branch tree a_iT_i, we have

(3.3) $\mathrm{NRS}(a_i T_i) = \{a_i.nrs \mid nrs \in \mathrm{NRS}(T_i)\} = a_i.\mathrm{NRS}(T_i)$.

Next, we compute $\mathrm{NRS}(NP(b_i))$ for a nondeterministic part $NP(b_i) = \sum_{k=1,n_i} b_i T_{ik}$ Suppose that the characterization set for T_{ik} in each branch tree $b_i T_{ik}$ of $NP(b_i)$ is $\mathrm{NRS}(T_{ik})$. Let $nrs_{ik} \in \mathrm{NRS}(T_{ik})$ and the corresponding choice pattern be $C_{ik} \in CS(T_{ik})$. According to definition 3.4, it is known that a choice pattern of $NP(b_i)$ is

$$C_1 = (\cup_{k=1,n_i} C_{ik}) \cup (NP(b_i), b_i)\}.$$

With C_1, the first interaction of the process $NP(b_i)$ is b_i, and then all the potential nodes which may be propagated to are T_{ik} (k=1,n_i). From this point on, the potential execution of the $NP(b_i)$ can be considered separately in each T_{ik} with their corresponding C_{ik} because each T_{ik} is a separate process without nondeterministic rippling effect into other T_{ij} $j \neq i$. This execution of T_{ik} is characterized by its nrs_{ik}. That is, all the potential traces followed in this execution of $NP(b_i)$ with C_1 are $\cup_{k=1,n_i} b_i.nrs_{ik}$, i.e.,

$$\cup_{k=1,n_i} b_i.nrs_{ik} \in \mathrm{NRS}(NP(b_i)).$$

Note that the choice pattern C_1 is the union of each C_{ik}'s with the action selected at the root state of the $NP(b_i)$. The set of all choice patterns of $NP(b_i)$ will consist of the elements of all combinations of these C_{ik}'s with the action selected at the root state of the $NP(b_i)$, that is,

$$CS(NP(b_i)) = \{(\cup_{k=1,n_i} C_{ik}) \cup (NP(b_i), b_i) \mid C_{ik} \in CS(T_{ik})\ 1 \le k \le n_i\}.$$

This makes that for a nondeterministic part,

$$\mathrm{NRS}(NP(b_i)) = \{\cup_{k=1,n_i} b_i.nrs_{ik} \mid nrs_{ik} \in \mathrm{NRS}(T_{ik}),\ 1 \le k \le n_i\}.$$

By using operators $\mathfrak{U}$ and ., this can be simplified as

$$\begin{aligned}(3.4)\quad \mathrm{NRS}(NP(b_i)) &= \mathrm{NRS}(\textstyle\sum_{k=1,n_i} b_i T_{ik}) \\ &= \mathfrak{U}_{k=1,n_i} b_i\ .\mathrm{NRS}(T_{ik}) \\ &= \mathfrak{U}_{k=1,n_i} \mathrm{NRS}(b_i\ .T_{ik}) \\ &= b_i\ .\mathfrak{U}_{k=1,n_i} \mathrm{NRS}(T_{ik}).\end{aligned}$$

Finally, By proposition 3.2, we can collect all NRS($NP(b_i)$) together to obtain the characterization set for the ST T to obtain the formula in (3.5):

(3.5) $$\begin{aligned} \mathrm{NRS}(T) &= \cup_{i=1,m}\, \mathrm{NRS}(NP(b_i)) \\ &= \cup_{i=1,m}\, \mathcal{U}_{k=1,n_i} \mathrm{NRS}(b_i . T_{ik}) \\ &= \cup_{i=1,m}\, \mathcal{U}_{k=1,n_i} b_i . \mathrm{NRS}(T_{ik}) \end{aligned}$$

For the trivial tree *NIL*, its NRS(*NIL*) = $\{\{\varepsilon\}\}$.

Theorem 3.1 Given a synchronization tree T, its characterization set NRS(T) is the set computed by (3.5).

Proof that the NRS(T) computed by formula (3.5) is the NRS(T) defined in definition 3.6 is similar to the above derivation. It is omitted here.

Note that formula (3.5) is recursive. For a finite process or an approximation of an infinite process, the computation will eventually reach a leaf node labeled by *NIL* for which its NRS(*NIL*) = $\{\{\varepsilon\}\}$.

(3.5) tells us that to compute a NRS from a process, the first step is to classify its branch trees into nondeterministic parts, then, compute the NRS for each nondeterministic part from (3.4) and (3.3), and finally, collect the NRS's of all nondeterministic parts together. An example of applying (3.5) to compute the NRS for the middle ST in Figure 3.4(b) is shown below.

$$\begin{aligned} \mathrm{NRS}(a(b+c)+ad) &= \mathrm{NRS}(a(b+c))\ \mathcal{U}\ \mathrm{NRS}(ad) = a.\mathrm{NRS}(b+c)\ \mathcal{U}\ a.\mathrm{NRS}(d) \\ &= (a.(\mathrm{NRS}(b) \cup \mathrm{NRS}(c)))\ \mathcal{U}\ ad.\mathrm{NRS}(NIL) \\ &= \{\{ab\}, \{ac\}\}\ \mathcal{U}\ \{\{ad\}\} = \{\{ab, ad\}, \{ac, ad\}\} \end{aligned}$$

3.4 Deterministic Tree Representation of NRS

In this section, we introduce an alternative *deterministic tree* representation for nondeterministic ripple sets NRS(P) for a process P. This will facilitate analyzing process properties by means of NRS, and applying this semantic approach to test generation and test specification discussed in following chapters.

By definition, the NRS(P) of a process P is a collection of sets, and each of these sets again is a set of traces. We intend to arrange all traces in a $nrs \in$ NRS(P) to form a deterministic tree by sharing all longest common prefixes between the traces.

A ***prefix*** of a trace is a leading sequence of the trace. Formally, for $s \in Act^*$, s_1 is said to be a prefix of s if there is a s_2 such that $s = s_1 s_2$. Two special prefixes of a trace s are ε (the trace without any action symbols) and the trace s itself.

For the traces in a *nrs* , it often happens that a trace is a prefix of another one. If these two traces are made to share their longest common prefix in a deterministic tree, the short trace would be covered by the longer one. Thus, we need some way to distinguish between the two traces in the tree. For this purpose, we attach a distinguished symbol σ ($\notin Act$) to the end of each finite trace in the *nrs* . Then, by arranging all the traces in the set to share all their longest common prefixes, we can obtain a deterministic tree which shows all the traces in the *nrs*. The idea can be formalized as follows.

Let $nr\sigma = \{s\sigma \mid s \in nrs\}$, and

$\mathrm{NR}\sigma(P) = \{nr\sigma \mid nr\sigma$ is computed for each $nrs \in \mathrm{NRS}(P)\}$.

Let *nrt* represent the deterministic tree obtained from a $nr\sigma$ by arranging all the traces in the $nr\sigma$ to share all their longest common prefixes, and

NRT(P) = { *nrt* | *nrt* is the deterministic tree obtained from each $nr\sigma \in$ NRσ(P)}

We, now, have the following proposition.

Proposition 3.3: Given a process P and its NRS(P), a one to one correspondence exists between the elements of NRS(P), NRσ(P) and NRT(P).

Proof: The following is a one to one correspondence between the elements of NRS(P), NRσ(P) and NRT(P),

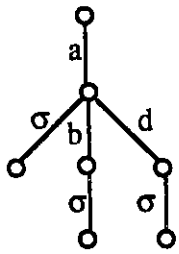
$nrs \Leftrightarrow nr\sigma \Leftrightarrow nrt$

where $nrs \in$ NRS(P), $nr\sigma \in$ NRσ(P), and $nrt \in$ NRT(P); $\Leftrightarrow$ means "corresponds to". The correspondence relations are

$nrs \Leftrightarrow nr\sigma$ iff $nrs = \{s \mid s\sigma \in nr\sigma\}$, and $nr\sigma = \{s\sigma \mid s \in nrs\}$

$nr\sigma \Leftrightarrow nrt$ iff $nr\sigma = \{s\sigma \mid s\sigma$ is a trace of *nrt*$\}$, and *nrt* is the deterministic tree obtained from $nr\sigma$.♦

We show the transformation between a $nrs \in$ NRS(P) and a $nrt \in$ NRT(P) with a simple example. Suppose that the $nrs = \{a, ab, ad\}$, then, the corresponding $nr\sigma = \{a\sigma, ab\sigma, ad\sigma\}$, and the corresponding *nrt* is shown below:



Theorem 3.2 A process P can be transformed into a set of deterministic trees, NRT(P). Each tree in the NRT(P) describes a behavior of the P resulting from the mutual influences between P's process control and environment control.

Proof: We prove by construction.

1) Compute the characterization set NRS(P) of P by Definition 3.6.

2) Transform the NRS(P) into NRT(P) by Proposition 3.3. ♦

As an example, in Figure 3.5, the deterministic tree representation NRT(P)'s for processes in (a) are shown in (c). (b) shows the corresponding NRS(P)'s of the processes. The ST graphical representation of these processes are shown in Figure 3.4. Note that in Figure 3.5(c), the symbol σ is not used to mark the end of each trace in the trees for simplicity. In the following, if the ends of the traces in a tree are clear in the context, we do not mark them for simplicity. Furthermore, we use either NRS(P) or NRT(P) of a process P to characterize its behavior when it is convenient.

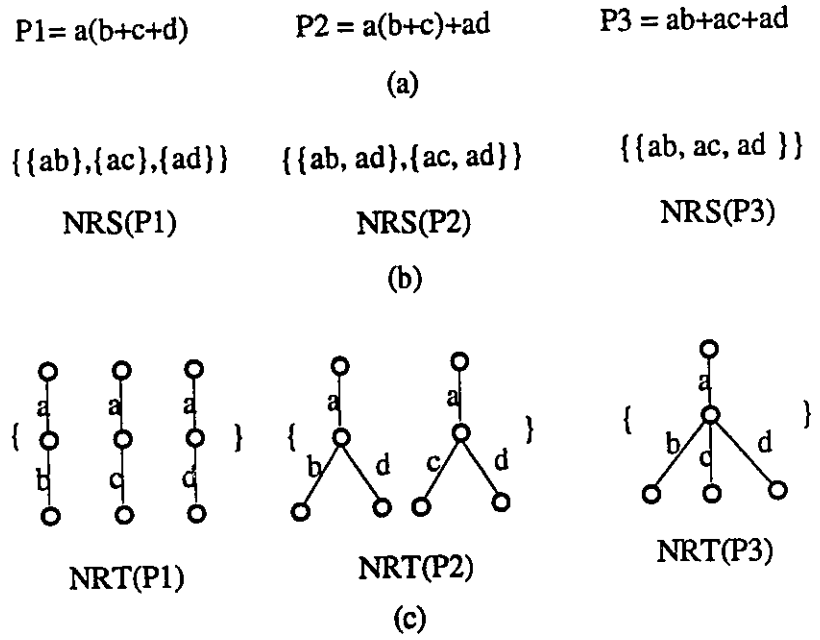


Figure 3.5 ST's and their deterministic tree characterizations

The deterministic tree representation of the characterization set of processes reflects the behavioral observation of a system from the environment side as motivated in section 2.2. Informally at the moment, each tree in a NRT(P) can be seen as a test case for an implementation of the process, if we add some verdicts such as pass or fail to each end of the trace in the tree and synchronize the tree with the process by the similar testing configuration defined in [Bri 88] or [Hen 88]. From this point of view, the characterization set of a process actually serve as a test suite for test selection, and the procedure for computing it is a test generation process. Practically, specifying test cases as deterministic trees is a popular specification method for test cases. They have been used extensively in test suite standard specifications written in TTCN [MoPr 92], for example. We will discuss these points in chapter 5 and 8 in detail.

♠ ♠ ♠ ♠

Now, we end this chapter with a summary. This chapter mainly focused on the definition of the semantic object, nondeterministic ripple sets. By accomplishing this, we defined two alternative process notations, *labelled transition systems* and *synchronization trees*, and two concepts, nondeterministic rippling effect and choice pattern for process and environment controls to a system behavior. Then by considering the mutual influences of these two controls, the major concept, *nondeterministic ripple sets*, was defined. Next, we derived an easy-to-use formula to calculate the NRS for processes represented as synchronization trees. Finally, we proved an alternative deterministic tree representation for a NRS(P) which reflects the environment observation of system behaviors.

The terms and definitions in this chapter form a basis for our development of this semantic approach. In the following chapters, we will describe various process properties based on nondeterministic ripple sets. In chapter 4, we focus on the algebraic characterization of processes based on NRS.

Chapter 4

Algebraic Characterization of NRS

The validation problem of distributed systems, as discussed at the beginning of this thesis, is to decide the equivalence(or preorder) between processes. Due to the concurrency and nondeterministic features, there are many ways to determine the equivalence between processes in a distributed system. Bisimulation [Mil 88] and failure/testing equivalences are well-known ones, for example. In this chapter, based on the nondeterministic ripple sets, we define and study a new equivalence called *Nondeterministic Rippling equivalence* .

In section 4.1, we define the NR-equivalence. Then, we devote most of this chapter to the algebraic characterization of this equivalence in sections 4.2 and 4.3. This includes an *axiom system* on processes, and a soundness and completeness *proof* of the axiom system with respect to NR-equivalence. To algebraically describe the environment and process impact on system behaviors, we introduce two auxiliary *operators* $\pm$ and $\oplus$ to model the environment control and process control, respectively discussed in the previous chapter. A *normal form* is defined for proving completeness, which is also the algebraic representation of NRS(P) for a process P. Furthermore, we also define a *DT-normal form* which algebraically represents the deterministic tree representation of the NRS(P). Finally, we present some discussion regarding this new equivalence in the concluding section.

Thus, in this chapter, most of the concepts in chapter 3 are revisited algebraically to provide a sound and complete basis for a calculus of processes based on NRS. For the basic algebraic concepts of processses, we follow the Hennessy's work [Hen 88] for a reference. Comparison of this equivalence with others appears in chapter 6.

4.1 Nondeterministic Ripple Equivalence

In this section, we define our nondeterministic ripple equivalence over the domain of synchronization tree (ST) processes defined in chapter 3, and show two examples of equivalent and non-equivalent processes based on the definition of this equivalence. In the following, we often use capital T, P, ···to represent ST processes. NRS(T) was defined in chapter 3 for any ST T.

Definition 4.1. Given processes T_1, T_2, and their characterization sets NRS(T_1) and NRS(T_2), we define a *Nondeterministic Ripple* (NR-) *Equivalence* between T_1 and T_2, denoted $=_{NR}$, as:

$$T_1 =_{NR} T_2 \quad \text{if} \quad \text{NRS}(T_1) = \text{NRS}(T_2).$$

we say that T_1 andT_2 are *nondeterministic ripple equivalent*, if $T_1 =_{NR} T_2$.

Trivially, the equivalence $=_{NR}$ is an equivalence relation. It induces a semantic interpretation for processes in that two processes are equivalent if they belong to the same equivalence class ($\in$ ST/$=_{NR}$) induced by $=_{NR}$. Here, we will use **ST** to represent the set of all processes described by a synchronization tree ST.

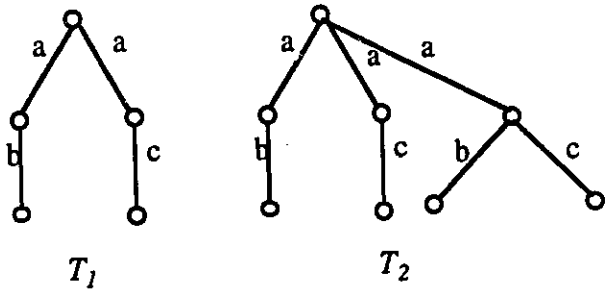


Figure 4.1. NR-equivalent processes

As an illustration of this equivalence, let us look at two processes $T_1 = ab + ac$, and $T_2 = ab + ac + a(b+c)$ in Figure 4.1. Both T_1 and T_2 have the same characterization set, $\mathrm{NRS}(T_1) = \mathrm{NRS}(T_2) = \{\{ab, ac\}\}$. Thus $T_1 =_{\mathrm{NR}} T_2$. In process T_1, the environment cannot force T_1 to follow one of the traces ab or ac, because they are nondeterministically related at the root node. In T_2, although travelling through ab or ac in its rightmost branch tree can be controlled, this is countered by the nondeterminism occuring between its two leftmost branch trees each of which consists of only one path labelled by ab or ac, respectively. This results in T_1 and T_2 having the same nondeterministic rippling behaviors. The reader may wish to confirm this by applying formula (3.5) in chapter 3 to T_1 and T_2.

For an example of non-NR-equivalent processes, we can consider again the two processes analyzed in Figure 2.1 of chapter 2, where

$$P_1 = coin.coffee + coin.(coffee+tea) + coin.(coffee+juice) \text{ and}$$

$$P_2 = coin.coffee + coin.(coffee+tea+juice).$$

P_1 is not NR-equivalent to P_2 because $\mathrm{NRS}(P_1) \neq \mathrm{NRS}(P_2)$ as shown in Figure 2.2 (intuitively, P_2 is more deterministic than P_1). Note that these two processes are failure equivalent.

We now proceed to section 4.2 to define the algebraic operators $\oplus$ and $\pm$ to enable the axiomatization of NR equivalence in section 4.3.

4.2 Operators ⊕ and ± for Algebraic Representation of Process and Environment Controls

In this section, we model the process control and environment control algebraically. This will help us to characterize NRS in an algebraic way and derive an axiom system for NR-equivalence in section 4.3.

4.2.1 Definition of Extended Processes

Nondeterministic ripple sets are based on mutual influences of *process control and environment control.* These two basic factors in the NRS semantics are now formalized in an algebraic setting. We define two *auxiliary* operators, "±" and "⊕", for this purpose. "±" and "⊕" are called *auxiliary* operators, because firstly, we do not intend to use them to represent a ST process, and secondly, they are only incorporated into processes by equations to help describe algebraic properties of processes and to enable proofs.

To use operators ⊕ and ± in a process, we first define the extended representation of processes as follows. A *process t* is now a *term* generated by the following BNF notation:

(4.0) $t ::= T \mid t \oplus t \mid t \pm t \mid a.t$

where $T \in$ **ST**, and $a \in Act$. Note that "." is still the prefix operator, but, t may use operators other than . and +. When writing terms/processes, the precedence between operators is ordered as ., +, ⊕, ±, that is, prefixing has the highest precedence, while ± has the lowest. In the following, we use small $x, y, t, \cdots$to represent terms which may contain ⊕ and ± ,and capital $T, P \cdots$for terms in **ST**. Note that from definition (4.0), a form like a + (b ⊕ c) is not a proper term/process here, since operator + is not used to connect terms containing ⊕ and ±.

Note that although we intend to discuss process properties only on the domain of ST, the following discussion also applies to all processes defined by (4.0). We can extend the equivalence definition 4.1 for $=_{NR}$ as follows:

Definition 4.1.1 For two processes x and y defined in (4.0) (not necessarily ST's defined in (3.3.0) of section 3.3), we define

$x =_{NR} y$ if NRS(x) = NRS(y).

We now explain the operations of the new operators and define the NRS interpretations of the extended processes.

-- Prefixing .

As in section 3.3, the operation of a process $a.t$ is that it first performs a transition labelled with a and then behaves like t. To compute the NRS of $a.t$, motivated by the reasoning for computing the NRS for a branch tree in (3.3), we define the following:

Definition 4.2 (NRS for $a.x$)

Given NRS(t), we define, NRS($a.t$) = a.NRS(t).

-- Operator $\oplus$ for representing process control

The operational interpretation of $x \oplus y$ is that the process can *internally* choose either subprocess x or y to execute without knowledge of its environment. Following the work of Hennessy [Hen 88], $\oplus$ is an internal nondeterministic choice operator which always introduces nondeterminism into a process. By transition rules, $x \oplus y$ is defined as:

$$x \oplus y \xrightarrow{pc} x$$
$$x \oplus y \xrightarrow{pc} y$$

where $\xrightarrow{pc}$ is a binary relation over the domain of processes. If a pair of processes $(t, t') \in \xrightarrow{pc}$, we write $t \xrightarrow{pc} t'$. $\xrightarrow{pc}$ is not labelled by an action in *Act*, and its activation is completely controlled by the process without synchronization with its environment. We referred to this phenomenon in chapter 3 as process control (*pc*) of the system behavior. The transition $\xrightarrow{pc}$ is also called an *internal transition*. Because actions in *Act* are not used in $\xrightarrow{pc}$ for process $x \oplus y$, as usual, we may use symbol "ε", the symbol representing the sequence of empty actions, to represent the action interacted in this transition. As such, the ripples at such a state labelled by $x \oplus y$ can be understood as $D(x \oplus y, \varepsilon) = \{D(x, \varepsilon) \cup D(y, \varepsilon)\}$ (ripples at a state were defined in definition 3.2). (let $D(NIL, \varepsilon) = \{NIL\}$, and $D(a.t, \varepsilon) = \{a.t\}$)

The operator $\oplus$ models the process control in the NRS interpretation of system behaviors. As defined by the transition rules of $\oplus$, process $x \oplus y$ can go to sub-process x or y by internal transition controlled by the process without knowledge of its environment. In x, suppose its NRS behavior is NRS(x), and in y, it is NRS(y). To derive the nondeterministic ripple sets for process $x \oplus y$, intuitively, it is easy to understand that each nondeterministic ripple set(*nrs*) of $x \oplus y$ is the union of a *nrs* in NRS(x) and a *nrs* in NRS(y). Thus, we define the following.

Definition 4.3 (*NRS for* $x \oplus y$) Given NRS(x) and NRS(y), we define the NRS of $x \oplus y$ as:

$$\text{NRS}(x \oplus y) = \text{NRS}(x) \,\mathfrak{U}\, \text{NRS}(y)$$

where operator $\mathfrak{U}$ is the *pointwise union* defined in section 3.3.

By the NRS formula (3.4) of chapter 3, and definitions 4.2 and 4.3, we have the following equation:

$$ax + ay =_{NR} a(x \oplus y) \qquad (+\oplus)$$

Proof:

$$\begin{aligned} NRS(ax + ay) &= NRS(ax) \,\mathfrak{U}\, NRS(ay) && \text{by (3.4) of section 3.3} \\ &= a(NRS(x) \,\mathfrak{U}\, NRS(y)) && \text{by properties of } \mathfrak{U} \\ &= a(NRS(x \oplus y)) && \text{by definition 4.3} \\ &= NRS(a(x \oplus y)) && \text{by definition 4.2} \end{aligned}$$

$ax + ay =_{NR} a(x \oplus y)$ means that transforming $ax + ay$ into $a(x \oplus y)$ does not change the NRS interpretation of $ax + ay$. Furthermore, with definition 4.3, the following equations can be proved by simple calculations (proofs omitted),

(4.1)

$$x \oplus y =_{NR} y \oplus x \qquad (\oplus 1)$$

$$(x \oplus y) \oplus z =_{NR} x \oplus (y \oplus z) \qquad (\oplus 2)$$

$$a(x \oplus y) =_{NR} ax \oplus ay \qquad (.\oplus)$$

By definition 4.3, and formula (3.4) of chapter 3, we know that in a process $x \oplus y$, x and y act like in the same *nondeterministic part of the process* (defined in section 3.3). In algebraic operations of a process, $\oplus$ is used to connect all sub-processes (like $a.t$) together in a nondeterministic part. Terms connected by $\oplus$ act are a nondeterministic part. As such, a nondeterministic part $NP(b_i) = \sum_{k=1,n_i} b_i T_{ik}$ *should be able to be* represented with $\oplus$ as

(4.2) $$NP(b_i) = \sum_{k=1,n_i} b_i T_{ik} =_{NR} \oplus_{k=1,n_i} b_i T_{ik} \; (=_{NR} b_i \oplus_{k=1,n} T_{ik})$$

This is true, since

$$\begin{aligned} NRS(NP(b_i)) &= \mathfrak{U}_{k=1,n_i} NRS(b_i . T_{ik}) && \text{by (3.4)} \\ &= NRS(\oplus_{k=1,n_i} b_i T_{ik}) && \text{by definition 4.3 and equation } (\oplus 2) \text{ in (4.1)} \\ &= NRS(b_i \oplus_{k=1,n_i} T_{ik}) && \text{by equations in (4.1)} \end{aligned}$$

In a similar way, we define operator $\pm$.

--Operator $\pm$ for representing environment control

The operational meaning of process $x \pm y$ is that during an *interaction session* between this process and its environment, the environment has the *total ability to choose* either x or y to continue. In contrast to operator $\oplus$, operator $\pm$ always describes determinism in a process and models the environment control over the process. By transition rules, $\pm$ can be defined as:

$$x \pm y \xrightarrow{ec} x$$
$$x \pm y \xrightarrow{ec} y$$

where as for $\xrightarrow{pc}$, $\xrightarrow{ec}$ is also a binary relation over the domain of processes. Activation of $\xrightarrow{ec}$ is completely controlled by the environment, i.e., environment control (ec) of system behaviors. The nondeterministic rippling effect at a state labelled with $x \pm y$ can be understood as $D(x \pm y, \varepsilon) = \{D(x, \varepsilon)\}$, or $D(x \pm y, \varepsilon) = \{D(y, \varepsilon)\}$. From the side of a process, operator $\pm$ is also considered as an external nondeterministic operator, because the process does not know which subprocess x or y the environment will choose.

According to the transition rules of $\pm$, an environment can control the process $x \pm y$ to execute subprocess either x or y. This implies that a nondeterministic ripple set of $x \pm y$ is either a *nrs* in NRS(x) or a *nrs* in NRS(y). This motivates that we define the following.

Definition 4.4 (*NRS for* $x \pm y$), Given NRS(x) and NRS(y), we define the NRS of $x \pm y$ as:

$$\text{NRS}(x \pm y) = \text{NRS}(x) \cup \text{NRS}(y).$$

Similar to operator $\oplus$, in determining NRS($x \pm y$), we did not utilize the basic NRS definitions in chapter 3. $\pm$ is also used as an auxiliary operator which is not used to specify a ST in the first place. In this limited application of operator $\pm$, the definition for NRS($x \pm y$) does not change the NRS interpretation of a process if $\pm$ is transformed into the process.

We have the following equation from formula (3.5) of chapter 3 and definition 4.4.

$$ax + by =_{NR} ax \pm by \quad \text{where } a \neq b \qquad (+\pm)$$

Proof: NRS($ax + by$) = NRS(ax) $\cup$ NRS(by) by (3.5) of section 3.3

= NRS($ax \pm by$) by definition 4.4

This means that introducing $\pm$ into $ax + by$ by the equation in (+$\pm$) does not change the NRS interpretation of $ax + by$. We can also prove the following equations by simple reasoning (proofs omitted).

(4.4) $a(x \pm y) =_{NR} ax \pm ay$ (.$\pm$)

$x \pm x =_{NR} x$ ($\pm$1)

$(x \pm y) \pm z =_{NR} x \pm (y \pm z)$ ($\pm$2)

$x \pm y =_{NR} y \pm x$ ($\pm$3)

By definition 4.4 and proposition 3.1 of chapter 3, we know that in a process $x \pm y$, x and y act like two different nondeterministic parts in the process. In algebraic operations of a process, $\pm$ is used to separate all different nondeterministic parts of a process. Terms connected by $\pm$ act like different nondeterministic parts. As such, a process, $T = \sum_{i=1,n} a_i T_i$ *should be able to* be represented as

(4.5) $T = \sum_{i=1,m} NP(b_i) =_{NR} \pm_{i=1,m} NP(b_i)$

where $NP(b_j)$ is a nondeterministic part of T, and $b_i \neq b_j$ if i $\neq$j. This is true, because

$$\mathrm{NRS}(T) = \cup_{i=1,m}\, \mathrm{NRS}(NP(b_i)) \qquad \text{by proposition 3.1}$$

$$= \mathrm{NRS}(\pm_{i=1,m}\, NP(b_i)) \qquad \text{by definition 4.4 and equation } (\pm 2) \text{ in } (4.4)$$

4.2.2 Transformation from ST's to Extended Processes

In this section, we transform a ST process, i.e., a process described by operators (., +), into the process described by operators (., $\oplus$, $\pm$).by proving several properties between these operators based on the nondeterministic ripple equivalence.

We first prove two propositions for the following reasoning.

Proposition 4.1 if $x_1 =_{NR} x_2$ and $y_1 =_{NR} y_2$,

$x_1 \pm y_1 =_{NR} x_2 \pm y_2$, $a.x_1 =_{NR} a.x_2$, and $x_1 \oplus y_1 =_{NR} x_2 \oplus y_2$

Proof: Straightforward from the NRS definitions for operators $\pm$, ., $\oplus$.

Proposition 4.2 If $T = \sum_{i=1,m} NP(b_i) =_{NR} \pm_{i=1,n} b_i y_i$, and all $b_i \neq b_j$ i ≠ j, then,

n = m, and $NP(b_i) =_{NR} b_i y_i$.

Proof: To prove that $NP(b_i) =_{NR} b_i y_i$, we have to prove for the same b_i in $\sum_{i=1,m} NP(b_i)$ and $\pm_{i=1,n} b_i y_i$, $\mathrm{NRS}(NP(b_i)) = \mathrm{NRS}(b_i y_i)$. Note that the orders of operands in $\sum_{i=1,m} NP(b_i)$ and $\pm_{i=1,n} b_i y_i$ can be transformed to be the same, i.e., b_i in $\sum_{i=1,m} NP(b_i)$ is equal to b_i in $\pm_{i=1,n} b_i y_i$, for easy reasoning. Suppose for some *nrs*, *nrs* ∈ $\mathrm{NRS}(NP(b_i))$, but *nrs* ∉ $\mathrm{NRS}(b_i y_i)$. We have for this *nrs*, *nrs* ∉ $\mathrm{NRS}(b_j y_j)$ for all j, j ≠ i, because each trace in this *nrs* starts with b_i , and it is impossible for the trace to be in any *nrs* ∈ $\mathrm{NRS}(b_j y_j)$ in which any trace starts with $b_j (\neq b_i)$. This implies that $\sum_{i=1,m} NP(b_i)$ $\neq_{NR} \pm_{i=1,n} b_i y_i$, a contradiction. The other way around, the same reasoning also applies to reason for each *nrs* ∈ $\mathrm{NRS}(b_i y_i)$, *nrs* ∈ $\mathrm{NRS}(NP(b_i))$. This proves $NP(b_i) =_{NR} b_i y_i$.

Furthermore, n = m must hold. Otherwise, if n > m, for $b_{m+1}y_{m+1}$ in $\pm_{i=1,n}b_iy_i$, we have $NRS(b_{m+1}y_{m+1}) \cap NRS(b_iy_i) = \varnothing$ for i ≠ m+1. Because $NP(b_i) =_{NR} b_iy_i$, this implies that $\sum_{i=1,m}NP(b_i) \neq_{NR} \pm_{i=1,n}b_iy_i$ and leads to a contradiction.♦

To transform a ST T into the form $\pm_{i=1,n}b_iy_i$ based on $=_{NR}$, the following steps can be used:

(4.6) $T = \sum_{i=1,m}NP(b_i)$ group operands of + in T into nondeterministic parts

$=_{NR} \pm_{i=1,m}NP(b_i)$ by (4.5)

since each $NP(b_i)) = \sum_{k=1,n}b_i.T_{ik}$

$=_{NR} \oplus_{k=1,n_i}b_iT_{ik}$ by (4.2)

$=_{NR} b_i\oplus_{k=1,n_i}T_{ik}$ by equations in (4.1)

$= b_iy_i$ let $y_i = \oplus_{k=1,n_i}T_{ik}$

Thus, $T = \sum_{i=1,m}NP(b_i) =_{NR} \pm_{i=1,m}b_iy_i$ by proposition 4.1

The following equation (4.7) is useful in the proof of the next proposition.

(4.7) $ax \oplus (by \pm cz) =_{NR} (ax \oplus by) \pm (ax \oplus cz)$ $(\oplus\pm)$

Proof: NRS($ax \oplus (by \pm cz)$)

= NRS(ax) ℧ (NRS(by) ∪ NRS(cz)) by definitions 4.3, 4.4

= (NRS(ax) ℧ NRS(by)) ∪ (NRS(ax) ℧ NRS(by)) by property of ℧

= NRS($ax \oplus by$) ∪ NRS($ax \oplus cz$) by definition 4.3

= NRS(($ax \oplus by$) ± ($ax \oplus cz$)) by definition 4.4

Transformation of a ST T into the form $\pm_{i=1,n}b_iy_i$ based on $=_{NR}$ can also be done by the following proposition.

Proposition 4.3 If $T_1 =_{NR} \pm_{j=1,n} a_j x_j$ and $T_2 =_{NR} \pm_{i=1,m} b_i y_i$, then

$$T_1 + T_2 =_{NR} \pm_{f=j_1,j_k} a_f x_f \qquad \text{for each } a_f x_f \text{ such that } a_f \neq b_i \text{ for all } b_i\text{'s.} \qquad (+\oplus\pm)$$

$$\pm \pm_{g=i_1,i_l} b_g y_g \qquad \text{for each } b_g y_g \text{ such that} b_g \neq a_j \text{ for all } a_j\text{'s.}$$

$$\pm \pm_{h=1,q} t_h \qquad t_h = c_h(x_j \oplus y_i), \text{ let } c_h = a_j = b_i$$

$$\text{for each pair } (a_j x_j, b_i y_i) \text{ such that } a_j = b_i.$$

Proof: First, we assume that all a_j's are different. If not, the equations in (4.4) can be used to combine all $a_j x_j$ terms with the same a_j together by factoring out the a_j and reach a term like $a_j z$, where $z = \pm_{i=1,l_j} x_j$. Similarly, we also assume that all b_i's are different. Let $T_1 = \sum_{j=1,n} NP(a_j)$ and $T_2 = \sum_{i=1,m} NP(b_i)$. Because $T_1 = \sum_{j=1,n} NP(a_j) =_{NR} \pm_{j=1,n} a_j x_j$ and $T_2 = \sum_{i=1,m} NP(b_i) .=_{NR} \pm_{i=1,m} b_i y_i$, by proposition 4.2, we can obtain $NP(a_j) =_{NR} a_j x_j$ .and $NP(b_i) =_{NR} b_i y_i$. Then,

$$T_1 + T_2 = \sum_{j=1,n} NP(a_j) + \sum_{i=1,m} NP(b_i)$$

$$= \sum_{f=j_1,j_k} NP(a_f) \qquad \text{for each } a_f x_f \text{ such that } a_f \neq b_i \text{ for all } b_i\text{'s.}$$

$$+ \sum_{g=i_1,i_l} NP(b_g) \qquad \text{for each } b_g y_g \text{ such that } b_g \neq a_j \text{ for all } a_j\text{'s.}$$

$$+ \sum_{h=1,q} NP(c_h) \qquad NP(c_h) = NP(a_j) + NP(b_i) \text{ for each } a_j = b_i = c_h$$

$$=_{NR} \pm_{f=j_1,j_k} NP(a_f) \pm \pm_{g=i_1,i_l} NP(b_g) \pm \pm_{h=1,q} NP(c_h) \quad \text{by (4.5)}$$

$$(4.8) \qquad =_{NR} \pm_{f=j_1,j_k} a_f x_f \pm \pm_{g=i_1,i_l} b_g y_g \pm \pm_{h=1,q} c_h(x_j \oplus y_i)$$

by propositions 4.1 and 4.2

If some x_j or y_i has the form $z = \pm_{l=1,l_j} x_l$. or $z = \pm_{e=1,l_i} y_e$, the equations in (4.7) can be used to transform line (4.8) into the same form as the result required. For example, suppose that $c_h(x_j \oplus y_i) = c_h(\pm_{l=1,l_j} x_l \oplus \pm_{e=1,l_j} y_e)$. Then,

$$c_h(\pm_{l=1,l_j} x_l \oplus \pm_{e=1,l_i} y_e) =_{NR} c_h(\pm_{e=1,l_i}((\pm_{l=1,l_j} x_l) \oplus y_e)) \quad \text{by (4.7)}$$

$$=_{NR} c_h(\pm_{e=1,l_i} \pm_{l=1,l_j} (x_l \oplus y_e)) \quad \text{by (4.7)}$$

$$=_{NR} \pm_{e=1,l_i} \pm_{l=1,l_j} c_h(x_l \oplus y_e)) \quad \text{by (4.4)}$$

This completes the proof.♦

we can obtain two corollaries to simplify the transformation between processes.

Corollary 4.3.1 (*classify a term ax into its nondeterministic part*)

If $T =_{NR} (\pm_{i=1,n} b_i y_i)$ and for all b_i's, $a \neq b_i$, then $ax + T =_{NR} ax \pm (\pm_{i=1,n} b_i y_i)$.

Proof: In proposition 4.3, let $T = T_2$ and $ax = T_1$, then

$$T_1 + T_2 =_{NR} ax \pm \pm_{i=1,n} b_i y_i \cdot \blacklozenge$$

Corollary 4.3.2 (*classify a term ax into its nondeterministic part*)

If $T =_{NR} (\pm_{i=1,n} b_i y_i)$ and for some b_i's, $a = b_i$, then $ax + T =_{NR} \pm_{i=1,n} t_i$,

where $t_i = b_i(x \oplus y_i)$ if $b_i = a$, $t_i = b_i y_i$ if $b_i \neq a$.

Proof: In proposition 4.3, let $T = T_2$ and $ax = T_1$, then

$$T_1 + T_2 =_{NR} \pm_{i=1,n} t_i \qquad t_i = b_i(x \oplus y_i), a = b_i. \blacklozenge$$

Putting corollaries 4.1 and 4.2 together, we have:

(4.9) If $T =_{NR} (\pm_{i=1,n} b_i y_i)$, then

$$ax + T =_{NR} \begin{cases} ax \pm (\pm_{i=1,n} b_i y_i), \text{ if } b_i \neq a \quad \text{for } 1 \leq i \leq n. \\ \pm_{i=1,n} t_i , \; t_i = b_i(x \oplus y_i) \text{ if } b_i = a, \; t_i = b_i y_i \text{ if } b_i \neq a \end{cases}$$

By (4.9), the transformation between processes can be carried out step by step on operands of +. We give an example to show how to transform a process $bx + az + a(by + dx) + dy$ into nondeterministic parts (terms connected ± by).

$a(by + dx) + dy =_{NR} a(by + dx) \pm dy$ by (+±)

$az + a(by + dx) + dy =_{NR} a\,(z \oplus (by + dx)) \pm dy$ by (4.9)

$bx + az + a(by + dx) + dy =_{NR} bx \pm a(z \oplus (by + dx)) \pm dy$ by (4.9)

$bx + az + a.(by + dx) + dy =_{NR} bx \pm (az \oplus a\,(by \pm dx)) \pm dy$

by (±), (+±) and proposition 4.1

$$bx + az + a(by + dx) + dy =_{NR} bx \pm a(z \oplus by) \pm a(z \oplus dx) \pm dy$$

by (4.7), (4.4) and proposition 4.1

Proposition 4.4. For a term $x \oplus y$ in the process transformed from a ST process, it is always possible to either transform the term $x \oplus y$ into a form like $a.(x' \oplus y')$, or, find a context of $x \oplus y$ like $a.[.]$.

Proof. Straightforward from proposition 4.3 and equation (4.9). An instance is shown in the above example.♦

Proposition 4.5 if $T_1 =_{NR} T_2$. and $P_1 =_{NR} P_2$, $T_1 + P_1 =_{NR} T_2 + P_2$

We omit the proof of proposition 4.5, since it is straightforward following the reasoning in the proof of proposition 4.3.

Proposition 4.1 and proposition 4.5 tell us that equivalence $=_{NR}$ is a congruence.

Finally, for convenience, we list the formulas for calculating the characterization set of processes, including process *NIL*, connected by each operator as follows.

(4.10) $NRS(a.t) = a.NRS(t)$ by definition 4.2

$NRS(x \oplus y) = NRS(x) \cup NRS(y)$ by definition 4.3

$NRS(x \pm y) = NRS(x) \cup NRS(y)$ by definition 4.4

$NRS(NIL) = \{\{\varepsilon\}\}$. by definition

$NRS(T) = \cup_{i=1,m} NRS(NP(b_i))$ by (3.5) of section 3.3

where each $NP(b_i)$ is a nondeterministic part of T

We end this section by some summary discussion.

We have introduced operators $\oplus$ and $\pm$ to model the process control and environment control in a system. Mutual influences between these two controls are reflected in algebraic transformations of processes by means of $\oplus$ and $\pm$. For instance,

$$P = a(b + c) + ad =_{NR} a(b \pm c) \oplus ad$$
$$= (ab \pm ac) \oplus ad =_{NR} (ab \oplus ad) \pm (ac \oplus ad)$$

In the transformation of process P, the process control and environment control are introduced step by step by operators $\oplus$ and $\pm$, and finally, reach a form, so called normal form in the next section, which is a result of mutual influences of these controls.

It is interesting to see that the three operators +, $\pm$, and $\oplus$ are very similar. In terms of their definitions, the functions of both $\pm$ and $\oplus$ can be replaced by + in a ST, and a process described in + can be replaced by a process described in $\pm$ and $\oplus$ under our NRS interpretation. However, their algebraic characteristics cannot replace each other. The following properties differentiate between operators +, $\oplus$ and $\pm$.

($\oplus$) $\quad ab + ac =_{NR} a(b \oplus c) =_{NR} ab \oplus ac$

($\pm$) $\quad a(b + c) =_{NR} a(b \pm c) =_{NR} ab \pm ac$

$\quad ab \oplus ac \neq_{NR} ab \pm ac$

By using operators $\oplus$ and $\pm$, we gain more insight into the structure of a process. In fact, operator + only provides a general description about the behavior of terms connected by +. With operator +, the specific behavior of a term depends on its two operands. Hoare calls this kind of operator "general choice" in his CSP [Hoa 85] (represented as []). While $\oplus$ and $\pm$ can directly describe the behavior of terms connected by them: nondeterminism or determinism. In our framework, terms connected by + can be transformed into two groups

represented by $\oplus$ and $\pm$, respectively. Concerning expressiveness, $\oplus$ and $\pm$ are exact opposite. This is illustrated in ($\oplus$) and ($\pm$), where nondeterministic term $ab + ac$ and deterministic term $a(b + c)$ can be transformed into the similar terms $ab \oplus ac$ and $ab \pm ac$, respectively. The transformation in ($\pm$) is not possible without operator $\pm$. It is also not possible in CCS or CSP. Our operator $\oplus$ is similar to the same operator in [Hen88] and the operator $\sqcap$ in [Hoa 85].

Operators $\oplus$ and $\pm$, and internal transitions $\xrightarrow{pc}$ and $\xrightarrow{ec}$ behind them represent and define the ***duality*** between process behavior and environment behavior in a system. This means that *determinism for a process is nondeterminism for its environment, and determinism for the environment is nondeterminism for the process.* For example, in process $x \oplus y$, the process knows which state between x or y will be reached after an internal transition $\xrightarrow{pc}$, but this is blind for its environment. The other way around, in process $x \pm y$, the environment can select either x or y to execute after an internal transition $\xrightarrow{ec}$, this is blind for process. This transition reflects the environment control, or user/test purpose when interacting with the process. For example, with a vending machine, *coin.tea* $\pm$ *coin.coffee* the tester knows that the function for tea can be obtained by his control, then, he selects to test/perform the function *coin.tea* . This should be possible by the definition of $\pm$ (how to implement the operator $\pm$ is a different topic.) This justifies the transition connected by $\xrightarrow{ec}$ which does not need an action for its label.

$$coin.tea \pm coin.coffee \xrightarrow{ec} coin.tea$$

Of course, in our setting, *coin.tea* $\pm$ *coin.coffee* comes from the process *coin (tea + coffee)* (= *coin (tea* $\pm$ *coffee)*), and the transition connected by $\xrightarrow{ec}$ reflects the environment control during the *whole* interaction session.

Both operators $\oplus$ and $\pm$ help us to transform from a process into its NRS characterization, and in practical situations, this helps test generation in an algebraic method. It seems that if

the two auxiliary operators are directly used for process specifications, it will be helpful for requirement formalization. For example, it helps isolate a set of functions controlled by an environment/user, or connect a set of optional functions together. This point is worth further investigating, however, further extension on this point would deviate from our main topic too far.

4.3 Axiomatization of the Nondeterministic Ripple Equivalence

In this section, we present and prove an axiom system for nondeterministic ripple equivalence $=_{NR}$.

Let a, b and c be actions in Act, x, y, z be processes, and T_1, T_2 and T_3 be in ST. A set of equations/axioms is presented in Figure 4.2. We denote this set of equations AE, and claim that AE is an axiom system for $=_{NR}$. If one process x can be proved to be equal to another process y by axiom system AE, we write $\vdash_{AE} x = y$ or $x =_{AE} y$. However, we omit the subscript AE, if it is clear from the context.

We now prove that AE is sound and complete with respect to $=_{NR}$. we prove the soundness first.

Proposition 4.6: AE is **sound** with respect to $=_{NR}$, i.e $\vdash_{AE} x = y$ implies $x =_{NR} y$.

Proof: It is enough to check that every pair of processes in each equation in AE satisfy $=_{NR}$. All these equations except equations 1, 2, 3 and 7 have been proved in section 4.2.1. Equations 1, 2, and 3 appeared in section 3.3.1. They hold because both sides of each equation represent the same tree. We now prove equation 7.

$$\begin{aligned}\mathrm{NRS}(NIL \oplus NIL) &= \mathrm{NRS}(NIL) \,\mathcal{U}\, \mathrm{NRS}(NIL) \\ &= \{\{\varepsilon\}\} \,\mathcal{U}\, \{\{\varepsilon\}\} \\ &= \{\{\varepsilon\}\} \\ &= \mathrm{NRS}(NIL)\end{aligned}$$

This shows that equation 7 holds. ♦

1	$T_1 + (T_2 + T_3) = (T_1 + T_2) + T_3$	(+1)
2	$T_1 + T_2 = T_2 + T_1$	(+2)
3	$T_1 + NIL = T_1$	(+3)
4	$a(x \oplus y) = ax \oplus ay$	(.⊕)
5	$x \oplus y = y \oplus x$	(⊕1)
6	$x \oplus (y \oplus z) = (x \oplus y) \oplus z$	(⊕2)
7	$NIL \oplus NIL = NIL$	
8	$x \oplus (y \pm z) = (x \oplus y) \pm (x \oplus z)$	(⊕±)
9	$a(x \pm y) = ax \pm ay$	(.±)
10	$x \pm x = x$	(±1)
11	$x \pm (y \pm z) = (x \pm y) \pm z$	(±2)
12	$x \pm y = y \pm x$	(±3)
13	$T_1 + T_2 = \pm_{f=j_1,j_k} a_f x_f \pm \pm_{g=i_1,i_l} b_g y_g \pm \pm_{h=1,q} t_h$	(+⊕±)
	If $T_1 = \pm_{j=1,n} a_j x_j$ and $T_2 = \pm_{i=1,m} b_i y_i$, for each $a_f x_f$ such that $a_f \neq b_i$ for all b_i's; for each $b_g y_g$ such that $b_g \neq a_j$ for all a_j's; $t_h = c_h(x_j \oplus y_i)$, for each pair $(a_j x_j, b_i y_i)$ such that $a_j = b_i (= c_h)$	

Figure 4.2 Axiom system AE for $=_{NR}$

Next, we prove the completeness of AE, that is, $x =_{NR} y$ implies $\vdash_{AE} x = y$.

To prove the completeness of AE, a general technique involving normal forms is used. The idea is to isolate a particular subclass of terms, called ***normal forms***, whose semantic denotations(here, NRS(x) for a term x), are reflected directly in their syntactic structures. This makes completeness straightforward to prove for this restricted subclass. Completeness for arbitrary terms will follow if we can show that every term can be reduced to a normal form using the equations in AE.

The normal form for each term x we will use is the algebraic representation of its NRS characterization NRS(x). Remember that each NRS(x) consists of a collection of sets, and each set ($\in$ NRS(x)) is a set of traces. The relation between the traces in each set is considered to be nondeterministic, (i.e., which of the traces is followed cannot be determined in advance). This suggests that such a nrs($\in$ NRS(x)) can be algebraically represented as a term by using the operator $\oplus$:

$$\oplus nrs = \oplus_{s_i \in nrs} s_i = s_1 \oplus s_2 \oplus \cdots \oplus s_m, \text{ for all } s_i \in nrs.$$

By (4.10), NRS($\oplus nrs$) = $\{\{s_i \mid \oplus_{s_i \in nrs} s_i\}\}$ = $\{nrs\}$

Since operator $\pm$ isolates terms into different nondeterministic parts, it can be used to connect these $\oplus nrs$ terms together, and this way, the NRS(x) for a term x can be algebraically represented as a term:

$$\pm_{nrs_j \in \text{NRS}(x)} \oplus nrs_j = \oplus nrs_1 \pm \oplus nrs_2 \cdots \pm \oplus nrs_r, \text{ for all } nrs_j \in \text{NRS}(x).$$

$$\begin{aligned}\text{By (4.10), NRS}(\pm_{nrs_j \in \text{NRS}(x)} \oplus nrs_j) &= \cup_{nrs_j \in \text{NRS}(x)} \text{NRS}(\oplus nrs_j) \\ &= \cup_{nrs_j \in \text{NRS}(x)} \{nrs_j\} \\ &= \text{NRS}(x)\end{aligned}$$

Definition 4.5 (*normal form*)

i) *NIL* is a normal form,

ii) $\oplus_{s_i \in F} v(s_i)$ is a normal form, where F is a non-empty subset of Act^*, and

$v(s_i) = s_i$ for $s_i \neq \varepsilon$, $v(\varepsilon) = NIL$

iii) $\pm_{j=1,n} x_j$ is a normal form, if all x_j are normal forms and do not contain $\pm$.

Obviously, if $x = \pm_{j=1,n} x_j$ is a normal form, according to (4.10),

(4.11) $NRS(x) = \cup_{j=1,n} NRS(x_j)$,

where $NRS(x_j\} = \{\{v(s_i) \mid \oplus_{i\geq 1} s_i = x_j, v(s_i) = s_i$, for $s_i \neq \varepsilon$ and $v(NIL) = \varepsilon\}\}$, $NRS(x_j)$ $= NRS(NIL) = \{\{\varepsilon\}\}$, if $x_j = NIL$.

Proposition 4.7: If x, y are normal forms, $x =_{NR} y$ implies $x =_{AE} y$.

Proof: This proof proceeds according to the structure of normal forms (definition 4.5)

(1). If either x or y is *NIL*, then both are. The result is immediate.

(2). Assume x has the form, $\oplus_{s_i \in F_n} v(s_i)$. Then, y must have a form like $\oplus_{sj \in F_m} v(s_j)$. According to (4.11), $NRS(x) = \{\{v(s_i) \mid s_i \in F_n\}\}$. $NRS(y).=\{\{v(s_j) \mid s_j \in F_m\}\}$. Because $x =_{NR} y$, $NRS(x) = \{\{v(s_i) \mid s_i \in F_n\}\} = NRS(y).=\{\{v(s_j) \mid s_j \in F_m\}\}$. Then, for each $v(s_i)$, there exists a $v(s_j)$ such that $v(s_i)$ = (is) $v(s_j)$. Thus, by repeatedly applying equation 5, to adjust the order of $v(s_i)$ and $v(s_j)$ in x or y, or both, we can obtain x is exactly the same as y, that is, $x =_{AE} y$.

(3). Assume x and y have the form $x = \pm_{i=1,n} x_i$ and $y = \pm_{j=1,m} y_j$ respectively. From (4.11), we have:

$NRS(x) = \cup_{i=1,n} NRS(x_j) = \{NRS(x_i) \mid x_i$ is an operand connected by $\pm$ in $x\}$ and

$NRS(y) = \cup_{j=1,m} NRS(y_j) = \{NRS(y_j) \mid y_j$ is an operand connected by $\pm$ in $y\}$.

Because $x =_{NR} y$, we have NRS(x) = NRS(y). Then, for each NRS(x_i) $\in$ NRS(x), there exists a NRS(y_j) $\in$ NRS(y) such that NRS(x_i) =(is) NRS(y_j). Now, we have x_i and y_j are normal forms, and $x_i =_{NR} y_j$. According to the reasoning in (2) of this proof, we have $x_i =_{AE} y_j$. Then, by repeatly applying equation 12 to adjust the order of x_i or y_j in x or y, or both, we can obtain x = (is) y (x is the same as y) that is $x =_{AE} y$.♦

We now show that every term can be transformed by equations in AE into a normal form. More specifically, for every x, there is a normal form, nf(x) such that $x =_{AE}$ nf(x).

Theorem 4.1 (*Normal Form Theorem*).
For every term t, there exists a normal form nf(t) such that $t =_{AE}$ nf(t).

Proof: We first prove for $T \in$ **ST**. The proof is by induction on the structures of T. There are three cases.

Case i: T is *NIL*. Then, T is a normal form by definition.

Case ii: $a.T$ has a normal form, assuming T has a normal form nf(T).

Suppose nf(T) = $\pm_{j=1,n} x_j$. Apply equation 9. We have:

$$a.T = a.\text{nf}(T) = a.\pm_{j=1,n} x_j = \pm_{i=1,n} a.x_j.$$

Then, by using equation 4, each $a.x_j$ can be transformed into a normal form. This proves $a.T$ has a normal form.

Case iii: $T_1 + T_2$ has a normal form, assuming T_1 and T_2 have normal forms nf(T_1) and nf(T_2).
First, if T_1 or T_2 is *NIL*, by equations 2 and 3,

$T_1 + NIL = T_1 = \text{nf}(T_1)$, and $NIL + T_2 = T_2 + NIL = T_2 = \text{nf}(T_2)$.

Furthermore, if some operand of + in T is NIL, suppose $T = T_1 + NIL + T_2$, and T_1 and T_2 do not contain any NIL operands, then $T = T_1 + NIL + T_2 = (T_1 + NIL) + T_2 = T_1 + T_2$ by equations 1 and 3. Now, we work on the normal form of $T_1 + T_2$.

Suppose that $\text{nf}(T_1) = \pm_{j=1,n} x_j$ and $\text{nf}(T_2) = \pm_{i=1,m} y_i$. Then, by proposition 4.4 or equations 4 and 13, we know that one action must be able to be factored out from terms connected by $\oplus$ in the process transformed from a ST process. Assume that

$T_1 = \pm_{j=1,n} a_j x'_j$, and $T_2 = \pm_{i=1,m} b_i y'_i$

by equation 13,

(4.12) $T_1 + T_2 = \pm_{f=j_1,j_k} a_f x'_f \pm \pm_{g=i_1,i_l} b_g y'_g \pm \pm_{h=1,q} t_h$

for each $a_f x_f$ such that $a_f \neq b_i$ for all b_i's; for each $b_g y_g$ such that $b_g \neq a_j$ for all a_j's; $t_h = (a_f x'_j \oplus b_i y'_i) =_{NR} c_h(x'_j \oplus y'_i)$, for each pair $(a_j x'_j, b_i y'_i)$ such that $a_j = b_i (= c_h)$.

Then, apply equation 4 to each term in (4.12) and simplify by equations 7, 10, 11, and 12. The normal form for $T_1 + T_2$ can be obtained.

For the extended processes $a.x$, $x \oplus y$ and $x \pm y$, where x and y may not belong to ST, it is not difficult in reasoning that if the normal forms of x and y is given, $a.x$, $x \oplus y$ and $x \pm y$ have normal forms. Here, the only non-trivial case is for $x \oplus y$, we prove as follows.

Suppose that $\text{nf}(x) = \pm_{j=1,n} x_j$ and $\text{nf}(y) = \pm_{i=1,n} y_i$. Then

$$x \oplus y = \text{nf}(x) \oplus \text{nf}(y) = (\pm_{j=1,n} x_j) \oplus (\pm_{i=1,n} y_i)$$
$$= \pm_{i=1,n}((\pm_{j=1,n} x_j) \oplus y_i)) \quad \text{by equation 8}$$
$$= \pm_{i=1,n}(\pm_{j=1,n}(x_j \oplus y_i)) \quad \text{by equation 8 again.}$$

This proves the theorem. ♦

We now have the following theorem for completeness of AE.

Proposition 4.8: The axiom system AE is complete with respect to $=_{NR}$,

i.e. $x =_{NR} y$ implies $\vdash_{AE} x = y$.

Proof: Suppose that $x =_{NR} y$. By theorem 4.1, we have the normal forms nf(x) and nf(y) such that $x =_{AE}$nf(x) and $y =_{AE}$nf(x). By proposition 4.6, $x =_{NR}$ nf(x), $y =_{NR}$ nf(y). This implies nf(x) $=_{NR}$ nf(y). Then, by proposition 4.7, nf(x) $=_{AE}$ nf(y). This immediately proves $\vdash_{AE} x = y$.♦

Now, we have the following soundness and completeness theorem.

Theorem 4.2 (Soundness and Completeness of AE)

The axiom system AE is soundness and completeness with respect to $=_{NR}$, i.e.,

$$x =_{NR} y \quad \text{iff} \quad \vdash_{AE} x = y.$$

Proof: Follows from propositions 4.6 and 4.8.♦

From AE, we have a derived axiom:

(4.13) $s \oplus s = s$, for $s \in Act^*$,

since $s \oplus s = s(NIL \oplus NIL) = s$

Also, we have equations (+⊕) and (+±) proved in section 4.2.1

$ax + ay =_{NR} a(x \oplus y)$, and

$bx + cy =_{NR} bx \pm cy$ for $b \neq c$ by equation 13.

An example is given to show the transformation of terms by axiom system AE.

For processes in Figure 5.1:

$P_1 = ab + ac = ab \oplus ac.$ by 13 and 4

$P_2 = ab + ac + a(b + c)$

$= ab \oplus ac \oplus a(b + c)$ by 13 and 4

$= ab \oplus ac \oplus a(b \pm c)$ by 13

$= (ab \oplus ac) \oplus (ab \pm ac)$ by 9

$= (ab \oplus ac \oplus ab) \pm (ab \oplus ac \oplus ac)$ by 8

$= (ab \oplus ac) \pm (ab \oplus ac) = (ab \oplus ac)$ by 10, (4.13)

Therefore, P_1 and P_2 are NR-equivalent by axiom system AE.

The normal form of a process gives an algebraic representation of its NRS characterization. However, this is for the trace set representation of NRS. Next, we show that the deterministic tree representation, i.e, NRT(P) of a process P, of NRS can also be algebraically represented.

First, we define a normal form for deterministic trees with a single root branch. To distinguish this normal form from the normal form defined above, we call this normal form Deterministic Tree normal form,"*DT-normal form*"

Definition 4.6 (*DT-normal form*)

i) *NIL* is a *DT*-normal form;

ii) $a.(n_1 \oplus n_2 \oplus \cdots \oplus n_m)$ is a *DT*-normal form, if $a \in Act$, and each n_i is a *DT*-normal form, and for all $i \neq j$, $n_i = b_i n'_i$ and $n_j = b_j n'_j$, $b_i \neq b_j$

If we see operator $\oplus$ as +, and transform each *DT*-normal form into its graphical representation, it is easy to see that each DT-normal form is a deterministic tree with only one root branch. Now, we show that the normal form of each process in **ST** can be represented by a set of *DT*-normal forms.

Recall that the normal form of a term x is defined as

$\pm_{nrs_j \in \text{NRS}(x)} \oplus nrs_j = \oplus nrs_1 \pm \oplus nrs_2 \cdots \pm \oplus nrs_r$ for all $nrs \in \text{NRS}(x)$

and each item, $\oplus nrs_j$ is

$$\oplus nrs_j = \oplus_{s_{ji} \in nrs_j} s_{ji} \text{ for all } s_{ji} \in nrs_j$$

For simplicity, we rewrite $\oplus nrs$ as:

$$\oplus nrs = \oplus_{s_i \in nrs} s_i = s_1 \oplus s_2 \oplus \cdots \oplus s_i, \text{ for all } s_i \in nrs.$$

Proposition 4.8: For every $nrs \in$ NRS(T), its algebraic form $\oplus nrs = \oplus_{s_i \in nrs} s_i$ can be represented by a DT-normal tree.

Proof: The proof proceeds by factoring all possible prefixes among all trace groups in $\oplus nrs$. This is done by induction on the number of traces **i** in nrs.

For i = 0, $\oplus nrs = NIL$ is a DT-normal form.

For i = 1, $\oplus nrs = s_1 = a_1 a_2 \cdots a_{n-1} a_n = a_1(a_2(\cdots a_{n-1}(a_n.NIL)$ is a DT-normal form.

For i = 2, $\oplus_{s_i \in nrs} s_i = s_1 \oplus s_2 = s_0.(s_1' \oplus s_2')$ is a DT-normal form,

since $s_0.(s_1' \oplus s_2') = a_1(a_2(\cdots a_k(s_1' \oplus s_2')))$, s_1' and s_2' are DT-normal forms with different first actions. where s_0 is the longest common prefix between s_1 ($=s_0 s_1'$)and s_2.($=s_0 s_2'$). s_0 has at least length 1 according to propcsition 4.1.

Suppose that when i = m, the result holds. At i = m + 1, we have

$$\oplus_{s_i \in nrs} s_i = (s_1 \oplus s_2 \oplus \cdots \oplus s_m) \oplus s_{m+1}$$

$$= s_0(n_1 \oplus n_2 \oplus \cdots \oplus n_m) \oplus s_{m+1} \qquad \text{By induction assumption}$$

Then, factor the longest common prefix between s_0 and s_{m+1} (at least length one by Proposition 4.1) denoted r. Two cases should be considered:

Case 1: $r = s_0$, $r.r_{m+1} = s_{m+1}$

$$s_0(n_1 \oplus n_2 \oplus \cdots \oplus n_m) \oplus r_{m+1}$$

$$= r.((n_1 \oplus n_2 \oplus \cdots \oplus n_m) \oplus r_{m+1}) \quad \text{by equation 4}$$

$$= r.(n_1 \oplus n_2 \oplus \cdots \oplus n_m \oplus r_{m+1}) \quad \text{by equation 6}$$

Suppose that there are no common prefixes between r_{m+1} and any n_i's. Then, the last step is already a *DT*-normal form. Otherwise, say n_k, $1 \le k \le m$, has a common prefix with r_{m+1},denoted as r_k i.e., $r_k.n_{k1} = n_k$ and $r_k.r_{m1} = r_{m+1}$ (note that there are no common prefixes between any n_i's by *DT*-normal form definition). Then, we have

$$r.(n_1 \oplus n_2 \oplus \cdots \oplus n_m \oplus r_{m+1})$$
$$= r.(n_1 \oplus \cdots \oplus r_k.(n_{k1} \oplus r_{m1}) \oplus \cdots \oplus n_m)$$

According to the inductive assumption, a *DT*-normal form can be derived from the last step above.

Case 2: $r.r_0 = s_0$, $r.r_{m+1} = s_{m+1}$

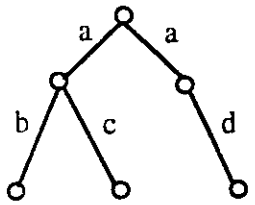
$$s_0(n_1 \oplus n_2 \oplus \cdots \oplus n_m) \oplus s_{m+1}$$
$$= r.(r_0(n_1 \oplus n_2 \oplus \cdots \oplus n_m) \oplus r_{m+1})$$

where there is no common prefix between r_0 and r_{m+1} and both $r_0(n_1 \oplus n_2 \oplus \cdots \oplus n_m)$ and r_{m+1} are in *DT*-normal form. Thus, $r.(r_0(n_1 \oplus n_2 \oplus \cdots \oplus n_m) \oplus r_{m+1})$ is a *DT*-normal form.

This proves the transformation from the from: $\oplus nrs = \oplus_{s_i \in nrs} s_i$ to a *DT*-normal form. ♦

The *DT*-normal form derived is considered as unique if we do not distinguish between the order of terms connected by $\oplus$. This is because the order of branches in a tree is not distinguished.

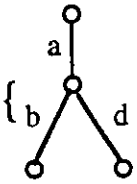
Now, we illustrate that the deterministic tree representation of NRS(P) of each process P can be algebraically represented as a set of DT-normal forms. Denote the DT-normal form of each nrs as dtn(nrs). NRS(P) of a process P can be algebraically represented as

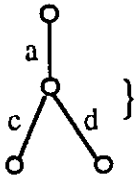


$P = a(b+c)+ad$

(a) P

NRS(P) = { {ab, ad}
{ac, ad} }

NRT(P) = { 



}

(b) NRS(P) and NRT(P)

$P = (ab \oplus ad) \pm (ac \oplus ad)$ $\qquad$ $P = a(b \oplus d) \pm a(c \oplus d)$

(c) Normal form of P and DT-normal form of P

Figure 4.3 Three kinds of representations of NRS

$$\begin{aligned} \text{NRS}(P) &= \pm_{nrs_j \in \text{ NRS}(P)} \oplus nrs_j = \oplus nrs_1 \pm \cdots \oplus nrs_j \cdots \pm \oplus nrs_m \\ &= \text{dtn}(nrs_1) \pm \cdots \text{dtn}(nrs_j) \cdots \pm \text{dtn}(nrs_m) \\ (\text{or } &= \{\text{dtn}(nrs_j) \mid nrs_j \in \text{NRS}(P)\}) \end{aligned}$$

This is shown by an example.

$$\begin{aligned} P = a(b + c) + ad &= (ab \oplus ad) \pm (ac \oplus ad) \\ &= a(b \oplus d) \pm a(c \oplus d) \end{aligned}$$

The three kinds of NRS representations of process P , trace set form, tree set form, and algebraic form, are shown in (b) and (c) of Figure 4.3 where P is in (a).

4.4 Some Discussion on the nondeterministic Ripple Equivalence

Finally in this short section, we *informally* discuss a property of nondeterministic ripple equivalence.

One unique capability of NR-equivalence is its ability to discriminate between processes of highly similar structures like x and y in Figure 4.4, where tree y possesses two identical branch trees, each the same as x. A redundant design of a specification like y can reflect some realistic considerations. For example, it may represent two communication points which are connected by two or more identical links for reliability. It may also represent a computer communication network in which one business entity has identical connections with two other business entities. Similar design considerations are common in hardware circuit design to enhance reliability and fault-tolerance.

To see why x and y are different, note that at the root node of x, there is only one possible transition for action a, while for y, two possible transitions exist. By analysis of the nondeterministic rippling between traces of process y, we see that the set $\{ab,ac\}$ is a *nrs* in NRS(y) because there is a nondeterministic choice between traces ab and ac in y, while they are deterministic in x. Thus, $x \neq_{NR} y$, i.e., the two designs are distinguished (as they should be) with respect to degree of nondeterminism.

A practical scenario of validating or testing the differences between these two designs may require turning off or cutting off one of the replicated parts in y such as the link labelled by a. In such a case, the system specified by y should still work properly under this constraint, while the system specified by x would not. For normal behavior black-box testing, the two implementations specified by x and y should test as identical. In this case,

the nondeterminism in y is not observable. In turn, the $nrs = \{ab,ac\}$ in NRS(y) cannot be considered to distinguish between x and y. However, in practice, product testing may include robustness tests which require verification of fault tolerance, and therefore, requires a NR-based or some other grey-box testing approaches [Pro 82]. (Note that the processes in Figure 4.1 are not a case of redundant design.)

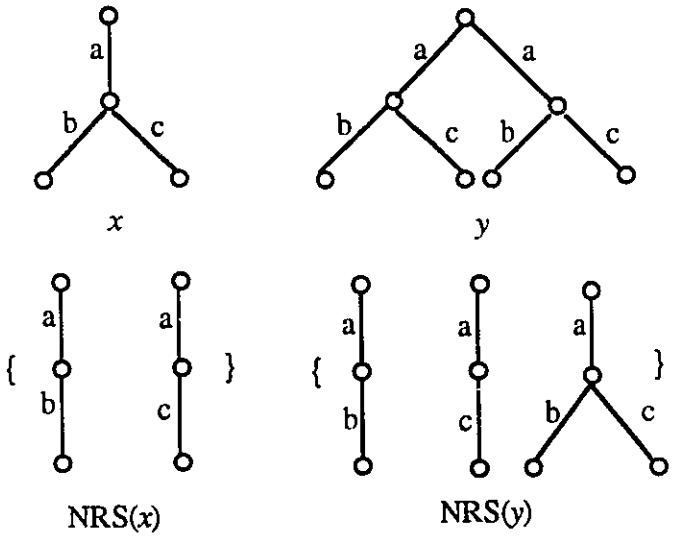


Figure 4.4. Example of non NR-equivalence $x \neq_{NR} y$

This example shows that the axiom $x + x = x$ does not hold in NR equivalence, while it holds in most of other semantic models. From the point of view of observable behaviors of a process, the axiom should be true. However, from the point of view of robustness or fault-tolerance testing, it should not be true, because if there is one unit x damaged in both sides, the two sides obviously have different behaviors. Some semantic theory should capture this practice of product design in its model.

One distinct point of our approach is considering environment control to system behaviors by using a choice pattern. Obviously, there are many different ways of controlling and observing process behaviors from environment side. The definition of the choice pattern in this work reflects only one way of doing it. Many other ways of defining the environment

control exist, and each applies some particular control constraints to a system depending on the intuitive notion of what one deems to be important about the system. Different semantic denotations can be formalized by different environment controls.

NR equivalence is made different from many other semantic approaches due to this property. Specifically, it is not comparable with the bisimulation equivalence which identifies x and y(see chapter 7). For those who prefer that equivalences should be comparable with bisimulation, we think that this is just a matter of how to define environment controls based on the basic framework in this approach. Studies on different ways of defining environment controls are another topic which is not studied in this work.

Now, we conclude this chapter. First, we defined the nondeterministic ripple equivalence($=_{NR}$). Then, most of the work was devoted to investigating the algebraic properties of this equivalence. An axiom system for $=_{NR}$ was derived. Its soundness and completeness were proved. Two new operators $\pm$ and $\oplus$ were introduced which algebraically characterize the environment and process controls in a system. Based on these two operators, the algebraic representation for NRS characterization of a process was formalized with normal forms. In summary, this section reworked most of the contents in chapter 3 in an algebraic setting of processes.

Chapter 5

Testing Processes Based on NRS

In this chapter, we study an *application* of the NRS semantics to formalization of a testing relation between processes. A testing preorder/equivalence called *NR-acceptance testing* is defined, and its distinguishing power is compared with the failure/testing equivalence [BHR 84, Hen 88] and the nondeterministic ripple equivalence defined in chapter 4.

Distinct from previous chapters, the process(or implementation) under test is viewed as a black box in this chapter, and its behaviors are observed from interactions between the process and the tests derived from the process's specification. Similar theories on testing formalization appeared in [Abr 87, NH 83, Hen 88, Phi 85]. The methods in these theories provided solid contributions to defining many critical concepts in formal description of testing which we will adopt in the following. However, we believe that an *improvement* on one aspect of these theories seems practical, that is, the set of tests used to define relations. In these methods, the set of tests used to test processes is the same for all processes and is generated from an algebra. Practically speaking, this seems unnatural because too many unrelated tests are considered. This aspect is treated differently in our testing formalization. Here, the tests are directly related to the specification of the process under test, and are generated from the NRS of the process. That is, each process x determines a set of tests, say $E(x)$, process y is said less than (or conform to) x if y satisfies every test in $E(x)$. This makes the set of tests manageable with respect to the number of tests and their structure, and meaningful for test selection and test specification with respect to test purpose/choice pattern. In chapter 8, We will further illustrate a case study which shows the direct application of NRS for test generation in practical testing, and addresses several key problems in theory and practice of conformance testing.

This chapter is organized as follows. In section 5.1, we set up the testing configuration, describe the definition of tests, and discuss their properties. Then, in section 5.2, we define a testing relation between a specification and its implementations, and prove that this relation is a preorder. Finally, in section 5.3, we locate the distinguishing power of this equivalence by comparing it with testing equivalence [Hen 88] and our NR equivalence.

5.1 Testing Configuration and Tests

--Testing configuration

A basic configuration for testing formalization was proposed in [NH 83, Hen 88]. Other similar methods were used in [Abr 87, Bri 88, Phi 85]. We shall use one which is similar to these methods except for the definition of tests.

Suppose that the implementation under test(or process under test) is a process which is described by a ST, and an observer is a user or tester of the process who applies a test to the process and observes the testing result from interactions between the process and the test. A test is also defined as a process but with extra abilities to report pass(success) or fail(failure) to the observer. The testing configuration between a process and a test is defined by an operator || which connects two processes, that is, given two processes p, q, a derivation from $p \| q$ is:

$$p \| q \text{---} a \rightarrow p_1 \| q_1 \quad \text{if} \quad p \text{---a} \rightarrow p_1 \text{ and } q \text{---a} \rightarrow q_1 \text{ for } a \in Act$$

For a process p and a test e, a ***test run*** of applying e to p is a series of derivations:

$$p \| e \text{---} s \rightarrow \quad \text{where} \quad s \in Act^*.$$

A test run is ***complete***, if $p \| e \text{---} s \rightarrow p_n \| e_n$ and $\neg \exists a \in Act$: $(p_n \| e_n \text{---} a \rightarrow)$ i.e., no further action can be derived from $p_n \| e_n$.

A complete test run $p \| e \text{---} s \rightarrow p_n \| e_n$ is ***successful***, if test e can report success at state $p_n \| e_n$. A test run ***fails***, if it is not successful. The way a test reports success will be substantiated later on when we discuss the definition of tests.

A process p passes a test e, if all possible test runs starting from $p \| e$ (or $r \| e$, if r is the root state of the process p *or* P) are successful, denoted as p ***pass*** e.

— Tests

The tests used in our method are different from others. In most of the testing equivalences, the set of tests are the same for every process and derived from some algebra. For example, in [NH 83], tests are CCS processes with a distinguished action ω to report success. Now, we take a different way, the tests are deterministic trees derived from the NRS of a process, if the implementation of this process is tested. Specifically, they are deterministic trees in NRT(P) for a process P (defined in section 3.4) with enhanced abilities to report success or failure during testing

Before defining the tests, we list some notations which are used for definitions and proofs in this and later chapters. Some of them were already defined in section 3.2, and they are relisted for easy references.

Definition 5.1 For an LTS = $(P, Act, r, \longrightarrow)$, $p, q \in P$, $s \in Act^*$,

i) $L(p) = \{s \mid p \text{---} s \rightarrow\}$, the *language* or the *set of traces* of p

ii) $D(p, s) = \{q \mid p \text{---} s \rightarrow q\}$, the s *-Ripples* or *Derivatives of* p.

iii) $S(p) = \{a \mid p \text{---} a \rightarrow\}$, the *Successors* of p, if p is a leaf node, $S(p) = \{\sigma\}$

iv) $A(p, s) = \{S(q) \mid q \in D(p, s)\}$, the *acceptance set of* p *after* s

v) $XA(p, s) = \times_{S_i \in A(p, s)} S_i$, the *cartesian product of elements in* $A(p, s)$, if $A(p, s)$ contains only one set S, let $XA(p, s) = \times S$

$$= \{(a_i) \mid a_i \in S_i\}$$

vi) $CA(p, s) = \{G(x) \mid x \in XA(p, s)\}$, *transform each tuple in* $XA(p, s)$ *into a set*

where for $x = (a_1 \cdots a_i \cdots a_n)$, $G(x) = \cup_{i=1,n}\{a_i\}$

vii) $S(p, s) = \cup_{q \in D(p,s)} S(q)$, the *Successors* of p *after* s

iix) $CT(P) = \{s \mid r \text{ —}s\text{→} q \text{ and } \neg(\exists a \in Act, q \text{—}a\text{→})\}$,

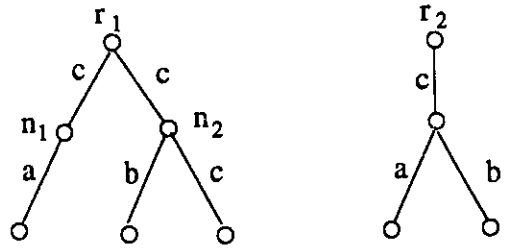
the set of *complete traces* of P

ix) In this chapter, we often refer to the elements in NRT(P) of a process P or a test. They are deterministic trees. A special notation is introduced for them.

$$S(dt(s)) = \{a \mid dt(s) \text{—}a\text{→}\},$$

where dt is a deterministic tree. Each node in a deterministic tree can be determined by the corresponding trace $s \in L(dt)$. We represent each node in a deterministic tree dt as $dt(s)$. $S(dt\ (s))$ is the set of all successive actions after node $dt(s)$. dt is usually used to denote an element in NRT(P) of a process P, or a test. Note that for each dt in NRT(P), every complete trace in dt is ended by the symbol σ according to the definition of NRT(P). the symbol σ is omitted for simplicity when it is clear from the context.

These definitions are illustrated by using the following processes P_1 and P_2 as follows:



$P_1 = ca + c(b + c)$ $\qquad$ $P_2 = c(a + b)$

$L(r_1) = \{\varepsilon, c, ca, cb, cc\}$, $D(r_1, c) = \{n_1, n_2\}$, $S(r_1) = \{c\}$, $S(r_1, c) = \{a, b, c\}$,

$A(r_1, c) = \{\{a\}\ \{b, c\}\}$, $XA(r_1, c) = \{a\} \times \{b, c\} = \{(a, b), (a, c)\}$

$CA(r_1, c) = \{\{a, b\}, \{a, c\}\}$, $S(P_2(c)) = \{a, b\}$

Now, given a process P and its NRT(P), the tests for testing the implementation specified by P is defined as:

Definition 5.2. (*Definition of tests*)

For a $nrt \in$ NRT(P) and all $s \in L(nrt)$,

(i) If $S(nrt(s)) \in CA(P, s)$, then, for all $a \in (Act - S(r, s))$,

add trace saF into the nrt.

(ii) If $S(nrt(s)) \notin CA(P, s)$, then, for all $a \in (S(r, s) - S(nrt(s)))$,

add trace $sa\sigma$ into the nrs,

for all $a \in (Act - S(r, s))$,

add trace saF into the nrs.

The nrt obtained is a test.

In the above definition, F stands for failure, σ for success, and they perform the task of reporting testing results to the observer. All tests of a process P defined for all $nrt \in$ NRT(P) are denoted as $T(p)$.

The rationale behind each of these conditions in the definition of tests can be understood by considering interactive behaviors between a test and the process under test. In derivation of tests, it is assumed that the process under test is viewed as a black box, and has no behaviors different from its specification. Each nondeterministic ripple set generated from the process's specification is controlled by a choice pattern, i.e., a test purpose. Now, we explain each of the conditions in the definition of tests.

(i) The process under test is a black box. After a trace s is interacted, the process should be in one of the states in $D(r, s)$, but, the tester or observer does not know which state it is in. Then, if $S(nrt(s)) \in CA(P, s)$, where each element in $CA(P, s)$ contains a set of necessary actions for a test run to progress, any action $a \in (Act - S(r, s))$ should not be accepted,

because it is not in any successive action sets of the states of P after trace s. If it is accepted, the process fails the test. Thus, trace *saF* is added into the *nrt*. Other actions $a \in S(r, s)$ but $a \notin S(nrt(s))$ are not considered as a failure here, because they are not included in the purpose of this test, i.e., they are not chosen by the tester. However, if these actions are interacted, they have the potential to be accepted or not.

(ii) If $S(nrt(s)) \notin CA(P, s)$, then a set of necessary actions for testing to progress is not provided after trace s in this *nrt*. At this point of testing, any actions in $S(r, s)$ have the potential to be accepted, thus, we add them to the test. If any one of these actions is accepted, a success is reported. We count this as a success, because no failures have been encountered up to now and continuation of testing along this path is not the purpose of this test. This is why the actions for extending these paths at this point are not chosen at the first place, i.e., in the generation of this nondeterministic ripple set. Any other actions $a \in (Act - S(r, s))$ should not be accepted. If they are, a failure appears.

σ is used in a *nrt* to denote the end of each trace. If a symbol σ is encountered during testing, it means that the function related to this trace has been successfully tested, and a success should be reported. Therefore, we have used symbol σ to report success. From conditions i and ii, if any other actions except in $S(r, s)$ are accepted at this point, a failure occurs.

Examples of defining tests are in Figures 5.1 and 5.2. In Figure 5.1, process P appears in (a) which has an action set $Act = \{c, a, b, d\}$, its NRT(P) is given in (b). The tests for this process is shown in (c). The calculations for deriving these tests are straightforward as follows:

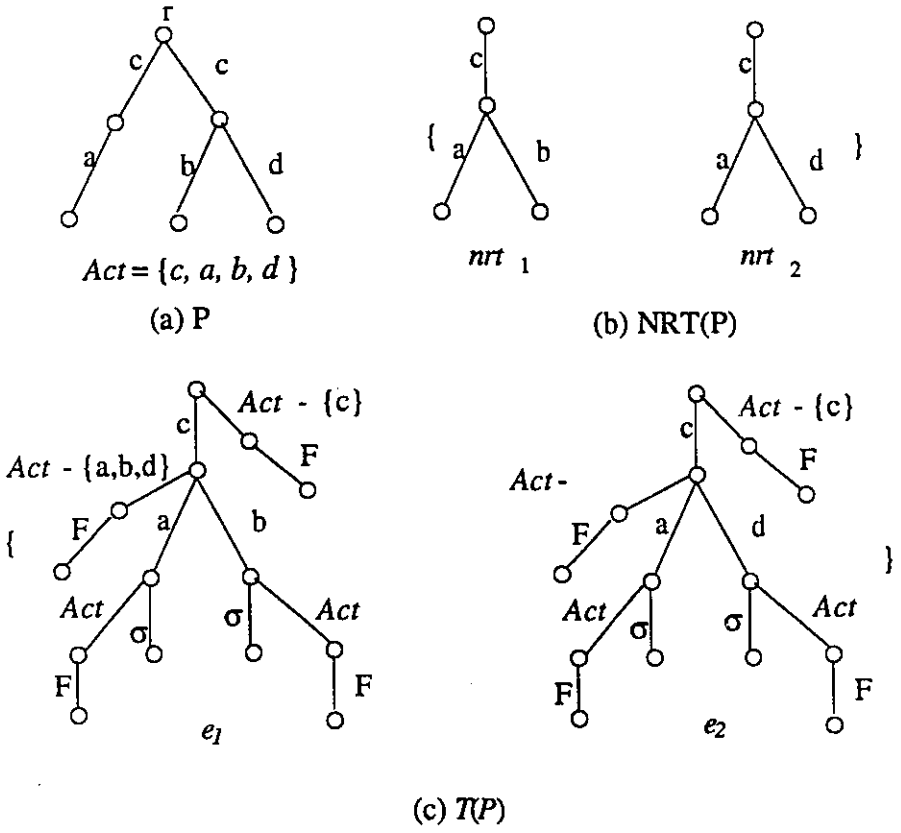


Figure 5.1 Defining tests from a process specification

In process P of Figure 5.1, $Act = \{c, a, b, d\}$, and remember that symbol σ is implied at each end of the paths in NRT(P) of (b). $A(r, \varepsilon) = \{\{c\}\}$, $CA(r, \varepsilon) = \{\{c\}\}$. In nrt_1 of the NRT(P), $S(nrt_1(\varepsilon)) = \{c\}$. Thus, $S(nrt_1(\varepsilon)) \in CA(r, \varepsilon)$, add all traces εaF, $a \in (Act - S(r, s)) = (Act - \{c\})$. For simplicity, we use only one branch labelled with $Act - \{c\}$ to represent this group of branches produced by the added traces in the first test tree of Figure 5.1(c). F after this branch represents a Failure during testing. Next, $A(r, c) = \{\{a\}, \{b,c\}\}$, $CA(r, c) = \{\{a,b\},\{a,c\}\}$, $S(r, c) = \{a,b,c\}$, $S(nrt_1(c)) = \{a,b\}$. Thus, $S(nrt_1(c)) \in CA(r, c)$, add all traces caF, $a \in (Act - S(r, c)) = (Act - \{a,b,c\})$. This is represented in the test tree as a branch labelled with $Act - \{a,b,c\}$ followed by a F. Then, for trace cr, $A(r, cr) = \{\{\sigma\}\}$, $CA(r, cr) = \{\{\sigma\}\}$, $S(r, cr) = \{\sigma\}$, $S(nrt_1(cr)) = \{\sigma\}$. Since, $S(nrt_1(cr)) \in CA(r, cr)$, add all traces caF, $a \in (Act - S(r, cr)) = (Act - \{\sigma\}) = Act$. This is represented in the test tree as a branch labelled with Act followed by a F. Similar

calculations can be done for trace *cf*. We obtain a test e_f for process P as shown in the first tree in (c) of Figure 5.1. All tests derived from the NRT(P) are listed in $T(P)$ of Figure 5.1(c).

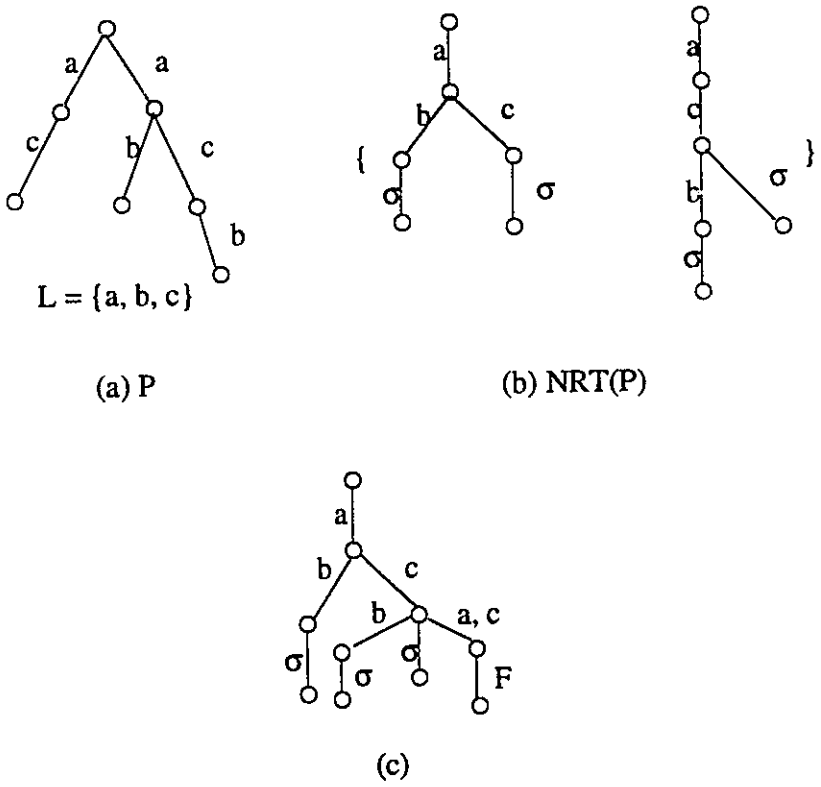


Figure 5.2 Explanation for condition ii in the definition of tests

Another example in Figure 5.2 shows the application of condition ii to defining a test for the process in 5.2(a) where $Act = \{a, b, c\}$. In the NRT(P) in (b), we have attached the symbol σ to the ends of traces in each $nrt \in$ NRT(P) for easy calculation. We take the left tree in the NRT(P) of Figure 5.2(b) to define a test for the process P in (a) and calculate the addition to this tree for trace ac only. Calculations for other traces and trees in NRT(P) are similar. For trace $ac \in L(nrt)$, (here, the nrt denotes the left tree in the NRT(P)), we have $A(r, ac) = \{\{\sigma\}, \{b\}\}$, $CA(r, ac) = \{\{\sigma, b\}\}$, $S(r, ac) = \{\sigma, b\}$, $S(nrt(ac)) = \{\sigma\}$. Thus, $S(nrt(ac)) \notin CA(r, ac)$, then, according to condition ii in the definition of tests, add trace $sa\sigma$ into the nrt, for $a \in (S(r, ac) - S(nrt(ac))) = \{b\}$, add trace saF into the nrt, for $a \in$

$(Act - S(r, ac))= \{a,b,c\} - \{\sigma, b\} = \{a, c\}$. These additions are represented in the test tree of Figure 5.2(c) as two branches labelled with b followed by σ and a,c followed by F, respectively.

For a process P, each $nrt \in$ NRT(P) leads to a test. Thus, the number of tests defined is at most the number of *nrt*'s in the NRT(P). In general, the original *nrt* part in a test is responsible for testing the ***normal valid functional part*** of the process, while the additions to the *nrt* test for the ***robustness or abnormal behaviors*** of the implementation of the process. In the format of our tests, report for a success is the following derivation

For a process p and an $e \in T(p)$,

Success: $p \| e \ \text{—}s\text{→}\ p_n \| e_n$ is complete and $e_n \text{—}\sigma\text{→}$

A failure in testing occurs, when a test run is complete and no success can be reported:

Fail: $p \| e \ \text{—}s\text{→}\ p_n \| e_n$ is complete and $\neg (e_n \text{—}\sigma\text{→})$.

The symbol F in the tests is not necessary, but it can be used to distinguish between two kinds of failure which helps with analysis of test results. Suppose that an implementation Q of process P is under test by a test e in $T(P)$. Then, these two kinds of failure are:

(i) $q \| e \text{—}s\text{→}\ q_n \| e_n$ is complete and $e_n \text{—}F\text{→}$, this means that Q has a trace s which is not in P.

(ii) $q \| e \text{—}s\text{→}\ q_n \| e_n$ is complete, and $\neg (e_n \text{—}F\text{→})$ and $\neg (e_n \text{—}\sigma\text{→})$, this means that Q has a trace s which is not a complete trace of $L(P)$, i.e. the function tested has been partially implemented.

We next prove some properties about a test *e in T(P)*. Assume that the root state of P is r.

Proposition 5.1. For each $sa \in L(e)$, and $s \in L(P)$, for some $\beta \in CA(r, s)$, $S(e(s)) \supseteq \beta$, holds.

Proof: Follows from the definition of the tests.♦

Proposition 5.2: If $s \in L(e)$ is a complete trace of P, then $e \text{---} s \rightarrow e' \text{---} \sigma \rightarrow$.

Proof: The symbol σ is attached to every complete trace in the *nrt* ($\in$ NRT(P)) from which e is defined. Thus, $e \text{---} s \rightarrow e' \text{---} \sigma \rightarrow$, if $s \in L(e)$ is a complete trace of P.♦

Note that the opposite of Proposition 5.2 is not true, that is, if $e \text{---} s \rightarrow e' \text{---} \sigma \rightarrow$, s is not necessarily a complete trace.

Proposition 5.3: For every $e \in T(P)$, *P pass e*.

Proof: *P pass e* means that for every test run, $r \| e \text{---} s \rightarrow r_n \| e_n$ is complete and $e_n \text{---} \sigma \rightarrow$. Suppose that for some test run, $r \| e \text{---} s \rightarrow r_n \| e_n$ is complete, but $e_n \text{---} \sigma \rightarrow$ is false. Obviously, $e_n \text{---} F \rightarrow$ is also false because $s \in L(P)$. From proposition 5.2, s is not a complete trace. Then, from definition 5.2, we have some $a \in Act$, $sa \in L(e)$. From proposition 5.1, $S(e(s)) \supseteq \beta$, for some $\beta \in CA(r, s)$ of P. $r \| e \text{---} s \rightarrow r_n \| e_n$ is complete means that for any $\beta \in CA(r, s)$, $S(e(s)) \cap \beta = \varnothing$. But, we have $S(e(s)) \supseteq \beta$, for some $\beta \in CA(r, s)$ Thus, $S(e(s)) \cap \beta \neq \varnothing$. A contradiction.♦

Proposition 5.4: For each $\beta \in CA(r, s)$ of a process P, there always exists a test $e \in T(P)$ such that $S(e(s)) \supseteq \beta$.

Proof: (This proof involves the basic definitions 3.4 and 3.5 in chapter 3 about selection of a choice pattern and computation of a nondeterministic ripple set.) We first choose a choice pattern C which satisfies the following condition: the action selection at each state in

C is made for process P to follow all paths labelled by a trace which has a prefix s. This condition can be simply satisfied by successively choosing actions in s at proper states in C, if it is possible. For example, let $s = a_1 \cdots a_k$, At the root state r of P, we select a_1, i.e., $(r_1 , a_1) \in C_1$. Then, a set of states $D(r_1, a_1) = \{\cdots c_i \cdots\}$ is reached. Now, we choose one action from each $S(c_i)$ according to the definition of a choice pattern. we choose a_2 , otherwise, any action in $S(c_i)$ can be chosen. Then, we reach another set of states at a deep level. By successively choosing actions in s at the proper states in C if possible until action a_k is chosen. Suppose that $D(r, s) = \{q_1, q_2, \cdots, q_n\}$. Then we can have a choice pattern

$$C = \{\cdots(q_1, b_1), (q_2, b_2), \cdots(q_n, b_n)\cdots\}, \text{ where } b_i \in S(q_i), \text{ for } 1 \leq i \leq n.$$

b_i 's are selected such that $\{b_1..b_n\} = \beta \in CA(r, s)$. This is possible from the definition $CA(r, s)$. By using this choice pattern C, and definition 3.5, it is straightforward that we have a nrt such that $S(nrt(s)) = \beta \in CA(r, s)$. Then, defining a test e from this nrt, we have the test e such that $e \in T(P)$ and $S(e(s)) \supseteq \beta$.♦

Note that for the test e constructed in the proof of proposition 5.4. if s is not a complete trace of P, σ(success) cannot be derived from e after s. Because for the nrt, $S(nrt(s)) = \beta \in CA(r, s)$, according to definition 5.2, no other traces like saσ are added to the e, and there is no symbol σ after s.

This ends our discussion about properties of tests.

5.2 NR-Acceptance Testing

In this section, we define a relation between processes by testing, and then prove that this relation is a preorder.

Definition 5.3: Given two processes P and Q, Q is said to be a conforming implementation of P by NR-acceptance testing, if for every $e \in T(P)$, Q *pass* e, denoted

$$P \geq_{nra} Q,$$

where the subscript *nra* means *Nondeterministic Ripple Acceptance testing*.

Examples of processes related by $\geq_{nra}$ are shown in Figure 5.3, where $P_1 \geq_{nra} I_1$, $P_1 \geq_{nra} I_2$, $P_2 \geq_{nra} I_2$, but, $P_2 \geq_{nra} I_1$ is not true. These can be easily checked by experimenting with the tests of P_1 and P_2 on both I_1 and I_2. For example, $\{ab\} \in$ NRS(P_2) is part of a test of P_2 for checking if the valid function ab is implemented by I_1. The test run of ab with I_1, i.e., $ab \parallel I_1 = ab \parallel ac \text{ — } a \rightarrow b \parallel c$, will derive a failure(deadlock). Therefore, the test made from $\{ab\}$ suffices to show that I_1 is not a conforming implementation of P_2.

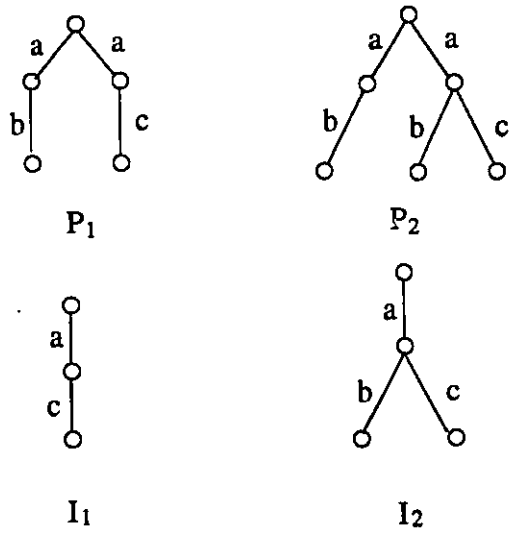


Figure 5.3 $P_1 \geq_{nra} I_1$, $P_1 \geq_{nra} I_2$, $P_2 \geq_{nra} I_2$, but $\neg(P_2 \geq_{nra} I_1)$

Now, we prove that $\geq_{nra}$ is a preorder. First, we need some auxiliary propositions.

Proposition 5.5: If $P \geq_{nra} Q$, then $s \in L(Q)$ implies $s \in L(P)$.

Proof: Suppose that $s \notin L(P)$. Then, we can find a prefix $s'a$ of s such that $s' \in L(P)$ but $s'a \notin L(P)$ (at least, s' can be ε). Choose a test $e \in T(P)$ such that $s' \in L(e)$. Note that $a \notin S(r, s')$ of process P and $s'aF$ is added to test e by the definition of this test. Since $s = s'as'' \in L(Q)$, there must be a test run $Q \parallel e \text{ — } s' \rightarrow Q' \parallel e' \text{ — a} \rightarrow$. And at this point $e \text{ — } F \rightarrow$, a failed test run. This contradicts with $P \geq_{nra} Q$. Thus, $s \in L(P)$.♦

Proposition 5.5 tells us that when $P \geq_{nra} Q$, we have $L(P) \supseteq L(Q)$

Proposition 5.6: When $P \geq_{nra} Q$, $CT(P) \supseteq CT(Q)$, where $CT(P)$ and $CT(Q)$ are complete trace sets of P and Q, respectively.

Proof. Suppose for some s, $s \in CT(Q)$, but $s \notin CT(P)$. From proposition 5.5, $s \in L(P)$. With this s, following the proof of proposition 5.4 to defining a test e, for the test e. if s is not a complete trace of P, σ(success) cannot be derived from e after s, because for the *nrt*, $S(nrt(s)) = \beta \in CA(r, s)$, According to definition 5.2, no other traces like $sa\sigma$ are added to the e, and there is no symbol σ after s. That is, any test runs of $Q \parallel e$ cannot be successful. Contradict with $P \geq_{nra} Q$. Therefore, $s \in CT(P)$.♦

Proposition 5.7: If $P \geq_{nra} Q$, $e \in T(P)$ and for each test run: $Q \parallel e \text{ — } s \rightarrow Q' \parallel e'$, then, if $\neg (e' \text{ — } \sigma \rightarrow)$, for some $\beta \in CA(r_2, s)$ of process Q, $S(e(s)) \supseteq \beta$, where the root states of P and Q are r_1 and r_2 , respectively.

Proof: To prove: $S(e(s)) \supseteq \beta$, for some $\beta \in CA(r_2, s)$ of process Q is equal to prove that for every $S(q_i) \in A(r_2, s)$, $S(e(s))$ at least contains one action in $S(q_i)$. Suppose this is not true. Then, for some $S(q_i) \in A(r_2, s)$, $S(q_i) \cap S(e(s)) = \varnothing$, that is, $S(e(s))$ does not include any successive actions after node q_i in process Q. Then, for the test run: $Q \parallel e \text{ —} s \rightarrow Q' \parallel e'$, if $\neg (e' \text{ — } \sigma \rightarrow)$, we know that $s \notin CT(P)$ from the definition of tests. And also, $s \notin CT(Q)$, otherwise, $s \in CT(P)$ according to proposition 5.6, a

contradiction. Since Q *pass* e from $P \geq_{nra} Q$, $Q \parallel e \text{---} s \rightarrow Q' \parallel e'$ has to be extended further to be successful. There must exist a test run where $Q \parallel e \text{---} s \rightarrow Q' \parallel e'$ and Q' is in state q_i. Because we have assumed $S(q_i) \cap S(e(s)) = \emptyset$, extension of $Q' \parallel e'$ by any action in $S(q_i)$ is not possible, i.e., $Q \parallel e \text{---} s \rightarrow Q' \parallel e'$ has already been a complete test run for which $\neg (e' \text{---} \sigma \rightarrow)$. This derives a failed test run and contradicts with Q *pass* e.♦

Now, we are in a position to prove that $\geq_{nra}$ is a preorder.

Theorem 5.1: $\geq_{nra}$ is a preorder.

Proof: (1) It is trivial to see that $\geq_{nra}$ is reflexive. (2) For $\geq_{nra}$ to be transitive, let $P \geq_{nra} Q$, $Q \geq_{nra} R$. We must show that $P \geq_{nra} R$, that is, for every $e \in T(P)$, R *pass* e. Assume that for some $e \in T(P)$, R *pass* e is not true. Then, there exists a failed test run $R \parallel e \text{---} s \rightarrow R' \parallel e'$ where it is complete and $\neg (e' \text{---} \sigma \rightarrow)$. The failure of this test run can be analyzed as following cases:

(i) $e' \text{---} F \rightarrow$. This implies $s \notin L(P)$. But, $s \in L(R)$, according to proposition 5.5, $s \in L(R) \Rightarrow s \in L(Q) \Rightarrow s \in L(P)$, because $P \geq_{nra} Q$, $Q \geq_{nra} R$. This is a contradiction.

(ii) $\neg (e' \text{---} F \rightarrow)$. At this case, if $R \text{---} s \rightarrow R'$ and $\neg(R' \text{---} a \rightarrow)$ for any $a \in Act$. Then, $s \in CT(R) \Rightarrow s \in CT(Q) \Rightarrow s \in CT(P)$ according to proposition 5.6. This contradicts with $\neg (e' \text{---} \sigma \rightarrow)$ according to proposition 5.2. Next, suppose that $R \text{---} s \rightarrow R'$ and $(R' \text{---} a \rightarrow)$. From this, we have $sa \in L(R) \Rightarrow sa \in L(Q) \Rightarrow sa \in L(P)$ from proposition 5.5. Because $P \geq_{nra} Q$, there exists a test run: $Q \parallel e \text{---} s \rightarrow Q' \parallel e'$ where $S(e(s)) \supseteq \beta$, for some $\beta \in CA(r, s)$ of process Q, from proposition 5.7. Now, we examine the tests in $T(Q)$. From proposition 5.4, there is a test $e_Q \in T(Q)$ such that $S(e_Q(s)) \supseteq \beta$ and $e_Q \text{---} s \rightarrow e_Q'$, $\neg (e_Q' \text{---} \sigma \rightarrow)$ because s is not a complete trace of Q. Since $Q \geq_{nra} R$, R *pass* e_Q. There is a test run: $R \parallel e_Q \text{---} s \rightarrow R' \parallel e_Q'$ where $S(e_Q'(\varepsilon)) = S(e_Q(s)) \supseteq \beta$, which can be extended further for some action a in β. Remember our goal,

we have assumed that $R \| e \text{---} s \rightarrow R' \| e'$ cannot be extended. But, because $S(e'(\varepsilon)) = S(e(s)) \supseteq \beta$, thus, for the same action a in $S(e(s))$, $R \| e \text{---} s \rightarrow R' \| e'$ should be extended. This shows that our assumption is wrong and proves that *R pass e*. This proves that the relation $\geq_{nra}$ is a preorder.♦

An equivalence can be induced by the preorder $\geq_{nra}$ as:

$$P =_{nra} Q \qquad \text{iff} \qquad P \geq_{nra} Q \text{ and } Q \geq_{nra} P.$$

We call this equivalence *Nondeterministic Ripple Acceptance testing equivalence*(or nra equivalence). This name is adopted because the tests in determining the equivalence are defined from the nondeterministic ripple sets of processes, and the information observed from testing mainly depends on acceptance(synchronization) of actions interacted between a process and its implementation.

5.3 Comparison of $=_{nra}$ with Testing and NR- Equivalences

In this section, we determine the distinguishing power of $=_{nra}$ by comparing this equivalence with two other equivalences, *testing equivalence* defined in [Hen 88] and the *nondeterministic ripple equivalence* defined in Chapter 4. We first provide a definition for the testing equivalence and then prove the relations between the three equivalences.

The equivalence and preorder we are to use from [Hen 88] are $=_{must}$ and $\geq_{must}$. The tests used in defining them are the same for every process. It is proved in [Hen 88] that for a set of tests denoted as E, $=_{must}$ and $\geq_{must}$ can be defined as

$P \geq_{must} Q$ iff for every $e \in E$, *P pass e* implies *Q pass e*.

$P =_{must} Q$ iff $P \geq_{must} Q$ and $Q \geq_{must} P$.

The set of tests E is defined as the set of all tests of the form

$$1\omega + b_1(1\omega + \cdots + b_n(1\omega + a)\ldots) \quad \text{denoted as } e(s, a)$$

$$1\omega + b_1(1\omega + b_2(1\omega + \cdots + b_n(a_1\omega + \cdots a_k\omega)\ldots) \quad \text{denoted as } e(s, A)$$

where $a \in Act, A \subseteq Act$ and $s = b_1 \cdots b_n$. 1 means that tests $e(s, a)$ or $e(s, A)$ can make a transition labelled with 1 without knowledge of the process tested. For example, $P \parallel (1\omega + b_1) \text{---} 1 \rightarrow P \parallel \omega$. Here, ω is used to report success and is same as our σ in function. They can be exchanged, but to conform to the notation in [Hen 88], we still keep it. With these definitions, it is easy to check that

(5.3.1) *P pass* $e(s, a)$ iff $a \notin S(r, s)$ of process P.

At this time, we are able to show a major result, namely:

Theorem 5.2: $P =_{\text{nra}} Q$ if and only if $P =_{\text{must}} Q$.

To prove this theorem, we first prove that $\geq_{\text{nra}}$ implies $\geq_{\text{must}}$.

Proposition 5.8: $P \geq_{\text{nra}} Q$ implies $P \geq_{\text{must}} Q$.

Proof: We must show that for every $e \in E$, *P pass e* implies *Q pass e*. It is sufficient to consider s such that $s \in L(Q)$. This also means that $s \in L(P)$ by proposition 5.5. For a test $e(s, a)$, if P pass $e(s, a)$, this means that $sa \notin L(P)$, otherwise there is a failed test run $P \parallel e(s, a) \text{---} s \rightarrow P' \parallel a$. In turn, $sa \notin L(Q)$, otherwise $sa \in L(P)$ by proposition 5.5, this implies *Q pass* $e(s, a)$ by line (5.3.1). Next, for the kind of test $e(s, A)$, if *P pass* $e(s, A)$, it means that s is not a complete trace of P, otherwise, there is a test run: $P \parallel e(s, A) \text{---} s \rightarrow P' \parallel (a_1\omega + \cdots a_k\omega)$ and $\neg (P' \text{---} a \rightarrow$ for any $a \in Act)$, that is, a failed test run. Furthermore, s is also not a complete trace of Q from proposition 5.6.

Because P *pass* $e(s, A)$, we have $A \supseteq \beta$, for some $\beta \in CA(r, s)$ of process P, otherwise, for some $S(q_i) \in A(r, s)$ of P, $A \cap S(q_i) = \varnothing$. This can derive a failed test run such as $P \| e(s, A) \text{---} s \rightarrow P' \| (a_1\omega + \cdots a_k\omega)$ where P' is in state q_i, and all a_j's are not in q_i. This is a contradiction with P *pass* $e(s, A)$. According to proposition 5.4: for each $\beta \in CA(r, s)$ of process P, there always exists a test $e \in T(P)$ such that $S(e(s)) = \beta$, and if s is not a complete trace, $e \text{---} s \rightarrow e'$ and $\neg(e' \text{---} \sigma \rightarrow)$. Moreover, because $P \geq_{nra} Q$ and $\neg$ (e' --- $\sigma \rightarrow$), for some $\gamma \in CA(r, s)$ of process Q, we have $S(e(s)) = \beta \supseteq \gamma$ from proposition 5.7.. Thus, we have $A \supseteq \beta$ and $\beta \supseteq \gamma$ for $\beta \in CA(r, s)$ of process P and $\gamma \in CA(r, s)$ of process Q. This implies that we always have

$$Q \| e(s, A) \text{---} s \rightarrow Q' \| (a_1 w + \cdots a_k w) \text{---} a_i \rightarrow Q'' \| w,$$

that is Q *pass* $e(s, A)$. This shows $P \geq_{must} Q$.♦

Now, we prove the converse direction.

Proposition 5.9: $P \geq_{must} Q$ implies $P \geq_{nra} Q$.

Proof: Assume the opposite $\neg(P \geq_{nra} Q)$ that for some $e \in T(P)$, P *pass* e, but $\neg(Q$ *pass* $e)$. That is, there exists a test run $Q \| e \text{---} s \rightarrow Q' \| e'$ where it is complete and $\neg$ ($e' \text{---} \sigma \rightarrow$). Because $s \in L(P)$, $\neg$ ($e' \text{---}$ F $\rightarrow$). Obviously, s is not a complete trace of P, otherwise, $e' \text{---} \sigma \rightarrow$. A contradiction. Let $S(e')$ = A. Make a test $e(s, A)$. We have that P ***must*** $e(s, A)$, because $s \in L(P)$ and $A \supseteq \beta$, for some $\beta \in CA(P, s)$. Since $P \geq_{must} Q$, we have P ***must*** $e(s, A)$ implies Q ***must*** $e(s, A)$. Then, for every test run in which s is followed, there must be an execution as follows:

$$Q \| e(s, A) \text{---} s \rightarrow Q' \| e'(s, A) \text{---} a \rightarrow Q' \| e''(s, A) \text{---} \omega \rightarrow$$

Otherwise, Q ***must*** $e(s, A)$ is not true. This means that for some $\gamma \in CA(Q, s)$, $A \supseteq \gamma$. Thus, we have $Q \| e \text{---} s \rightarrow Q' \| e' \text{---} a \rightarrow$ for some $a \in S(e')$ (= A). This means that the assumption is false. We have $P \geq_{must} Q$ implies $P \geq_{nra} Q$.♦

Combining Propositions 5.8 and 5.9, we proved the Theorem 5.2, namely:

$P =_{nra} Q$ if and only if $P =_{must} Q$.

i.e., $=_{nra} Q$ and$P =_{must}$ coincide.

In [Nic 87, Hen 85], equivalence $=_{must}$ and equivalence $=_{test}$ were proved to coincide, and moreover, these two equivalences are proved in [Nic 87] to coincide with the failure equivalence in [BHR 84] for strongly convergent processes. Thus, the equivalence $=_{nra}$ also coincides with failure equivalence in our setting. (A *strongly convergent process* is a process which does not have infinite internal transitions.)

Next, we present a relationship between $P =_{NR} Q$ and $P =_{nra} Q$. Its proof appears in section 6.2.of Chapter 6.

Theorem 5.3: $=_{NR} \neq =_{nra}$

Proof: Since we proved that $=_{nra}$ coincides *with* $=_{must}$ in theorem 5.2. This proof is done by means of the denotational model, acceptance trees ([Hen88]) of $=_{must}$, and appears in Chapter 6.

We, finally in this chapter, present some remarks on the axiom $x + x = x$.

By theorem 5.2, it is clear that $x + x = x$ is true for $=_{nra}$, since it is true for $=_{must}$. However, in Chapter 4, we showed that it is not true for $=_{NR}$. This also shows the difference between $=_{NR}$ and $=_{nra}$.

♠ ♠ ♠ ♠

In conclusion, this chapter investigated the application of the NRS semantics to formalization of a relation between processes by testing. We first defined the testing configuration and tests in section 5.1. In section 5.2, a testing relation was defined and proved to be a preorder. The equivalence induced by this preorder was called Nondeterministic Ripple Acceptance testing equivalence ($=_{nra}$). In section 5.3, the distinguishing power of $=_{nra}$ was studied by comparing it with the well-known testing equivalence $=_{must}$ and our new equivalence $=_{NR}$. We proved that the distinguishing power of $=_{nra}$ coincides with testing equivalence $=_{must}$, but different from $=_{nra}$. The advantage of nondeterministic ripple acceptance testing over others is that the tests used are related to each specific process under test, and this makes the set of tests considered in testing is practically manageable and meaningful. Moreover, the tests can be automatically generated from our NRT definition in section 3.4.

In general, this chapter focused on the theoretical formalization of testing. Practical issues of testing are discussed in chapter 8.

Chapter 6

Evaluation of NRS Approach

In this chapter, we *compare* our NRS approach with other major semantic theories.

The basic algebraic method adopted in our NRS semantics has been a common one for modeling distributed communicating systems. Several behavioral views have been proposed for interpreting interactions in a distributed system, and many semantic models (as surveyed in section 1.2) were proposed to describe relationships between processes based on the process algebraic methods. At the present, the existing approaches are still being experimented on their suitability for application in theoretical and practical areas of distributed systems.

The existing approaches or models can be roughly classified into two well-developed behavioral views, or two schools of thought [CH 93] according to the ways they observe process behaviors. We briefly recall them here.

i) Bisimulation behavioral view of processes [Mil 80, HM 80, Mil 88]. It defines equivalences or preorders in terms of recursively matched behaviors between the corresponding pair of states of processes compared. Process models based on this view have been well-studied and are mathematically elegant. However, two main arguments on bisimulation behavioral view are that (1) the relations induced by this notion are often too discriminating; two processes may not be related even though there is no practical way to distinguishing between them [CH 93]. (2) It is difficult to have a semantic object for each process in bisimulation models, and they only characterize processes by equations, i.e.,

term models [Hen 85]. To some extent, these two points limit the applications of the models in this behavioral view to practical situations.

ii) Failure/Acceptance behavioral view of processes. This view imagines that a process and its environment(observer) interact through an interaction point, and a semantic object is defined based on the process's refusals or acceptances of action sets offered by its environment. Then, The equivalences or preorders between processes are defined by comparing these sets through set-theoretic methods. This behavioral view is relatively close to the practical interactions between a process and its environment, and has led to a well-developed mathematical theory. The most representative models in this group include failure equivalence [BHR 84] and acceptance trees [Hen 85]. The fundamental concepts about testing have been studied in the models of this group such as testing equivalence and acceptance trees [NH 83, Hen 88]. However, for practical consideration, these models still need to be improved. It is difficult to attach practical meanings to the semantic objects and tests used in these models, in particular, for test generation and test specification.

We shall mainly compare the NRS approach with these two behavioral views, and discuss the improvements made by this approach on the *points* concerned above.

The chapter is organized as follows. In section 6.1, we make the comparison in general from several aspects. This includes behavioral views, semantic objects, characterization methods, and applicability. In section 6.2, we focus on the comparison of distinguishing powers between the NR-equivalence ($=_{NR}$) and other equivalences. The main effort is devoted to proving the implication between the NR-equivalence and the equivalence determined by acceptance trees.

6.1 Comparison With the Two Major Behavioral Views

In this section, we compare our NRS approach with the two behavioral views with respect to four aspects, behavioral views, semantic objects, characterization methods, and applicability.

i) NRS semantic approach is based on a *new behavioral view*

In parallel to the bisimulation and failure/acceptance behavioral views, NRS semantic approach is based on a new behavioral view. This view considers a system behavior to be the effect of *mutual influences* between a process and its environment, and describes the natural cooperation and control between the two sides of a system during interactions.

Similar to the bisimulation view, the NRS method starts with the operational interpretations of a process, and in the same way as failure/acceptance view, it also imagines that a process and its environment interact through an interaction/observation point. All three views have sufficient account into the nondeterminism inside a process according to their own standpoint. However, different from other two views, NRS view takes special account into the *environment activity* and its *interplay* with the nondeterminism of the process. Furthermore, this view is directly motivated from the practical experience of testing telecommunication protocols. Thus, in general, NRS semantic view more closely and practically describes interaction behaviors in a system. it appears that the mathematical properties similar to the ones in other two views can also be derived for the NRS approach, (as shown in chapters 3 and 4, but the mathematical development in this theory is only done to a limited extent at present.) In addition, the practical applicability of NRS semantics may be the *best* within the three. This point will be discussed later in this section and chapter 8.

ii) Semantic objects

By comparison of semantic objects, the NRS semantics is closer to the failure/acceptance behavioral view. Recall that in this view, the observer/environment offers actions to the process and records his observations from the process as its behaviors [OH 86]. A process behavior is usually recorded as:

(6.1) *A sequence of actions s accepted by the process + process behavioral expectation X after the sequence.*

This behavioral observation is usually represented as <s, X>. The process behavioral expectation X is formalized as a set which represents the set of actions which may be refused or accepted at the present state of the process. Refusal set [BHR 84], Acceptance set [Hen 88], and Ready set [OH 86] are instances of expectation X, for example.

To see the relation between the NRS semantic observation *nrs* and the observation (s, X) in (6.1) when we set X to the acceptance or ready set, the following can be observed with the three aspects of interactions in testing mentioned in section 2.1 in mind.

We know that the behavior of a process can not be decided by either process or its environment alone. The actual behavior is the result of the *mutual influences* between actions of the process and its environment due to the purpose of a tester and the nondeterministic transitions inside the process. This determines a specific path or trace to be followed in an execution session. However, from the tester's (environment) point of view, given his test purpose, he does not know which path or trace will be definitely followed in advance because of nondeterminism of the process. Relating this to the observation object (s, X), we can extend it from three directions:

First, suppose that the test purpose is to show that *the trace s can be accepted by the process*, but during execution, before s is accepted, a nondeterministic transition may happen and steer the process into another path which is labeled by only a prefix of s . In this case, at least two traces have the potential to be followed.

Secondly, suppose that trace s is accepted during interaction. In (s, X), X is only a behavioral expectation at one state which can be reached by acceptance of s. But, due to nondeterminism of the process, the process may be in any one of the states which may be reached after s. To expect the behavior of the process to continue the execution, the tester has to coordinate this situation by offering one action at any one of these potential states. This results in a set of potential traces to be accepted. Compared with (s, X), we may consider that we deal with a ***global*** situation, while X only expects a ***local*** one.

Finally, according to testing concerns, the execution has to be continued until an exit point of the process is reached. This means that we have to observe a process behavior beyond the state with expectation X and at all other potential states encountered.

The extension of (s, X) in these three directions transforms it into a *nrs*. Thus, it seems unnatural that in the observation (s, X), trace s should be able to be accepted by a nondeterministic process in an execution or test run when the tester wants to observe the behavior (s, X). This may show why NRS observation is more practical in modelling interactions between processes in a distributed system.

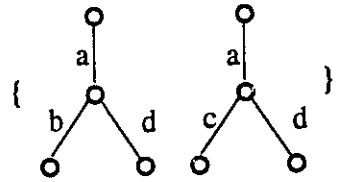
iii) Characterization methods

NRS approach has a relatively simple, intuitive, and practical representation of behavioral characterization of processes. Unlike bisimulation semantics, NRS semantics has a

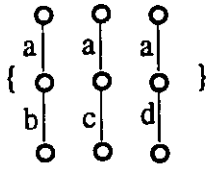
behavioral representation which is totally independent of the operational definition. Similar to Failure/Acceptance semantics, NRS semantics has both a deterministic tree representation NRT(P) for a process P which is similar to the acceptance trees in the Acceptance model [Hen 85, Hen 88], and a set representation NRS(P) which is similar to the Failure set in Failure model [BHR 84].

However, unlike these models, representations for NRS semantics are *simple* and *intuitive* for easy understanding and practical application in realistic validation activities. To be specific, the simple and intuitive nature can be illustrated in Figure 6.1 where (a) represents the NRS semantic representation of processes $P = ab + ac + ad$, $P1 = a(b + c + d)$, $P2 = a(b + c)$ and $P3 = ab + ac$; (b) and (c) are semantic representations of these processes in Acceptance trees and Failure sets. It can be seen in (b) and (c), either a large number of elements or a complex structure in their representations appear for these four simple processes. For example, 66 elements in F(P) while only 2 in NRS(P). These representations are mainly meaningful from a mathematical point of view. it is difficult to attribute a realistic meaning such as verification/testing purpose to them for practical applications besides their formal mathematical applications. This is overcome in NRS representations.

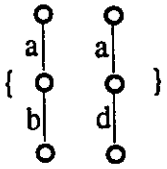
In Figure 6.1(a), in addition to the NRS's role as a semantic representations of processes, practical functional meanings can be easily attached to trees in each NRS. For example, each element in the NRS can be considered to test the functions represented by the complete traces in it. We attribute this simplicity of NRS approach to its proper formalization of interactions in a system.



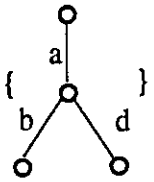
NRS(P)={{ab,ad},{ac,ad}}



NRS($P1$)
={{ab},{ac},{ad}}



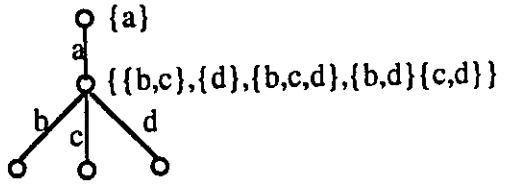
NRS($P2$)
={{ab},{ad}}



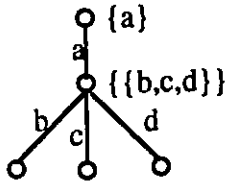
NRS($P3$)
={{ab,ad}}

(a) NRS semantic interpretations of processes $P = ab + ac + ad$,

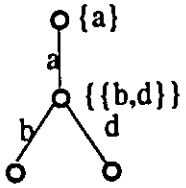
$P1 = a(b + c + d)$, $P2 = a(b + c)$ and $P3 = ab + ac$



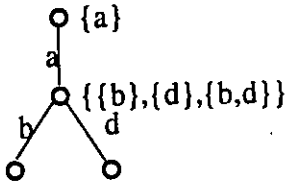
AT(P)



AT($P1$)



AT($P2$)



AT($P3$)

(b) Acceptance Tree(AT) semantic interpretations of P, $P1$, $P2$ and $P3$.

F(*P*)={ε{b,c,d}, a{}, a{a,d}, a{a,c,b},....., ad{a,b,c,d}}.

#F(*P*)=66.

F(*P1*)={ε{b,c,d}, a{}, a{a}, ab{a,b,c,d},.....ad{a,b,c,d}}.

#F(*P1*)=58.

F(*P2*)={ε{b,c,d}, a{}, a{a,c}, ab{a,b,c,d},.....ad{a,b,c,d}}.

#F(*P2*)=44.

F(*P3*)={ε{b,c,d}, a{}, a{a,c,d}, a{a,b,c},.....ad{a,b,c,d}}.

#F(*P3*)=51.

(c) Failure set interpretations of *P*, *P1*, *P2*, and *P3* in failure semantics.
#F(X) stands for the number of elements in set F(X).

Figure 6.1. Comparison of NRS with AT and Failure set.

iv) Applicability

Different from other semantic views, NRS semantic approach was directly abstracted from the interactive phenomenon in testing. Thus, application of this theory back to real situations appears promising. According to the elements used in the formalization of this approach, we can *anticipate* three aspects which are helpful with *practical* testing.

1) We believe that this approach is the first to formally relate test purposes to tests in a formal setting. Thus, it helps deal with test plan, test generation, and test management more formally and systematically. The current practice for this point is informal and manual, and largely depends on the ability of the persons concerned.

2) This approach enables the tests used to be directly related to the process under test and makes testing theory more closely match the practical requirements. In practical situations, tests used are always related to the process under test.

3) Representation of the semantic object, NRS, supports practical test specification such as by using international test specification notation TTCN [MoPr 92].

These points will be discussed in chapter 8, when we further consider the practical aspects of the NRS approach.

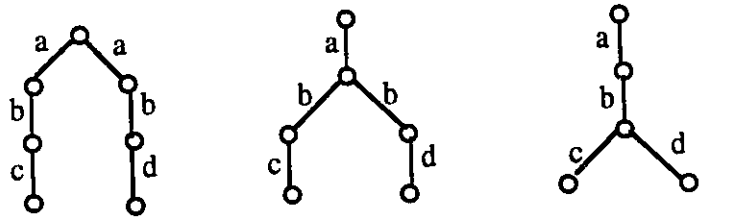
Finally in this section, we feel that the more important contribution of the NRS approach is its basic semantic view and framework, not the equivalence such as done in chapter 4 which is just an algebraic exercise. This is because there are many ways to consider and formalize the environment controls to a process in a system. Our work only considers one case of this view. By changing the definition of choice patterns or designing more proper environment control, we think that more proper semantic interpretations can be formalized to describe interactive behaviors in a distributed system more precisely and closely.

6.2 Comparison of Distinguishing Powers

In this section, we continue to compare NRS semantics with other semantic approaches, but, the comparison now is based on precisely reasoning on their distinguishing powers between processes. We prove that the nondeterministic ripple equivalence ($=_{NR}$) has no implication relation with bisimulation and several other equivalences, and it is different from failure/testing equivalence, but we conjecture that it implies this equivalence.

we first prove that there are no implication relations between nondeterministic ripple equivalence $=_{NR}$ and bisimulation, refusal testing, ready trace and readiness equivalences. For these equivalences, we refer to [Mil 88, Phi 85, BBK 87, OH 86] for their definitions.

Proposition 6.1: There is no implication relation between bisimulation equivalence($=_{bi}$) and nondeterministic ripple equivalence($=_{NR}$), i.e. $=_{bi} \not\Rightarrow =_{NR}$ and $=_{NR} \not\Rightarrow =_{bi}$.



P_1 P_2 NRS of P_1, P_2

Figure 6.2 $P_1 =_{NR} P_2$ but, $P_1 \neq_{bi} P_2$

For the proof of proposition 6.1, the non-implication from $=_{NR}$ to $=_{bi}$ can be shown by the example in Figure 6.2, where for processes $P_1 = abc + abd$ and $P_2 = a(bc + bd)$, $P_1 =_{NR} P_2$ since NRS(P_1) = NRS(P_2) = $\{\{abc, abd\}\}$, but $P_1 \neq_{bi} P_2$,. Their non-bisimulation equivalence can be easily reasoned following bisimulation definition in [Mil 88, Pnu 85]. These two processes are also well-known examples for showing that bisimulation equivalence is too strong in distinguishing between processes which are difficult to be practically distinguished. On the other hand, the non-implication from $=_{bi}$ to $=_{NR}$, can be shown in Figure 4.4 in chapter 4, where for processes x and y, $x =_{bi} y$ but $x \neq_{NR} y$.

For comparisons with refusal testing equivalence, ready trace equivalence and readiness equivalence, the results can be summarized in the following proposition:

Proposition 6.2: There are no implication relations between nondeterministic ripple equivalence and refusal testing equivalence($=_r$), ready trace equivalence($=_{rt}$) and readiness equivalence($=_{rd}$).

Proof: The following processes show the non-implication relations between $=_{NR}$ and $=_r$, $=_{rt}$, and $=_{rd}$.

From $P_1 = a(bc+c)+a(ce+b+c)$ and $P_2 = a(bc+b+c+ce)$

we have $P_1 \neq_{NR} P_2$ but $P_1 =_r P_2$

$P_1 \neq_{NR} P_2$ but $P_1 =_{rt} P_2$

$P_1 \neq_{NR} P_2$ but $P_1 =_{rd} P_2$

From $P_3 = aa+a(a+b)+a(a+b+c)$ and $P_4 = aa+a(a+c)+a(a+b+c)$

We have . $P_3 =_{NR} P_4$ but $P_3 \neq_r P_4$

$P_3 =_{NR} P_4$ but $P_3 \neq_{rd} P_4$, and

from $P_5 = ab+ac$ and $P_6 = ab+ac+a(b+c)$

we have $P_4 =_{NR} P_6$, but $P_4 \neq_{rt} P_6$.

Calculation of the NRS's of these processes is straightforward. Therefore, we omit it.♦

Next, we show that $=_{NR}$ is different from $=_{must}$. This is done by using the denotational model of $=_{must}$: acceptance trees. In [Hen88], Hennessy defined an acceptance tree model. each process is mapped to one acceptance tree. Two processes are equivalent if they have the same acceptance tree. $=_{must}$ is the operatioanl model of and coinsides with $=_{sat}$. We review some notations first. Acceptance tree equivalence is denoted $=_{sat}$ in the following.

Recall that in a deterministic rooted tree, a node of the tree can be uniquely determined by the corresponding trace in the tree. Let the trace set of a process P be $L(P)$. A deterministic

tree can be formed from the $L(P)$ by arranging the traces into the tree such that each node in the tree can be uniquely determined by a trace in $L(P)$. We denote this tree dt(P). Now, we define the acceptance tree and the saturated acceptance tree for a process.

Definition 6.1: Given a process P , its Acceptance Tree AT(P) is defined as.

i) Form the dt(P) of process P.

ii) For each $s \in L(\text{dt}(P))$, label the node determined by s in dt(P) with the set:

$A(r, s) = \{S(q) \mid q \in D(r, s)\}$, the *acceptance set of P after s*

where r is the root of P, and $A(r, s)$ means the *acceptance set of P after s* as defined in definition 5.1. The definition of $A(r, s)$ is the same as in [Hen 88], but each $A(r, s)$ is **not saturated** here.

Definition 6.2: Saturated Acceptance Tree SAT(P)

A Saturated Acceptance Tree SAT(P) is an acceptance tree AT(P) in which each node is labelled with the *saturated* acceptance set: sat($A(r, s)$).

A ***saturated set*** is defined as: for a process P, suppose $\mathcal{A}$ is a nonempty subset of $\mathcal{P}(Act)$ (power set of Act) and $S = \cup_{A_i \in \mathcal{A}} A_i$. Then, the set $\mathcal{A}$ is saturated with respect to S, if it satisfies:

i) $S \in \mathcal{A}$

ii) $X \subseteq Y \subseteq S$, $X \in \mathcal{A}$ implies $Y \in \mathcal{A}$.

We represent a *saturated set* $\mathcal{A}$ by *sat*($\mathcal{A}$).

The definitions of these two trees can be simply represented as:

$\text{AT}(P) = \{\langle s, A(r, s)\rangle \mid s \in L(\text{dt}(P))\}$

$\text{SAT}(P) = \{\langle s, sat(A(r, s))\rangle \mid s \in L(\text{dt}(P))\}$.

The acceptance tree equivalence is defined as:

$P =_{sat} Q$ if SAT(P) = SAT(P).

Proposition 6.3: $=_{NR} \neq =_{sat}$

Proof: We take two processes to show that they are $=_{sat}$ equivalent, but not $=_{NR}$ equivalent. Let P = c(t + bf) + c(f + bt) and Q = c(t + bt) + c(f + bf). Their ST representations are shown in Figure 6.3(a), and their acceptance trees and NRS characterizations are shown in Figure 6.3(b) and (c), respectively. As shown in Figure 6.3, $P =_{sat} Q$ but $P =_{NR} Q$. That is, $=_{NR} \neq =_{sat}$.♦

Accordingly, we have $=_{NR} \neq =_{must}$

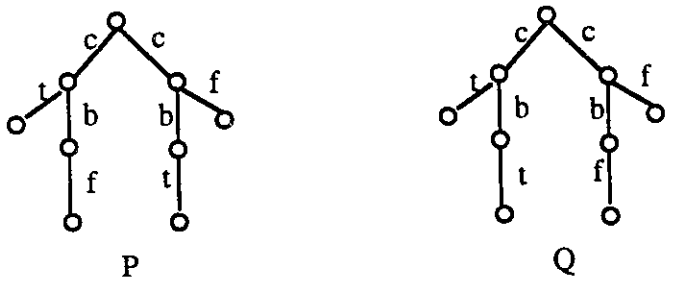
For theorem 5.3 of Chapter 5, since we proved that $=_{must} = =_{nra}$ in theorem 5.2, thus, we have $=_{NR} \neq =_{nra}$. This proves theorem 5.3.

We strongly conjecture that $=_{NR}$ implies $=_{must}$, but we are unable to prove it at the present.

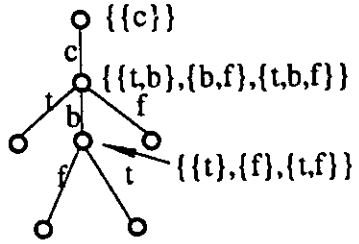
The difference between processes P and Q is also discussed in [Lan 89], where they used labels *coin* for c, *bang* for b, *coffee* for f, and *tea* for t and conceived P and Q as two vending machines. The equivalence defined there was called failure trace equivalence. Our NR-equivalence is also different from failure trace equivalence, because failure trace equivalence coincides with refusal testing equivalence.

♠ ♠ ♠ ♠

For a summary of this chapter, we evaluated the NRS semantics by comparing it with two other major semantic views of distributed systems. Then, the distinguishing power of the nondeterministic ripple equivalence is analyzed.

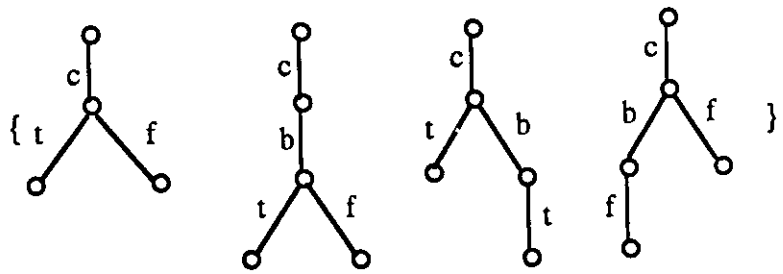


(a) Processes P and Q

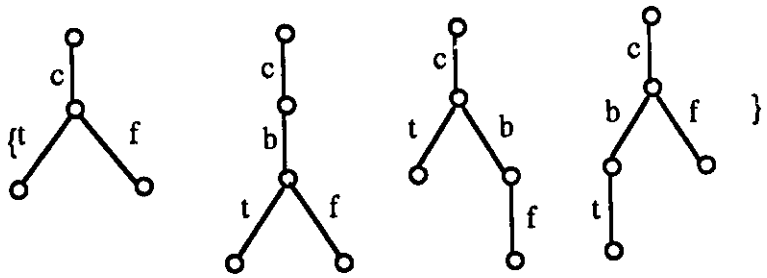


SAT(P) = SAT(Q)

(b) Acceptance trees of P and Q



NRS(P)



NRS(Q)

(c) NRS characterization of P and Q

Figure 6.3 (a) Processes P,Q and (b) their Acceptance trees and (c) NRS characterization Trees

Chapter 7

Extensions of NRS

In the previous chapters of this thesis, we have based the NRS semantic formalization on *finite* processes without invisible actions in order to focus on the basic concepts and development of this new semantic view. In this chapter, we *briefly* describe an outline about extensions of the NRS semantic formalization to recursive processes and invisible actions. Most of the proofs are not provided. Semantic developments similar to the work in this chapter appeared in [NH 83, Phi 85]. Detailed theory on recursive processes has been discussed in [Hen 88]. We refer to these works for the basic mathematical concepts and motivations.

7.1 NRS with Recursive Processes

For processes represented by labelled transition systems(LTS), an infinite process means that either the process has an infinite sequence of transitions, or, some sequence of its transitions includes cycles, that is, for some state p and trace s, $p \in D(p, s)$. The definitions 3.3 - 3.6 in chapter 3 about nondeterministic ripple sets have the ability to include these cases. Note that we do not include internal actions in a process until the next section.

Now, we discuss processes specified by recursive terms or infinite ST's. We first review the definition of recursive processes [Hen 88, NH 83].

Let X be a set of variables, ranged over by x. Let Σ_K be a set of operators of arity k, and we use Σ to represent $\cup\{\Sigma_K \mid k \geq 0\}$. The set of recursive terms over Σ, denoted REC_Σ, ranged over by t, u is defined by the following syntax:

(7.0) $t ::= x \mid op(t_1,\cdots,t_k),\ op \in \Sigma_K \mid rec\ x.t$

The operation rec $x._-$ binds occurrences of x in the subterm t of rec $x.t$. This results in the usual notions of free and bound variables in a term. Let FV(t) be the set of free variables in t. If FV(t) = 0, we say that t is *closed*. Let $t[u/x]$ denote the term which results from substituting u for every free occurrence of x in t. In general, let SUB be the set of *substitutions*, i.e., mappings from variables to terms. Let $\rho \in$ SUB, and $t\rho$ denote the result of substituting $\rho(x)$ for every free occurrence of x in t, for every x in X. A substitution is closed if for every x in X, $\rho(x)$ is closed. A closed term is *guarded* if, in a term, rec $x.t$, every free occurrence of x in t occurs within a subterm of the form $a.u$. Note that a term is *finite* if it is closed and contains no occurrence of rec $x._-$. Here, we are only interested in the closed and guarded terms/processes, and make $\Sigma = \{NIL, a., +\}$.

Similar to chapter 4, we also use processes defined by operators $t\oplus$ and $\pm$, where it is necessary. Thus, we define:

(7.0.1) $t_1 ::= t \mid t_1 \oplus t_1 \mid t_1 \pm t_1 \mid a.t_1$

where t is defined by (7.0).

The transition rules of a recursive process are defined to be the rules in (3.3.1) of section 3.3, and the following:

(7.1) rec $x.t \xrightarrow{a} q$, if $t[\text{rec } x.t/x] \xrightarrow{a} q$.

With the definition of recursive processes and their transition rules, the definitions in chapter 3 do not provide any difficulties in defining NRS for recursive processes.

From (7.1), we have: rec $x.t$ $=t$[rec $x.t$ /x]. This is called a *unwinding rule* of recursive terms. In practice, the recursive processes are often approximated by this finite unwindings. The unwindings t^n of a recursive term $t \in \mathrm{REC}_\Sigma$ may be defined as:

i) $t^0 = NIL$,

ii) $x^{n+1} = x$

iii) $(\text{rec } x.t)^{n+1} = t\,[(\text{rec } x.t)^n/x]$,

iv) $\mathrm{op}(t_1,\cdots,t_k)^{n+1} = \mathrm{op}(t_1{}^{n+1},\cdots,t_k{}^{n+1})$.

Let $\mathrm{Fin}(t) = \{t^n \mid n \geq 0\}$, and for $t^n \in \mathrm{Fin}(t)$, $\mathrm{NRS}(t^n)$ is defined as in (3.5) of chapter 3. Specifically, we have

$$\mathrm{NRS}((\text{rec } x.t)^{n+1}) = \mathrm{NRS}(t[(\text{rec } x.t)^n/x]).$$

In general, we expect that the following holds

(7.2) $\mathrm{NRS}(\text{rec } x.t) = \mathrm{NRS}(t[\text{rec } x.t/x])$.

Proof of (7.2) requires the fixpoint theory [Hen 88]. We leave it for future study.

We now show some examples of recursive processes:

-- $u = \text{rec } x.(ax + bx)$

$u^0 = NIL$, $\quad \mathrm{NRS}(u^0) = \{\{\varepsilon\}\}$

$u^1 = a + b$, $\quad \mathrm{NRS}(u^1) = \{\{a\}, \{b\}\}$

$u^2 = a(a+b) + b(a+b)$, $\quad \mathrm{NRS}(u^2) = \{\{aa\}, \{ab\}, \{ba\}, \{bb\}\}$

$= \{\{a\}.\mathrm{NRS}(u^1), \{b\}.\mathrm{NRS}(u^1)\}$

In general, we have $\mathrm{NRS}(u^{n+1}) = \{\{a\}.\mathrm{NRS}(u^n), \{b\}.\mathrm{NRS}(u^n)\}$

and expect $\quad \mathrm{NRS}(u) = \{\{a\}.\mathrm{NRS}(u), \{b\}.\mathrm{NRS}(u)\}$.

In fact, $\mathrm{NRS}(u) = \{\{s\} \mid s$ is *infinite* in $L(u)\}$, and each $nrt \in \mathrm{NRT}(u)$ consists of only one path labelled by an infinite trace of u.

-- t = rec $x.(abx + acx)$

$t^0 = NIL$, $\quad NRS(t^0) = \{\{\varepsilon\}\}$

$t^1 = ab + ac$, $\quad NRS(t^1) = \{\{ab, ac\}\}$

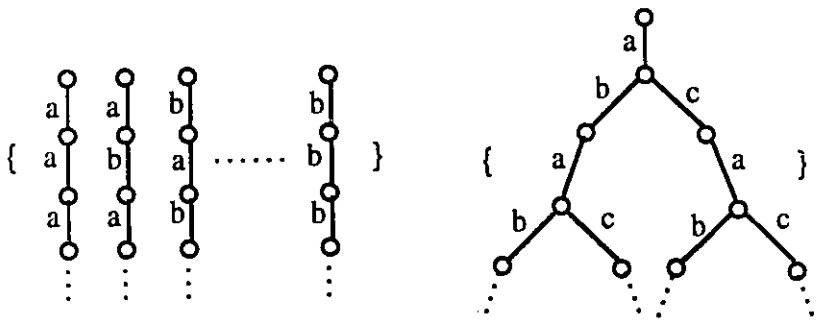
$t^2 = ab(ab + ac) + ac(ab + ac)$, $\quad NRS(t^2) = \{\{abab, abac, acab, acac\}\}$

$= \{ab.NRS(t^1) \cup ac.NRS(t^1)\}$

Similarly, we have $NRS(t^{n+1}) = \{ab.NRS(t^n) \cup ac.NRS(t^n)\}$

and expect $NRS(t) = \{ab.NRS(t) \cup ac.NRS(t)\}$

In fact, $NRS(t) = \{\{s \mid s \text{ is } \textit{infinite} \text{ in } L(t)\}\}$, and $NRT(t)$ contains only one tree which consists of all infinite traces of t.



(a) NRT(rec $x.(ax + bx)$) (b) NRT(rec $x.(abx + acx)$)

Figure 7.1 Examples of NRS for recursive processes

Note that the definition of a complete trace in the domain of recursive processes should be modified as:

$$CT(t) = \{s \mid r \text{ —}s\text{→} q \text{ and } \neg(\exists a \in Act, q \text{ —}a\text{→}) \text{ or}$$
$$r \text{ —}s\text{→ is an infinite derivation}\}.$$

That is, "*complete*" in an infinite environment means "*infinite*". Then, we show the deterministic tree forms of NRS(u) and NRS(t) in Figure 7.1 where each path in (a) and (b) is labelled by a complete trace of u and t, respectively.

Next, we discuss testing of recursive processes.

Testing recursive processes can be implemented by finite tests. For the finite tests defined in definition 5.2 of chapter 5 to be applicable here, we modify the definition as follows.

Definition 7.1 (*Definition of tests for recursive processes*)

For a $nrt \in \mathrm{NRT}(t^n)$ and all $s \in L(nrt)$,

(i) If $S(nrt\ (s)) \in CA(s)$, then,

for all $a \in (Act - (S(r, s) \cup (S(r, s) \text{ of } t^{n+1}))$,

add trace saF into the nrt.

(ii) If $S(nrt\ (s)) \notin CA(s)$, then,

for all $a \in (S(r, s) - S(nrt\ (s)))$,

and $a \in (S(r, s) \text{ of } t^{n+1} - S(nrt\ (s)))$

add trace $sa\sigma$ into the nrs,

for all $a \in (Act - (S(r, s) \cup (S(r, s) \text{ of } t^{n+1}))$,

add trace saF into the nrs.

The nrt obtained is a test.

The difference between definitions 5.2 and 7.1 is only for treating complete traces in t^n. Because in the implementation of a recursive process, when the traces in its n-th unwinding process is consumed in an execution, the traces in its n+1 unwinding process is followed, we should not count these traces as a failure in finite tests, that is, actions in $S(r, s)$ of t^{n+1} are not considered as a failure, if s is a complete trace of t^n. For finite processes, definition 7.1 goes back to definition 5.2, because $t^n = t^{n+1}$ if t is a finite process.

With this definition of tests, it is interesting to see the relation between terms in Fin(t) based on the testing relation in chapter 5. Given $t^n \in$ Fin(t), and the corresponding set of tests $T(t^n)$, we have, for each $e \in T(t^n)$, and $t^m \in$ Fin(t) and m $\geq$ n, t^m *pass* e. This is because all traces in $L(t^n)$ are finite, when a complete trace of $L(t^n)$ in e is consumed in a testing run, success σ is reported according to proposition 5.2. The complete trace must be consumed in the test run starting with $t^m \| e$ according to propositions 5.2 and 5.3 (note that t^m has a prefix process which is the same as t^n according to the unwinding definition). Then, we have the following proposition.

Proposition 7.1: Let t^n, $t^m \in$ Fin(t) and n $\leq$ m, then $t^n \geq_{nra} t^m$

Proof: Straightforward from above discussion, according to the definition of $\geq_{nra}$.

This tells us that testing of recursive processes can be implemented by finite tests, and depending on time and cost, the testing procedure can be carried to any unwinding level of the recursive process under test.

7.2 NRS with Invisible Actions

Invisible (or internal) actions of a process are actions whose executions cannot be observed and controlled by its environment. To model internal actions in a system specified by a labelled transition system, a special action symbol "τ" is used to label the transitions which cannot be controlled by the environment, i.e., *internal transitions*. We now discuss how to extend our NRS approach to handle processes with internal actions/transitions. We use Act_τ to represent $Act \cup \{\tau\}$, and a labelled transition system is now defined as $\{P, Act_\tau, r, \longrightarrow\}$.

First of all, we analyze the possible transitions at a state. Three cases of transitions can be distinguished at a state $p \in P$.

i) The transitions from p are all labelled by *visible* actions a, i.e., $a \neq \tau$. That is,

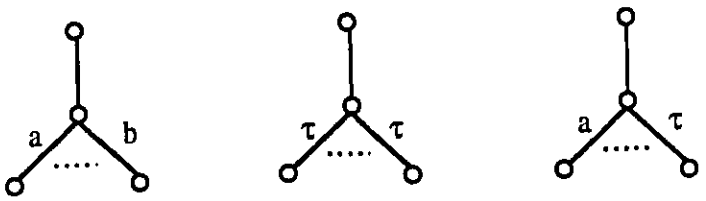
$\neg(p \xrightarrow{\tau})$

ii) The transitions from p are all labelled by the *invisible* action τ. That is,

$\neg(\exists a \neq \tau, p \xrightarrow{a})$

iii) Some transitions from p are labelled by visible actions, and some others are labelled by the invisible action τ.

These cases are shown in Figure 7.2.



Case i Case ii Case iii

Figure 7.2 Possible transitions at a state

We call the states in case i and ii *balanced* states, and the states in case iii *unbalanced states*. We also call states in case i *stable* states and states in cases ii and iii *unstable* states. Predicates *balanced*(p) and *stable*(p) are used to describe these properties of each state p, respectively. The definitions about NRS in chapter 3 can be modified to handle these cases as follows.

Case i is the case we are based on in chapter 3, and has been fully discussed. In the states of case i, both environment control and process control to a process can be clearly demonstrated as shown in Figure 3.2. Let us look at case ii. The transitions at a state in case ii are totally controlled by the process. This means that the process does not need to ask its environment for an action synchronization (not like in case i), and the next state

reached in the transition is also out of the environment control (like in case i). In this case, only a slight explanation is needed in the definitions about NRS. Because the environment has nothing to do in the state, the action selection in the state (p) should be ε, that is, (p, ε) is in a choice pattern of the process. The ripples at the state p is simply defined as before in definition 3.2, that is,

$D(p, \tau) = \{p' \mid p \text{—}\tau\text{→} p'\}$, the *Ripples* at p with respect to τ.

This is enough to handle case ii. It appears that for balanced states, the definitions are not changed.

Handling case iii is not as easy as cases i, and ii, because the controls to a process at a state in case iii cannot be clearly distinguished between the process and its environment. At case iii, the state p has transitions which are labelled by visible or invisible actions. The transitions labelled by invisible actions may happen anytime without the environment knowledge. Two situations can be distinguished here for analysis.

--*Situation 1*. The environment is ready to perform some action $a \neq \tau$ at p, and the interaction of a between the process and the environment is executed before any of the internal transitions happens. After this, the process may be in anyone of the states in $D(p, a)$. But, under the circumstances, it is quite possible that any of the internal transitions at p may happen before the interaction of action a. In this condition, the environment cannot do anything, and the process may be in any state in $D(p, \tau)$ after the internal transition. Furthermore, even if the interaction of a occurs, it might have been performed in a transition starting at a state in $D(p, \tau)$, and this also means that an internal transition happen before interaction of a. Thus, in this situation, the process behavior at p is performing either a or τ, and then reaching a possible state in $D(p, a) \cup D(p, \tau)$.

--*Situation 2*. The environment chooses to do nothing or is not ready to do something at state p and wait until any of the internal transitions happens. Under this circumstance, it means that the environment chooses the action τ, and then the process reaches some state in $D(p, \tau)$.

Thus, in case iii, at an unbalanced state p, there are two kinds of action selections in a choice pattern. One is selecting some $a \in D(p, a)$, and the invisible action τ (situation 1). The other is selecting the invisible action τ only (situation 2).

According to the reasoning for cases i - iii, we modify definition 3.4 for a choice pattern to accommodate internal actions as follows. For symbol consistency between τ and ε, we define that, for $s_1, s_2 \in Act^*$

$$s_1\tau^n s_2 = s_1 s_2 \text{ and } \tau^n = \varepsilon \text{ where } n \geq 0.$$

Definition 7.2 (*definition of a choice pattern*)

1) $(r, a) \in C$, r is the initial (root) state of process P

2) If $(p, a) \in C$, then for all $q \in D(p, a)$ where $S(q) \neq \emptyset$,
   if *balanced*(q) and *stable*(q), then for exactly one $a \in S(q)$, $(q, a) \in C$,
   if *balanced*(q) and $\neg$*stable*(q), then $(q, \varepsilon) \in C$,
   if $\neg$*balanced*(q), then
   (for only one $a \in S(q)$ and $a \neq \tau$, $(q, a) \in C$; and $(q, \tau) \in C$)
   or (for all $a \in S(q)$ and $a \neq \tau$, $(p, a) \notin C$; and $(p, \tau) \in C$).

Condition 2) is modified according to the above discussion. It is interesting to note that an unbalanced state has both features in a stable state and an unstable state. Intuitively, an unbalanced state in case iii is the combination of states in case i and case ii. Definition 7.2 is compatible with definition 3.4, and will degenerate to definition 3.4 if the internal action is not considered.

From the above reasoning, we have the following equations for a process p.

if $balanced(p)$ and $\neg stable(p)$, $\sum_{q_i \in D(p,\tau)} \tau q_i =_{NR} \tau \oplus_{q_i \in D(p,\tau)} q_i$,

if $\neg\, balanced(p)$, $\sum_{q_i \in D(p,\tau)} \tau q_i + \sum_{a_k \in S(p),\, a_k \neq \tau} a_k t_k$

$$=_{NR} (\oplus_{q_i \in D(p,\tau)} q_i) \oplus (\sum_{a_k \in S(p),\, a_k \neq \tau} a_k t_k) \pm (\oplus_{q_i \in D(p,\tau)} q_i)$$

These equations hold since they have the same NRS in both sides due to the NRS definitions for processes with τ actions and the definitions of $\oplus$ and $\pm$ operators. The last two equations can be simplified as

$$\tau t \oplus \tau u =_{NR} \tau(t \oplus u), \text{ and } \tau t + \tau u =_{NR} \tau t \oplus \tau u,$$

and $$\tau t + u =_{NR} (t \oplus u) \pm t$$

The above equations are also true for testing equivalence and failure equivalence [NH 83, Hoa 85]. This shows that our understanding of internal actions is consistent with theirs.

However, in general, if $t =_{NR} u$, $t + z =_{NR} u + z$ does not hold. This can be illustrated by an example. $a =_{NR} a$, $\tau NIL =_{NR} NIL$, but $a + \tau NIL \neq_{NR} a + NIL$, because

$$a + \tau NIL =_{NR} a \oplus \tau NIL \pm NIL \text{ and } a + NIL =_{NR} a$$

We expect that for processes t, u and z, if they are stable and $t =_{NR} u$, $t + z =_{NR} u + z$ holds. To make it hold in general, the form of NRS needs to be slightly modified. We leave it for our future work.

For testing processes with τ actions, definition 7.1 should be modified. This mainly concerns extension of the definition for the transition relation in a labelled transition system. The relation $p \xrightarrow{a} q$ needs to be extended as

$$p = a \Rightarrow q \text{ if } p \xrightarrow{\varepsilon} p' \xrightarrow{a} q' \xrightarrow{\varepsilon} q$$

that is $p = a \Rightarrow q$, if p can evolve into q by performing an a action interspersed with an arbitrary, possibly zero, number of internal actions. Furthermore, for action sequences in Act^*, we define:

$$p = \varepsilon \Rightarrow q \quad \text{if} \quad p \xrightarrow{\tau^n} q$$
$$p = as \Rightarrow q \quad \text{if} \quad p = a \Rightarrow p' \text{ and } p' = s \Rightarrow q \text{ for some term } p'.$$

that is, $p = s \Rightarrow q$ means that p can be transformed into q by performing the sequence of actions s, possibly interspersed with internal actions.

With the extended transition relation, the definitions/notations in definition 5.1 can be redefined by using $= a \Rightarrow$ in place of $p \xrightarrow{a} q$. Accordingly, definition 7.1 should be modified by using new meanings of the notations in definition 5.1. This only concerns some notation transformations and thus, is omitted here.

♠ ♠ ♠ ♠

At the end of this chapter, we discuss an attempt to extend NRS to the full language of pure CCS [Mil 80, Mil 88].

The terms in CCS can be considered as being generated by the following BNF notation:

(7.3) $t ::= NIL \mid a.t \mid t + t \mid t|t \mid t \backslash a \mid t[a/b]$

and each term can be represented as a synchronization tree [Bro 83]. We have already dealt with the first three forms. Now, we discuss the others.

$t|u$ is called the composition of t and u. For $t = \sum_{i=1,n} \lambda_i t_i$, $u = \sum_{j=1,m} \mu_j u_j$, where $\lambda_i, \mu_j \in Act_\tau$, the composition is defined by

$$t|u = \sum_{i=1,n} \lambda_i (t_i | u) + \sum_{j=1,m} \mu_j (t | u_j) + \sum_{\lambda_i = \mu_j} \tau (t_i | u_j)$$

where the actions λ_i and $\underline{\mu}_j$ are *complementary* actions (note that a up-bar is used for complementary actions in literature, but, in my word processor, I cannot place a bar above a letter). Without losing any generality, we assume that only visible actions have complements., and $a = \underline{a}$ for all $a \in Act$.

The other two forms of CCS processes can be interpreted by synchronization trees as, using the same notation for t as above:

$$t \backslash a = \sum_{\lambda_i \neq a} \lambda_i (t_i \backslash a),$$

$$t[a\backslash b] = \sum_{i=1,n} \lambda_i [a\backslash b] t_i [a\backslash b],$$

where for an action λ, $\lambda[a\backslash b]$ is a if $\lambda = b$ and λ otherwise. These two operations $\backslash b$ and $[a\backslash b]$ are called *restriction* and *relabelling* by Milner. Restricting prunes away branches involving the particular action b, while relabelling replaces all occurrences of one action by another.

These processes are defined recursively and enable a term involving composition, relabelling or restriction to be manipulated into a ST form. Accordingly, our NRS definition can be applied to them. The processes defined by (7.3) are finite. For recursive processes involving these operators, we can set

$$\Sigma = \{NIL, a., +, |, \backslash a, [a\backslash b]\}$$

in (7.0), and similar discussion in section 7.1 applies.

In conclusion, this chapter provided an outline on extension of NRS approach to recursive processes, processes with internal actions, and other process constructors in CCS. Related issues such as algebraic characterization and testing for these processes were also discussed. We have omitted most of the proofs in this chapter, but, most of the extensions are straightforward.

Chapter 8

Considerations of Practical Applications of NRS Approach

In this chapter, we investigate the aspects about practical applications of the NRS semantic approach. While in previous chapters, we have concentrated on the formalization of mathematical properties of this approach.

As mentioned in earlier chapters, the NRS approach was motivated from our experience in practical testing. Thus, we expect that applying this theory back to activities in practical testing should be promising. Theoretical formalization of testing with the NRS has been studied in chapter 5. Discussion of the aspects of NRS regarding practical testing will be the main topic of this chapter.

This chapter is organized as follows. In section 8.1, we present a general discussion on the application of NRS semantics to various aspects of system development. Then, in section 8.2, we describe the special contributions of the NRS semantics to conformance testing of telecommunication systems/distributed systems. Finally, in section 8.3, we provide a case study of applying this theory to the practically-oriented protocol INRES defined by ISO and CCITT [FMCT] for purpose of comparing the suitability of formal methods and models in conformance testing.

8.1 Aspects of Applications of NRS Approach

The behavioral view of the NRS approach appears natural and practical in understanding and capturing system operations. This renders the semantic formalization of the view

ability to be practically used in various aspects of system development. Some application areas of the NRS approach are discussed in this section. They include: interpretation and representation of system behaviors with NRS, system verification with NRS, testing with NRS, and other aspects of applications.

--Interpretation and representation of system behaviors with NRS

The formalization method of the NRS semantic view constitutes a *basic framework* in interpreting and representing system behaviors. This includes the labelled transition systems(LTS's)/synchronization trees(ST's), the NRS representation, and the NRS derivation.

Assume that a system developer(an analyst, designer, or tester) is given a basic system specification to base his work on for a project. In this work, the analyst needs to know how the system described by the specification is supposed to work, the designer needs to transform this basic specification into a series of design refinements, and the tester needs to derive a set of tests (test suite) from the specification for conformance testing. In each case, the first thing for the developer to do is to understand and analyze the behaviors of the system specified. In general, one may hope three things in doing this:

(a) the modeling of system behaviors should naturally correspond to the system's operation so that the method of behavioral understanding can be practically used.

(b) there is a simple and intuitive behavior representation for describing and reasoning system behaviors.

(c) the framework of this behavioral analysis can be automated.

It seems that these requirements are satisfied by the basic framework. Firstly, the behavioral view in the framework is natural and practical for the system behavioral understanding. This view takes a system behavior as the result of mutual influences between a process and its environment, and thus, models both contributing factors in the system operation. Based on this view, a system behavior can be projected onto the behaviors of both a process and its environment sides. Their mutual influences result in the semantic object, nondeterministic ripple set. The two representations of NRS, trace set notation and deterministic tree notation, are actually representations of the process and environment behaviors, respectively. Understanding system behaviors is actually observing process behaviors from the environment side. That is, this behavioral view naturally models system behaviors for a system developer. This is illustrated by the testing theory developed in chapter 5, and explains why the deterministic tree notation of NRS can be directly used for test definition/generation. Test cases are the behavioral representation of the environment side in a system. This is one example of practical application of the NRS approach.

Secondly, NRS approach has a simple and intuitive behavior representation for describing and reasoning system behaviors. This is reflected in the two ends of the approach. At one end, this approach takes LTS's/ST's as basic description methods of system specifications. LTS's/ST's have been extensively used in theoretical studies and practical applications in the area of distributed systems. They are models or theoretical foundations for the specification languages such as **SDL** [ST 87], **Estelle** [BD 87], **Lotos** [EVD 89] and the standard test specification language **TTCN** [MoPr 92], and informally, they have been used as graphical representations of system behaviors in many applications. This means that the NRS approach can be applied to various occasions according to the needs. On the other end, the semantic objects, NRS, have been represented to be applicable in different situations such as verification, design, and trace analysis. These representations are 1)

algebraic notation(for verification), 2) trace set notation(for trace analysis), and 3) deterministic tree notation (for system design and test design). The intuitivity and simplicity of the NRS representations have been discussed in section 6.1.

As a summary, this basic framework acts like an interface between the NRS approach and its users(system developers). It takes a system specification and decomposes it into a set of behavioral representations(NRS) which can be used for system analysis and design in the subsequent phases of a system development process.

--System verification with NRS

Verification concerns the validation of process properties by a calculus with the algebraic(or symbolic) representation of processes. The work on algebraic characterization of NRS discussed in chapter 4 has established such a calculus for verifying process properties based on NRS semantics. The examples of the properties which can be verified include the NR-equivalence and testing equivalence between processes, liveness of processes and other similar properties formalized in other semantic theories. A realistic example of process calculus for test generation is given in section 8.3. As a summary, this work about NRS calculus provides a solid mathematical foundation for the studies of processes based on the NRS theory, even though many aspects of this theory have not been fully developed compared with other well-known theories.

--Testing with NRS

It has been mentioned that the motivation of this semantics came from our experience in observation and analysis of communicating system behaviors in conformance testing. As a result, the NRS semantics is well suited for application to the conformance testing of

communicating systems. Several subparts of this semantics actually support a testing theory for test generation, test selection, and test specification.

The basic framework involved in understanding system behaviors discussed above is also a process for test generation, namely, given a specification, map it into its semantic object NRS. This NRS can be considered as a comprehensive test suite for testing all valid functions in implementations derived from the specification. Many different test suites relating to the different set of test purposes and testing configurations can be defined and selected from it. Test specification can be easily done with this semantics. Each deterministic tree representation of a NRS can be directly mapped into the international standard test specification language TTCN [MoPr 92] as a dynamic behavior description tree. Examples of such TTCN specifications are provided in the following section of this chapter.

The discussion about testing with NRS in chapter 5 provided some practical and theoretical results toward a complete testing theory. These include precisely defining a testing configuration, a procedure for test generation, and a conformance relation. The test suite defined from the NRS of a specification exactly tests the conformance relation $\leq_{nra}$ between the specification and its implementations. Test suites for other conformance requirements can also be defined from a NRS in a manner similar to the approach in chapter 5.

We have discussed the application of this semantics to testing in general above. More specifically, the main contributions of this semantics to testing theory and practice are the following:

(1) It contributes to solving a practical testing issue concerning relations between test purposes and the corresponding test cases [BP 89].

(2) It contributes to solving a theoretical testing issue discussed in chapter 5 about generating and using tests only from a specification when testing for a conformance relation between this specification and its implementations.

(3) It illustrates a direct application of a semantic theory to testing theory and practice.

(4) The NRS semantic theory can unify important formal and informal testing theories for both deterministic and nondeterministic specifications within the same framework. These, for example, include D-method(FSM-based) [SL 86], Test equivalence(LTS-based) [NH 83], and Canonical testers [Bri 88].

The general discussion above and points (1)--(3) will be further illustrated and explained in the next two sections. However, point (4) is not pursued further because it needs too much context introduction and may confuse our main focus here. Some ideas on this point are given in the discussion of section 8.3 of this chapter.

--Other aspects of applications

The applications discussed above are based on the development of the NRS approach in this thesis. However, the work here only includes a very basic part. Many other aspects of this approach need to be further studied. Some other applications of this theory include description of system design such as the process transformation relation between refinements of a system, trace analysis, feature interaction, test result analysis, and so on. Due to the different natures of these topics, we do not provide any further discussion here.

8.2 Contributions of NRS approach to Conformance Testing

In this section, we briefly discuss the current state of the field of conformance testing, and point out a gap between theoretical studies and practical applications in this field. Then, we discuss the special contributions of NRS semantics to bridging this gap. Finally, an example follows these discussions to show the application of NRS to conformance testing.

-- A gap between theory and practice of conformance testing

In recent years, theory and practice on conformance testing have been actively carried out in both academic and industrial worlds. In theoretical aspect, the representative testing models: bisimulation testing [Abr 87, Mil 80], failure/testing equivalence [BHR 84, NH 83, Hen 88] have been developed. Many other theories also exist but they mainly follow the formats of these models. The main advantage of these models is that they are precise, mathematically-based theories and possess well-defined theorems for discussing process properties. But, in considering applications of these theories to practical testing, some unfavorable features are common in them.

(1) Tests in these models are terms of some algebra. The set of tests are identical for every process without discrimination, and thus, even for trivial processes, the tests considered can be infinite. This seems impractical.

(2) There are no relations between tests and test purposes. This point is extremely important for practical testing.

Thus, the present formal testing models are still not in the stage of practical applications.

In the practical world of conformance testing, on the other hand, the situation is different. Here, practical applicability of each testing theory or technique is the main requirement. At the state of art, the aspects (1) and (2) listed above are practised in realistic testing as follows:

(i) Tests are derived from the specific process(specification) whose implementation is under test.

(ii) Each test is derived from a test purpose.

In general, the practical testing techniques are ad hoc and informal. The work for test generation and selection are mainly manually done , which is tedious and error-prone. Furthermore, most practical techniques deal with only deterministic processes.

At the current stage, many systems are specified in natural languages or informal methods. But, applying formal specification and testing techniques has been gradually recognized in the industry world. For large systems specified in formal specification techniques such as Estelle, SDL and Lotos, current testing techniques are not adequate for test generation and verification of testing completeness. To solve this problem, applying formal methods to testing is necessary.

--NRS contributions to bridging the gap in theory and practice of testing

The above discussion points out a gap between theoretical studies and practical applications in conformance testing, that is, the existing testing theory is formal but less practical, and practical testing techniques are practical but informal. Two *crucial* points which are handled differently in the two sides are listed. For a formal testing theory to be practically applicable, its processing of these two points has to be matched with realistic testing

techniques, that is, the gap between theory and practice of conformance testing has to be bridged.

To make formal theories practical, our NRS approach has made some contributions to bridging the gap as indicated in the following. we use the indicator like (1-i) to indicate how the theoretical and practical differences of conformance testing are harmonized in the NRS semantics.

(1-i) NRS semantics contributes to solving a theoretical issue in testing. That is, the testing framework derived from NRS approach generates and selects tests from a specification if the implementation described by the specification is under test. Furthermore, to attach practical meaning to each test, complete traces are used as the basic elements in test design, and to deal with the possible nondeterministic rippling, coordination between a test and process is considered. These considerations reduce the number of tests to the comparable level required in realistic situation for test selection. This is illustrated in Figure 8.1.

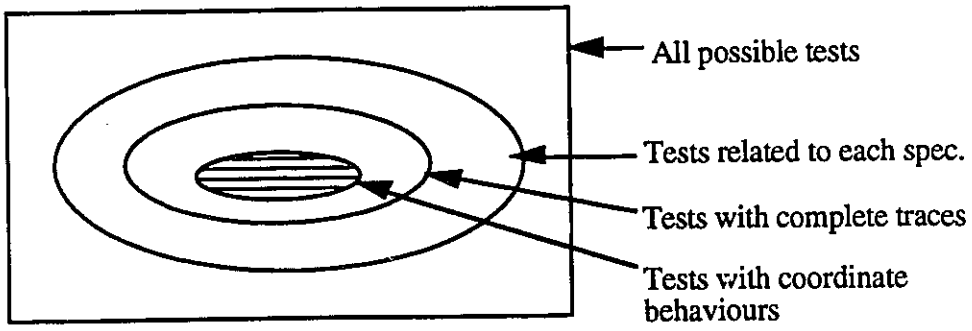


Figure 8.1 Domain of test cases

(2-ii) NRS semantics contributes to solving the problem that a test in a testing theory should be related to a test purpose. In NRS semantics, a nondeterministic ripple set is derived from a choice pattern, and the choice pattern is decided by a certain function to be

tested. This attaches to each test a test purpose, that is, *test to check if the related function conforms to its specification.*

In addition, NRS approach shows a direct application of a semantic theory to testing practice. The tests derived in NRS semantics can be directly mapped to TTCN [MoPr 92]. since each deterministic tree representation of the elements in NRS set can actually be seen as a dynamic behavioral tree in TTCN. This is illustrated by the example in section 8.3.

Now, we present an example for applying the testing preorder defined in chpater 5.

We now use a change-giving machine P in Figure 8.2(a) to show the points discussed above. The change-giving machine always gives the right changes in one of the three combinations of 1p (penny) and 2p after it accepts a 5p coin. A user can select to obtain a 1p or 2p in his changes, but if he selects to have a 2p, it is up to the machine to select the combinations of the remaining changes according to availability of coins inside it. Note that P has recursive behaviors. After a change-giving service is provided, P comes back to its initial state and is ready for a next service. The black state in Figure 8.2(a) means that the behaviors of these states are the same. P can be specified recursively as

$$P = \text{rec } x.5p.(1p.1p.1p.1p.1p.x + 2p.(\tau.1p.1p.2p.x + \tau.2p.2p.1p.x))$$

Since P is a recursive process with internal actions, the definitions in chapter 7 are used. However, we only test the implementation up to its first level. Thus, the discussion below is also straightforward with the definitions of finite processes.

Now, we generate/define tests to test the implementations of P. The $NRT(P^1)$ of process P is computed as in Figure 8.2(b) which consists of two elements. By the algebraic method, $NRT(P^1)$ can be calculated as:

$$P^1 = 5p.(1p.1p.1p.1p.1p.NIL + 2p.(\tau.1p.1p.2p.NIL + \tau.2p.2p.1p.NIL))$$

$$= 5p.(1p.1p.1p.1p.1p.NIL \pm 2p.(\tau.1p.1p.2p.NIL \oplus \tau.2p.2p.1p.NIL))$$

$$= 5p.1p.1p.1p.1p.1p.NIL \pm (5p.2p.1p.1p.2p.NIL \oplus 5p.2p.2p.2p.1p.NIL)$$

$$= 5p.1p.1p.1p.1p.1p.NIL \pm 5p.2p.(1p.1p.2p.NIL \oplus 2p.2p.2p.1p.NIL)$$

Thus, NRT(P^1) = {$5p.1p.1p.1p.1p.1p.NIL$, $5p.2p.(1p.1p.2p.NIL \oplus 2p.2p.2p.1p.NIL)$}

The set of tests $T(P^1)$ can be generated according to definition 7.1 with extensions discussed in section 7.2, and is shown in Figure 8.2(c). $T(P^1)$ contains two tests e_1 and e_2 each of which is derived from nrt_1 and nrt_2 of NRT(P). The test purpose with each test in $T(P)$ is

e_1: test if the function of distributing all 1p's is correct.

e_1: test if the function of distributing all 1p's is correct.

2p with a total value of 3 pennies is correct.

Suppose that there are four implementations I_1: - I_4 of P as shown in Figure 8.2(d). By using the testing configuration and conformance relation $\geq_{nrs}$ in chapter 5, we have

$P \geq_{nra} I_1$, $P \geq_{nra} I_2$, but $P \geq_{nra} I_3$ and $P \geq_{nra} I_4$ do not hold.

This means that I_1 and I_2 are conforming implementations of the change-giving machine P, but I_1 and I_2 are wrong implementations because they do not provide the choice for a user to select a 1p or 2p for his change.

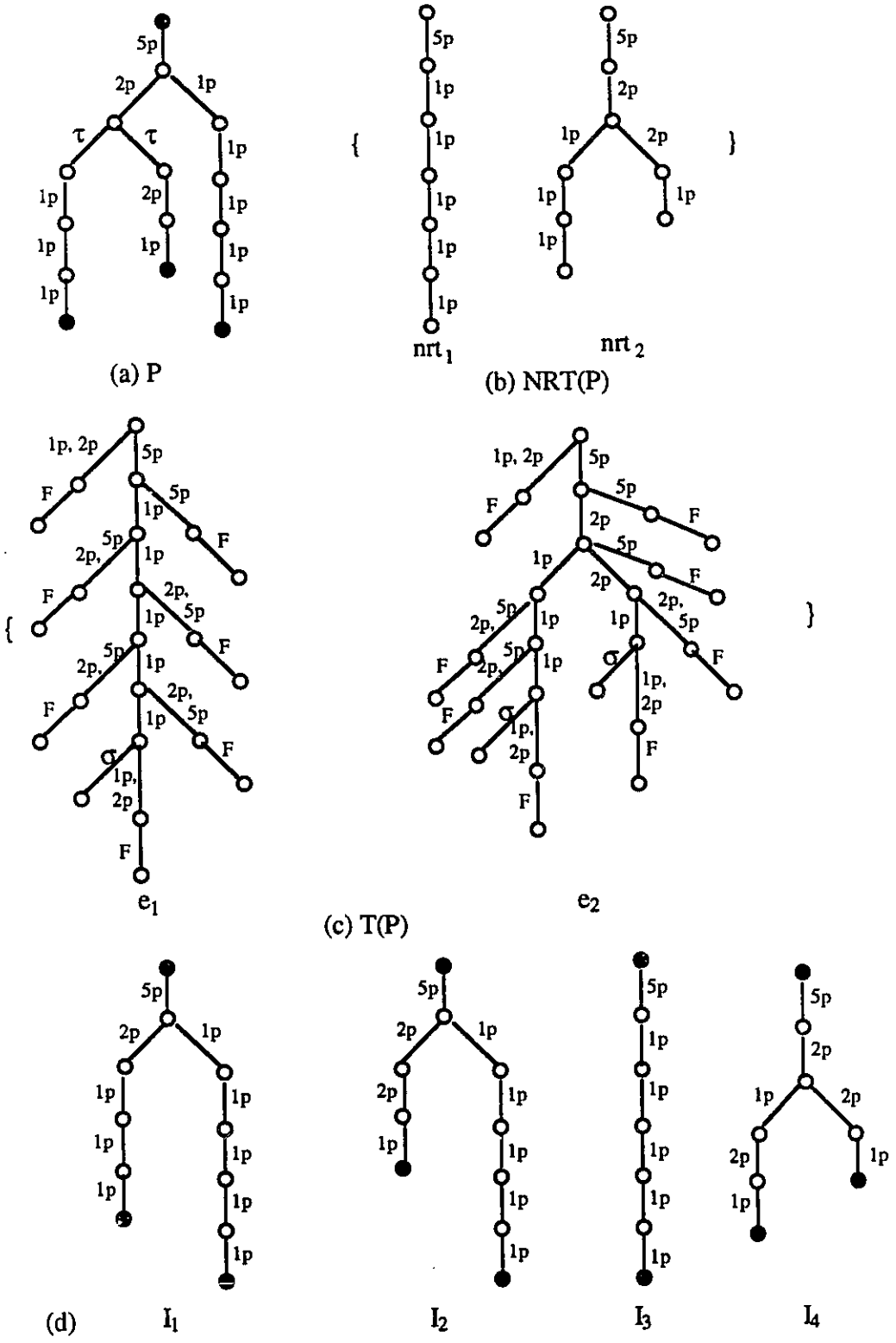


Figure 8.2 An change-giving machine *P*, its NRT(P), tests T(*P*), and implementations I's

8.3 A Case Study: Test Generation for INRES Service Provider

In this section, we provide a case study to illustrate the practicability of the NRS approach. we apply it to a toy but practical-oriented data transfer service protocol: INRES for test generation. INRES is a simplified version of the Abracadabra service introduced in [Hog 92] and has been selected by the international standards organizations ISO/IEC JTC1/SC21/P.54 and CCITT SG X/Q.10 as a reference protocol for studying testing techniques in the context of formal methods(FMCT)[FMCT]. The detailed structure and a Lotos specification of the service are included in Appendix A. We only focus on the testing issues related to this application in this section. Firstly, we introduce some testing environment and then, turn to the discussion for test generation by applying the NRS theory to the INRES. Many basic OSI(Open Systems Interconnection) concepts are used in the discussion for which we refer the ISO document: conformance testing framework and methodology [CTFM] for the standard reference.

Some explanations might be suitable for the context of this case study. The NRS behavioral view is motivated just from the practical situations similar to the example in this section, and it is formalized based on the algebraic calculus of processes. However, when we apply this theory back to the practical situation, some realistic conditions must be considered. The labelled transition systems are a formal abstract of the system operational model. In labelled transition systems, nondeterministic transitions/behaviours of a specification are represented by transitions labelled with a same action. However, in practice, especially in communicating systems, it is a common case that the branches emitted from a node are labelled with different actions, but nondeterministic transitions may happen in this node due to different initiators of the actions. In this case, nondeterminism in the specification concerned has to be analyzed based on the system configuration. This case study shows a substantiation of a formal theory into a practical application. For

example, we treat ?a (receiving an action) or !a (sending an action) as a whole action, while relations between the branches labelled by these actions are determined by the uncertainty in communications. Now, we turn to the case study.

The INRES (INitiator-RESponder) data transfer service is a connection oriented service. It consists of a Service Provider, an Initiator and a Responder. The service can be accessed from two SAPs(Service Access Point). On the one SAP(ini), the initiator must initiate a connection request before sending data. On the other SAP(res), the responder can accept the connection or reject it. After a connection is established, the responder can receive data from the initiator until a disconnection request is sent by the responder.

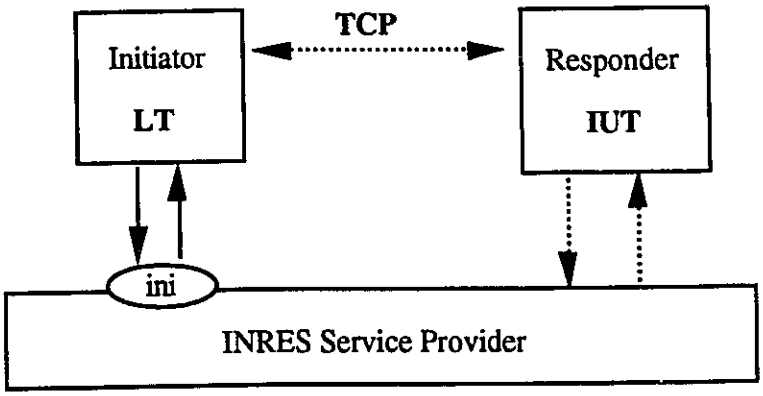


Figure 8.3. Remote testing configuration for INRES

Suppose that we use the **remote testing method** [CTFM] to test this service. Then, the testing configuration looks like the one shown in Figure 8.3. This testing method assumes that there is no test interface at the top of the IUT(Implementation Under Test) and Test Coordination Procedures(TCP) between LT(Lower tester) and UT(Upper Tester, not appear in Figure 8.1) do not exist either(if any, it is manually done). The method relies solely on the protocol to be tested for synchronization between LT and IUT. The state of the IUT is assumed to be known from actions specified for the LT. Verdicts are formulated based on stimulus provided by the LT and responses of the IUT observed by the LT. The

remote testing method is widely used for testing implementations of X.25 protocol, where the IUT is the Data Terminal Equipment(DTE) and the LT emulates the Data Communications Equipment(DCE). In our case, the IUT is the responder entity and the LT emulates the initiator.

Now, we transform the remote testing configuration and the INRES service specification into the structures which satisfy the conditions for applying our NRS approach.

According to the remote testing method, the whole system behaviors have to be controlled and observed at the SAP: **ini**. Thus, it is sufficient to generate test cases from the specified behaviors at this point. We can transform the configuration in Figure 8.3 into the similar interactive structure as shown in Figure 8.4 which is modelled by our NRS approach where LT represents the environment, and both INRES Service Provider and the IUT together(called system under test) represents the process which interacts with the LT.

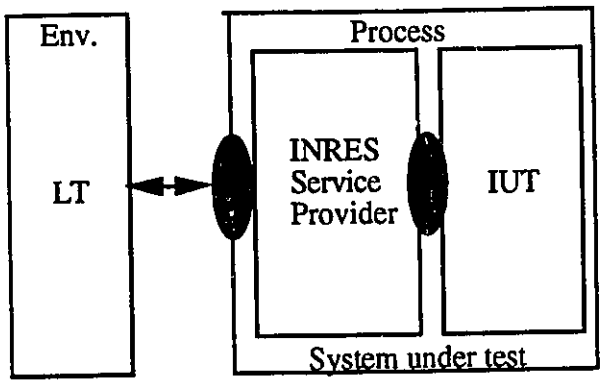


Figure 8.4 The process to be tested and its environment

Next, we need the system behavioral specification at the point: **ini**, i.e., between the process and its environment. This is obtained by projecting the whole system specification into the SAP: **ini**, and is shown as the behavioral tree in Figure 8.5, which consists of all

interactive actions controllable and observable at **ini** and time orders between them. The black and grey states represent the states in which the system has the same behaviors.

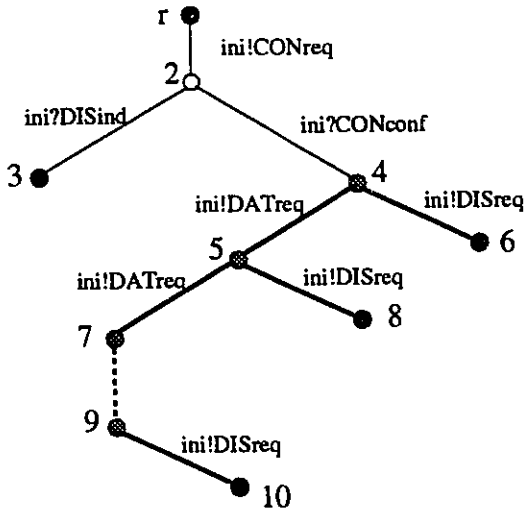
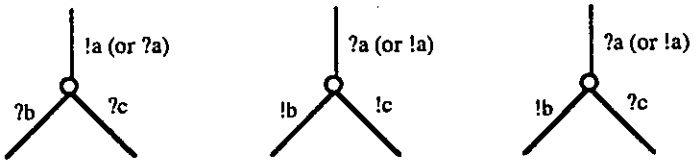


Figure 8.5. Process *ini* of showing behaviours at the SAP: ini

Furthermore and importantly, we need to determine the nondeterministic and deterministic factors in the behavioral tree of Figure 8.3 because what is described in the behavioral tree still reflect the system overall behaviors at the interaction point, and the nondeterministic and deterministic factors in the process to its environment(LT) is not specified. This usually depends on the specific configuration(i.e. testing method used) and many practical conditions which may not be described in a formal theory. Note that this is also the normal case in that when applying a formal theory back to practical situations, many practical concerns should be handled, and usually, how to deal with these concerns is informal. For example, in this case, the behavioral tree derived is not like the abstract ones where the initiators of actions are not distinguished. This gives the concepts of nondeterminism in a process concrete and practical meanings other than branches labelled with the same actions at a state. Note that for an action, we now understand that it is the whole of both its initiator and the action symbol. As such, !.a and ?.b.are examples of actions.

We decide the nondeterminism in the process according to the usual sense in the context of interactions between two entities: two branches emitting at a state are nondeterministic if the actions(prefixed with ?) labelling these branches are the potential answers to the action(prefixed with !) labelling the branch leading to this state. Similar method is used in [Pha 92]. According to this definition and the conditions in our testing configuration, three cases in a behavioral tree may be considered.



Case 1 Case 2 Case 3

In Case 1, the two branches labelled with ?b and ?c are nondeterministic because ?b and ?c are responses from the responder entity to action !a. Note that the action ?a is possible before ?b and ?c and it results in two consecutive received actions. This situation(also two consecutive sending actions prefixed with ! such as in case 2) is called *unsynchronized interaction* in protocol design which is undesirable [SB 84]. The two branches in Case 2 which are labelled by the two actions initiated by the initiator are deterministic because which one of them can be sent is under the control of the initiator(i.e., environment). Finally, Case 3 describes a situation where after receiving action a, the initiator wants to send action b, but unexpectedly, action c may arrive. In this situation, whether to accept c or send b first is under the control of the initiator, if the medium uses two channels for output and input messages, which is our assumption in this configuration. The two branches labelled by !b and ?c can be deterministic.

For the behavioral tree in Figure 8.5, by the method discussed above, we can decide that the relation between branches at state 2 labelled with ?DISind and ?CONconf are

nondeterministic, and the relation between branches at states 4, and 5 labelled with !DATreq and ?DISreq is deterministic. *Note that this behavioral tree has infinite behaviors, that is, action ?DATreq can repeatedly appear, and after action !Disind, the tree would repeat the same behavior as at root state r. In this case, as mentioned in the definition of the NRS, some proper exit point should be made in test generation.* Then, the nondeterministic rippling at each state is,

(8.3.1) D(r, !CONreq) = {2}, D(2, ?CONconf) = {3, 4}, D(2, ?DISind) = {3,4 },
D(4, !DATreq) = {5}, D(4, ?DISreq) = {6}, D(5, !DATreq) = {7},
D(5, ?DISreq) = {8}, ... D(9,!DATreq) = {10}.

Up to now, we are in a position to generate the tests for the INRES service under remote testing configuration. We first determine test purposes. As usual, test purposes for such a protocol include:

1) check for connection establishment.
2) check for data transfer of one data unit.
3) check for connection release

Next, choice patterns determined by these test purposes are, (*note that exit point is made for just complete each test purpose, i.e., one recursive level only*)

c1 = {(r, !CONreq), (2, ?CONconf), (4, ?DISreq)},
c2 = {(r, !CONreq), (2, ?CONconf), (4, !DATreq), }
c3 = {(r, !CONreq), (2, ?CONconf), (4, !DATreq), (5, ?DISreq)}

Then, we generate the finite elements in NRT(ini) based on the choice patterns c1 - c3 which are shown in Figure 8.6. *Note that nondeterministic rippling effect of process ini is based on ones in (8.3.1).*

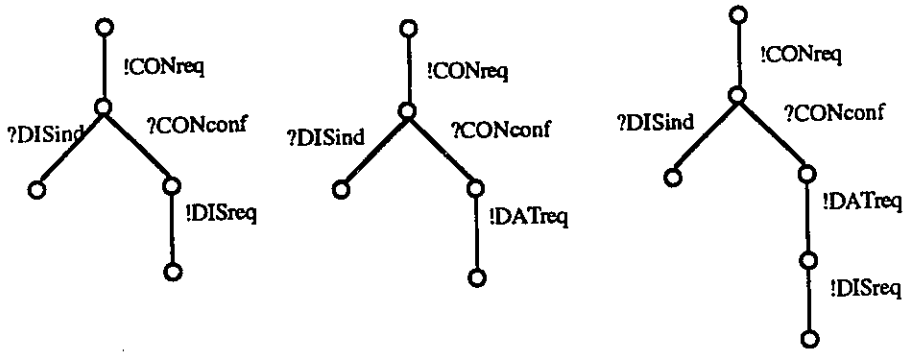


Figure 8.6 Three finite elements in NRT(ini) determined by choice patterns c1- c3

Finally, in TTCN-like notation, the three test cases are described below with verdicts added:

Test case N0.: 01 Test purpose: check for connection establishment.	Verdict
Test behavioral tree !CONreq. ?DISind ?CONconf !DISreq	*Pass*

Here, for assigning a verdict to a test, we only consider the path which completes the function of the test purpose as *Pass* (success), if it is executed. For other paths, any verdicts such as *Pass, Fail* and *Inconclusive* may be assigned depending on the practical considerations.

Test case N0.: 02 Test purpose: check for data transfer of one data unit.	Verdict
Test behavioral tree !CONreq. ?DISind ?CONconf !DATreq	*Pass*

Test case N0.: 03 Test purpose: check for connection release	Verdict
Test behavioral tree !CONreq. ?DISind ?CONconf !DATreq !DISreq	*Pass*

These three tests are the main representatives in the test suite for INRES service. Other tests actually consist of one of these three tests with some repetitions of its subpart such as sending more data units or disconnection after sending more data units.

In the above discussion, we have outlined a method of applying our test-purpose directed semantics to test generation in conformance testing for a formal Lotos specification of INRES service protocol. We may remark that the method is also well applicable to FSM-based formal specification techniques such as Estelle and SDL, and the D-method (As in [SL 86, SL 88]) could be placed under the same framework of this test generation.methodology. D-method is a commonly-used test generation method for protocols specified by FSM-based specification techniques. This can be argued by using

the same example above, if we ignore the Lotos specification and directly start with the behavioral tree of the service shown in Figure A.1 in Appendix A. By combining the states of the same behaviors together in Figure A.1 and Figure 8.3, Two FSMs are formed and can be understood as the FSM-based specification of the INRES service(similar Estelle and SDL specifications of INRES service are found in [Hog 92]). Then, the same procedure as described above can be followed for test generation. If the *distinguishing sequence* for each state is available, it can be used to verify the state following the transition to be tested as practised in D-method [SL 86, SB 82]. Then, the test for test purpose 1 will look like as follows:

Test case N0.: 01 Test purpose: check for connection establishment.	Verdict
Preamble !CONreq. Test-body ?DISind ?CONconf Post-amble ?DISind +Check(state r)	*Pass*

The test above has the same format as a real test such as the one in the test suite of X.25 protocol [MoPr 92].

From this case study, we may see that, given a specification, the steps of applying NRS semantics to test generation could be outlined as the following NRS methodology for test generation and specification:

NRS Methodology for Test Generation and Specification
1. Construct the behavioural/scenario tree of the specification
2. Derive the behavioural tree at the point of control and observation (PCO) in the testing configuration by projecting the tree to the PCO.
3. Determine the nondeterministic relations among branches at each node
4. Compute the nondeterministic ripplling effect for each action at each node
5. Determine test purposes from the specification
6. Compute choice pattern based on each test purpose
7. Generate nondeterministic ripple sets for the behavioural tree at the PCO
8. Design and specify tests from each nondeterministic ripple sets by using their deterministic tree representation

In summary, this section outlined a methodology for test generation in conformance testing by applying our NRS semantics to the INRES service. The results obtained are close to the ones obtained by informal and manual methods, and show much promise in using the formal semantics in a disciplined way for test generation and test specification in practical situations. However, our main purpose here is not to develop the methodology, but try to convey some ideas about the potential practical applicability of this semantics. As a consequence, the discussion above is only illustrative, and may be incomplete in some aspects. However, this is the basic which allows development of an NRS test generation tool in future work.

♠ ♠ ♠ ♠

This chapter has focused on the consideration on how to apply the NRS approach to practical situations. A general discussion was given in section 8.1, while in other two sections, special attentions were paid to applications in testing. Two application examples were provided, one for a change-giving machine, and the other for a practically-oriented protocol called INRES. These two examples showed different ways of applying the NRS approach and NRS advantages for conformance testing. As a result, it appears that the NRS approach is promising in studies and practical applications of distributed systems. However, discussions here are very limited, and most of them should be further extended.

Chapter 9

Conclusion

In this chapter, we conclude the thesis and discuss some directions for future research.

The idea of this work came from our experience of observation and analysis of process behaviors in test generation and test specification for conformance testing. By investigating and relating the inherent factors inside a system(process and its environment), the original idea was developed into this new semantic approach which might be called *NRS approach*. The *intention* of developing this approach was to improve the applicability of formal semantics to practical development of distributed systems. The *basic principle* of this approach is that: *both a process and its environment in a system have control over the system, and a system behavior is the result of the mutual influences of the process control and its environment control.* I think that this approach might closely capture the practical interactions of a system and hope that it provides some contributions to our intention and the formal semantic studies of distributed systems.

The basic principle was substantiated into a formal semantic interpretation of processes based on Labelled Transition systems. This was done by defining two concepts and their related formal objects: *ripples* and *choice pattern* which model the process control and the environment control , respectively. Then, by considering the mutual influences of these two objects, the semantic object, *nondeterministic ripple set(NRS)* was defined.

As a semantic object, NRS can be used to describe various properties of processes. In this thesis, based on NRS, we studied a new process equivalence, Nondeterministic Ripple equivalence, and its axiomatization. An axiom system of this equivalence was provided,

and its soundness and completeness were proved. Then, we studied an application of the NRS approach. A testing theory was formalized based on NRS. Its main differences from other formal testing theories are 1) tests are defined from the process under test, 2) each test can be related to a test purpose. The practical application of NRS approach to conformance testing was considered, and the results appeared promising. Comparison of NRS approach with other major theories such as bisimulation and failure equivalences was discussed, and the relationships between their distinguishing powers were proved. Extension of the basic NRS formalization to recursive processes with internal actions was also briefly outlined. With algebraic representation of processes, the NRS approach has been applied to a subset of pure CCS.

We now provide some remarks on the methods we adopted in the thesis.

1) The NRS semantic approach is different from other approaches in that the basic principle of the NRS approach considers the environment influence on the whole system operation. The NRS formalization of this thesis is only one specific instance of the basic principle. We expect that there are many other ways to formalize process control and environment control to obtain different semantic denotations based on this principle.

2) In the nondeterministic ripple equivalence, the equation $x + x = x$ does not hold. On the one hand, we think that this is only a matter of how to define the controls in the NRS approach. On the other hand, this aspect of the nondeterministic ripple equivalence does describe the practice in design for fault-tolerance, and distinguishes between redundant and non-redundant designs. The point here is when the equation should hold and when it should not. Furthermore, when the equation should hold, how far should the redundant factor be distinguished in a semantic definition . This seems to be a difficult topic at the moment, since some people may think that the equation should hold anyway.

3) In chapter 4, we introduced two operators $\oplus$ and $\pm$ to help with axiomatization of the nondeterministic ripple equivalence. They describe the exact duality between a process and its environment. Operator $\oplus$ is quite similar to $\oplus$ defined by Hennessy [Hen 88]. With this operator, some relation similar to the one in [NH 87] has been derived for CCS with internal actions and CCS without internal actions. These two operators were discussed to only a very limited extent in our work. We think that the application of these two operators, in particular, the operator $\pm$, is worth further investigating.

4) In the testing theory of chapter 5, we have proved that the nondeterministic ripple acceptance testing equivalence coincides with failure equivalence. We also proved that the nondeterministic ripple equivalence is different from the nondeterministic ripple acceptance testing equivalence. We strongly suspect that the nondeterministic ripple equivalence implies the nondeterministic ripple acceptance testing equivalence, but it was not able to be proved in this thesis. We are currently looking for a proof for this point.

Finally, we present some directions for future studies of the NRS approach.

1) Compared with well-known theories like bisimulation and failure/testing semantics, the work presented in this thesis is only a starting point for NRS approach. More sophisticated mathematical properties of this approach should be investigated.

2) Practical applicability of this theory to realistic issues in systems development is a major area for further study. Such research would shorten the gap between theory and practice in specification and validation of distributed systems. We expect that the potential contribution of the NRS theory to this area is promising. For example, by refined considerations and observations of interactions between a process and its environment,

smaller and more practical NRS sets can be obtained for equivalence definition, test generation and test specification.

3) The NRS behavioral view suggests trying a set of interactive alternatives in an interaction between a process and its environment. This actually assumes that the process behaviors after a deadlock can be tested with alternative actions. Thus, a different scenario can be tried to investigate testing behaviors of processes in which NRS is used to generate tests based on both refusal and acceptance of actions.

We hope that further studies of the NRS approach would fulfill our expectation for its contributions to development of distributed systems.

References

[AB 84] Austry D. and Boudol G., *Algebra de processus et synchronisation*, TCS 30, 1984.

[Abr 83] Abramsky S., *Experiments, power domains, and fully abstract models for applicative multiprogramming*, LNCS 158, 1-13, 1983.

[Abr 87] Abramsky S., *Observation equivalence as a testing equivalence*, TCS 53(3) 1987.

[ADF 86] Aceto L., De Nicola R. and Fantechi A., *Testing equivalences for event structures*, Nota interna B4-63, Istituto, di Elaborazione dell'Informazione, Pisa, 1986

[ADLU 88] Aho A.V., Dahbura A.T., Lee D. and Uyar M.U., AT&T Bell Laboratories, *An optimization technique for protocol conformance test generation based on UIO sequences and Rural chinese postman tours*, 8th IFIP on Protocol Specification,Testing, and Verification,1988.

[AR 87] Astesiano E., Reggio G., *Comparing direct and continuation semantics styles for concurrent languages*, Dipartimento di Matematica Universita'di Genova, Italy, LNCS 247.

[BALL 89] Brinksma E., Alderden R., Langerak R.,van de Langemaat J., Tretmans J., *Formal approach to conformance testing*, 2nd Intel. Workshop on Protocol Test Systems, Berlin(West),Germany,1989.

[Bau 88] Baudinet M., *On the semantics of temporal logic programming*, STAN-CS-88-1203,Stanford University.

[BBK 87] Baeten J.C.M., Bergstra J.A., Klop J.W., *Ready trace semantics for concrete process algebra with priority operator*, Computre Journal, Vol 30. No.6, 1987.

[BB 87] Bolognesi T. and Brinksma E., *Introduction to the ISO specification language Lotos*, Computer Networks and ISDN Systems, Vol.14, No.1, 1987.

[BC 86] Boudol G., Castellani I, *On the semantics of concurrency: partial orders and transition systems*, LNCS 249, 1987.

[BD 87] Budkowski S. and Dembimski P., *An introduction to Estelle: a specification language for distributed systems*, Computer Networks and ISDN Systems, Vol.14, No.1, 1987.

[BDC+ 89] Bowen T.F., Dwarok F.S., Chow C.H., Griffeth N.D., Herman G.E. and Lin Y.-J., *The feature interaction problem in telecommunication Systems*, Proc. of the 7th International Conference on Software engineering for Telecommunication Switching Systems, pp 59-62, 1989.

[Ben 89] van Benthem J., *Time, logic and computation*, LNCS 354, 1989.

[Bes 87] Best E., *COSY:its relation to NETs and to CSP*, LNCS 255, 1987.

[BHR 84] Brookes S., Hoare C. and Roscoe A., *A theory of communicating sequential processes*, JACM, 31, No.7, 560-599.

[BK 84] Bergstra J.A. and Klop J.W., *Algebra of Communicating Processes with abstraction*, TCS 37(1),77-121.

[BKP 85] Barringer H., Kuiper R., Pnueli A., *A composional temporal approach to a CSP-like language*, Proc. of IFIP Conference,the Role of Abstract Models in Information Processing, Vienna, 1985.

[BMO 84] de Bakker J.W., Meyer J.J.C. and Olderog E.R., *Infinite streams and finite observations in the semantics of uniform concurrency*, LNCS 194, 149-157, 1985.

[BN 92] Boreale M. and De Nicola R, *Testing equivalence for mobile processes*, Concur 92, pp 1--15, 1992.

[BS 80] Bochmann G.V. and Sunshine C.A., *Formal methods in communication protocol design*, IEEE Trans. on Comm.,Vol-28, No.4 Apr. 1980, 624-631.

[BS 83] Bochmann G.V.,and Sunshine C.A., *A survey of formal methods, Computer Networks and protocols*, Green P.E.,ed. New York,Plenum Press,1983.

[Boud 85] Boudol G. *Notes on Algebraic calculi of processes*, in Logics and Models of Concurrent Systems, NATO ASI Series F13, Springer Verlag, 1985.

[BP 89] Boyes T.T., Probert R., *Phase-directed testing of Esttle specification*, Proc. 2nd Intel. Workshop on Protocol Test Systems, Berlin(West), Germany, 1989.

[BR 83] Brookes S. and Rounds W., *Behavioural equivalence relations induced by programming logics*, LNCS 154, 97-108, 1983.

[Bri 88] Brinksma H., *A theory for derivation of tests*, Proc. 8th IFIP on Protocol specification, Testing, and Verification, 1988.

[Bro 83] Brookes S.D., *On the relationship of CCS and CSP*, LNCS 154, pp 83-96 1983.

[BSS 87] Brinksma E., Scollo G., Steenbergen C., *Lotos specifications their implementations and their tests*, Proc. 6th IFIP on Protocol specification, Testing, and Verification, 1986.

[Cas 88] Castellani I., *Bisimulations for concurrency*, CST-51-88, Ph.D. Thesis, 1988. University of Edinburgh.

[CCITT87] CCITT Recommendation Z.100, *Specification and description language SDL*, COM X-R15, 1987.

[CH 93] Cleaveland R. and Hennessy M., *Testing equivalence as a bisimulation equivalence*, Formal Aspects of Computing 5, pp 1--20, 1993.

[Chr 88] Christopher M.N.T., *Temporal ordering for concurrency*, ECS-LFCS-88-49, Department of Computer Science, University of Edinburgh, 1988.

[CTFM] IS 9646, Conformance Testing Methodology and Framework.

[CVI 89] Chan W.M., Vuong S.T., Ito M.R., *The UIOv-method for protocol test sequence generation*, Proc. 2nd Intel. Workshop on Protocol Test Systems, Berlin(West), Germany, 1989.

[Dar 82] Darondeau P., *An enlarged definition and complete axiomatization of observational congruences of finite processes*, LNCS 137, 47-62, 1982.

[DG 87] Darondeau ph. and Gamatie B., *Modelling infinitary behaviours of communicating systems*, Rapports de Recherche No.749,Unite de Recherche INRIA-RENNES.

[DG 87] Darondeau Ph. and Gamatie B., *A fully observational model for infinite behaviours of communicating systems*, LNCS 249, 153-168, 1987.

[DNM 88] Degano P., De Nicola R. and Montanari U., *A distributed operational semantics for CCS based on condition\event systems*, in Acta Informatica 26, 59-91 1988.

[EVD 89] Eijk P.H.J., Vissers C.A., Diaz M., *The formal description technique LOTOS*, Results of the ESPRIT/SEDOS Project. North-Holland.

[FB 91] Fujiwara S. and Bochmann G., *Testing non-deterministic state machines with fault coverage*, Proc. of the 4th International Workshop on Protocol Test Systems, Leidschendam, the Netherlands, 1991.

[FMCT] ISO/IEC JTC1/SC21/P.54 and CCITT SG X/Q.10, *Formal Methods in Conformance Testing*, Working Draft for review and comments, 1992.

[GB 87] Gerth R. and Boucher A., *A timed failures model for extended communicating processes*, LNCS 267, 1987.

[Gor 79] Gordon M., *The denotational description of programming languages*, Springer-Verlag,1979.

[GS 86] Graf S., Sifakis J., *A logic for the description of non-deterministic programs and their properties*, Information and Control 68, 254-270, 1986.

[GTW 78] Goguen J.A., Thatcher J.W., and Wagner E.G., *An initial algebra approach to the specification,correctness and implementation of abstract data types*, Current Trends in Programming Methodology, Vol.IV,.Prentice-Hall 1978.

[Gue 81] Guessarian I., *Algebraic semantics*, LNCS 99, 1981.

[GuLo89] Guillemot, R. and Logrippo, L., *Derivation of useful execution trees from Lotos specifications by using an interpreter*, FORTE'89, pp 311-320, 1989.

[Hen 81] Hennessy M., *A term model for synchronous processes*, Information and Control, 51, No 1, 58-75, 1981

[Hen 85] Hennessy M., *Acceptance trees*, JACM 32, 896-928, 1985.

[Hen 88] Hennessy M., *Algebraic theory of processes* ,The MIT Press, 1988.

[HM 80] Hennessy M. and Milner R., *On observing nondeterminism and concurrency*, LNCS 85, 1980.

[HM 85] Hennessy M. and Milner R., *Algebraic laws for nondeterminism and concurrency*, JACM 32, No.1, 1985.

[Hoa 69] Hoare C.A.R., *An axiomatic basis for computing programming*, CACM, Vol.12, 1969.

[Hoa 85] Hoare C.A.R., *Communicating Sequential Processes*, Prentice-Hall 1985.

[Hog 90] Hogrefe D., *Conformance testing of communication protocols in the framework of Formal Description Techniquess*, report, University of Bern, 1990

[Hog 92] Hogrefe D., *OSI formal specification case study: the Inres protocol and service*, revised, IAM-91-012, update 1992, Dept. of Computer Sci.,University of Bern.

[HU 79] Hopcroft J.F. and Ullman J.D., *Introduction to automata theory, languages and computation*, Addison Wesley, Reading MA

[Jon 82] Jones G., A timed model for communicating processes, Ph.D Thesis, Oxford University,1982.

[Kel 76] Keller R., *Formal verification of parallel programs*, Comm.ACM 19, 1976.

[KH 80] Kennaway J.K.and Hoare C.A.R., *A theory of non-determinism*, LNCS 85, 338-350, 1980.

[KSR+ 85] Koymans R., Shyamasundar R.K., De Roever W.P., Gerth R. and Arunkumar S., *Compositional semantics for real time distributed computing*, LNCS 193, 1985.

[Koh 78] Kohavi Z. *Switching and Finite Autonata Theory*, New York, McGraw-Hill, 1978.

[Lan 89] Langerak R., *A testing theory for Lotos using deadlock detection*, 9th IFIP WG 6.1 Intel Symp. on Protocol Specification,Testing, and Verification, The Netherlands 1989.

[Lee 78] Lee, S.C., *Digital circuits and logic design*, Prentice Hall, 1976.

[LBDW92] Luo, G. Bochmann, G., Das, A. and Wu, C., *Failure-equivalent transformation of transtion systems to avoid internal actions*, Information processing letters 44, pp 333-343, 1992.

[LG 81] Levin G., Gries D., *A proof technique for CSP*, Acta Informatica 15, 1981.

[Lig 64] Lightstone A.H., *The axiomatic method: an introduction to mathematical logic*, Prentice Hall, 1964.

[Mcl 84] Mclean J., *A formal method for the abstract specification of software*, JACM, Vol.31, No.3 1984, 600-627.

[MH 78] Miller E.and Howden W.E., *Tutorial: Software testing & validation techniques*, IEEE Catalog No. EHO 138-8, 1978.

[Mil 80] Milner R., *A Calculus of Communicating System*, LNCS, Vol 92, Springer Verlag 1980.

[Mil 81] Milner R., *A modal characterisation of observable machine behaviour*, LNCS 112, 1981.

[Mil 83] Milner R., *Calculi for synchrony and asynchrony*, TCS 25, 267-310.

[Mil 84] Milner R., *A complete inference system for a class of regular behaviours*, J. of Comput. and Syste. Sci. 28, pp. 439-466, 1984

[Mil 88] Milner R., *Operational and algebraic semantics of concurrent processes*, ECS-LFCS-88-46, Department of computer Science University of Edinburgh, 1988.

[MP 74] Manna Z.,a nd Pneuli A., *Axiomatic approach to total correctness of programs*, Acta Informatica 3, 1974.

[Mye 79] Myers G.J., *The art of software testing*, John Wiley and Sons Inc., 1979.

[MoPr 92] Monkewich O., Probert R., *TTCN - effective standard notation for specification of conformance test of communication systems*, Journal of Computer networks and ISDN, Vol. 23, No. 5, pp 417-438, (1992)

[NH 83] De Nicola R. and Hennessy M., *Testing equivalences for processes*, TCS 34, 83-133, 1983.

[NH 87] De Nicola R. and Hennessy M., *CCS without τs*, Proceedings of Taosoft, 1987, LNCS 250, 1987

[Nic 87] De Nicola R., Extensional equivalences for transition systems, Acta Informatica 24, 211-237, 1987.

[NPW 81] Nielsen M., Plotkin G., Winskel G., *Petri net, event structure and domains*, TCS, Vol.13, 1981.

[OH 86] Olderog E. and Hoare C., *Specification-oriented semantics for communicating processes*, Acta Informatica, Vol.23, 9-66, 1986.

[Old 87] Olderog E.R., *TCSP: theory of Communicating Sequential Processes*, LNCS 255,1987.

[PA 88] Probert R.L., Akhavan S., *Abstract conformance test results for open systems*, Proc. 2nd Intel. Symp. on Interoperable Information Systems, INTAP, 1988.

[Par 81] Park D., *Concurrency and automata and infinite sequences*, LNCS 104, Springer 1981.

[Pet 73] Petri C.A., *Concepts of net theory*, Mathematical foundation of Computer Science, Math. Inst. of Slovak Academy of Science, 1973.

[Pha 92] Phalippou M., *Test suite generation for the Inres protocol with TVEDA tool*, CNET/LAA/SLC/EVP, BP. 40 F-22301 Lannion CEDEX France.

[Pit 87] Pitt P.H., *A formal approach to conformance testing*, FORMAP WP B69, GB, 1987.

[Plo 76] Plotkin G., *A powerdomain construction*, SIAM J. Computing 5, 1976.

[Plo 81] Plotkin G., *A structured approach to operational semantics*, DAIMI FN-19, Computer Sci. Dept., Aarhus University.

[Pnu 81] Pnueli A., *The temporal semantics of concurrent programs*, TCS 13, 1981.

[Pnu 85] Pnueli A., *Linear and branching structures in the semantics and logics of reactive systems*, LNCS 194, 14-32, 1985.

[Pnu 86] Pnueli A., *Applications of temporal logic to the specifications and verifications of reactive systems: a survey of current trends*, LNCS 224, 1986.

[Phi 85] Phillips I., *Refusal testing*, TCS 50, No. 2, 1985.

[Pro 82] Probert R. L., *Life-cycle/grey-box testing*, Congressus Numeratium, Vol. 34, (1982) pp. 87-117.

[RR 86] Reed G.M. and Roscoe A.W., *A timed model for CSP*, LNCS 226.

[RB 81] Rounds W. and Brookes S., *Possible futures, Acceptances, Refusals and Communicating processes*, Proc. of 22nd IEEE Symp. on Foundations of Computer Science. 1981.

[Ray 87] Rayner D., *OSI Conformance testing*, Computer Networks and ISDN Systems 14, 1987.

[San 88] Sannella D., *A survey of formal software development methods*, ECS-LFCS-88-56, Department of Computer Science University of Edinburgh. 1988.

[SB 84] Sarikaya B. and Bochmann G.V., *Synchronization and specification issues in protocol testing*, IEEE Trans. on Comm., Vol.32, No.4, 1984.

[SB 82] Sarikaya B., Bochmann G.V., *Some experience with test sequence generation*, 2nd Intel. Workshop on Protocol Specification,Testing, and Verification, North-Holland, 1982.

[SD 85] Sabnani K.K., Dahbura A.T., *A new technique for generating protocol* tests, Proc. 9th Data Comm. Symp., IEEE Computer Soc. Press, 1985.

[Smy 78] Smyth M.B., *Power domains*, Journal of Computer and System Sciences 16, 1978

[SL 86] Sidhu D., Leung T., *Formal methods for protocol testing: a detailed study*, Aepr. of computer Science, Iowa State University, Iowa, 1986.

[SL 88] Sidhu D., Leung T, *Fault coverage of protocol test methods*, Proc. IEEE Infocom'88, pp 80-85, 1988.

[Sou 84] Soundararajan N., *Axiomatic semantics of CSP*, ACM TOPLAS 6, 1984.

[Sou 86] Soundararajan N., *Total correctness of CSP programs*, Acta Informatica 23, 1986.

[ST 87] Saracco R. and Tilanus P.A.J., *CCITT SDL: overview of the language and its applications*, Computer Networks and ISDN Systems, Vol.13, No.2, 1987.

[Ste 86] Steenbergen C., *Conformance testing of OSI systems*, MSc Thesis, University of Technology, T.H. Twente, The Netherlands, 1986.

[Sto 77] Stoy J.E., *Denotational semantics: the Scott-Strachey approach to programming language theory*, MIT Press, 1977.

[Str 85] Stirling C., *A proof-theoretic characterization of observational equivalence*, TCS 39, 27-45, 1985.

[Str 87] Stirling C., *Modal logics for communicating systems*, TCS 49, 311-347, 1987.

[Sti 89] Stirling C., *Tempoal logics for CCS*, LNCS 354, 1989.

[TV 87] Taubner D., Vogler W., *The step failure semantics*, LNCS 247, 1987.

[VSZ 92] Velthuys R. J., Schneider J.M. and Zorntlein G, *A test derivation method based on exploiting structure information*, Proc. 12nd IFIP Protocol specification, testing, and verification, 1992.

[Win 82] Winskel G., *Event structure semantics for CCS and related languages*, LNCS 140, 1982.

[Win 87] Winskel G., *Event structures*, LNCS 255, Springer, 1987.

[Wez 89] Wezeman C.D., *The CO-OP method for Compositional derivation of conformance testers*, Proc. 9th IFIP on Protocol Specification, Testing, and Verification, 1989.

Appendix A

The INRES Service and Its Lotos specification

This appendix presents the INRES Service in A.1 and its Lotos specification in A.2.

A.1 The INRES Service

The INRES (INitiator-RESponder) data communication service is a simple connection oriented service. It consists of a service Provider, an Initiator and a Responder. The service can be accessed from two SAPs(Service Access Point). On the one SAP(ini), the initiator-user must initiate a connection before sending data. On the other SAP(res), the response-user(entity) can accept the connection or reject it. After acceptance it can receive data from the initiating user. This INRES structure is shown in Figure A.1:

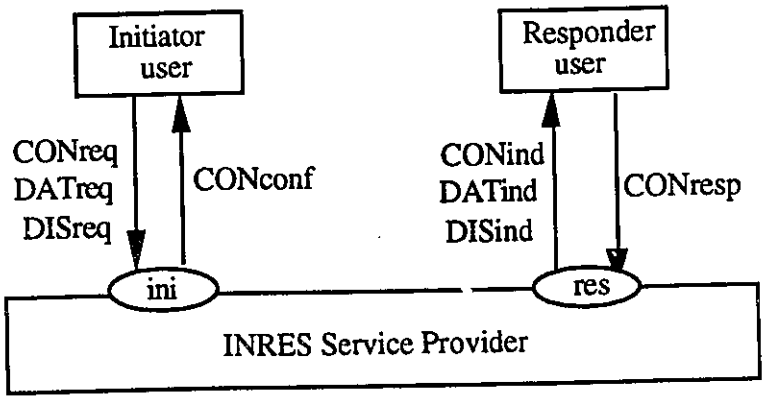


Figure A.1 INRES service structure

The SPs(Service Primitive) used for the interactions between users and provider are explained as follows:

-CONreq: request for a connection by Initiator user

-CONind: indication of a connection requst to Responder user

-CONresp: response to a connection request by Responder user

-CONconf: confirmation of a connection provided by Provider

-DATreq: request for sending a data unit from Initiator user

-DATind: Data indication from Provider to Responder user

-DISreq: request for a disconnection from Initiator user

-DISind: indication of a disconnection by Provider to Responder user

A.2 Lotos specification of INRES service

The three phases of the INRES service operation are connection establishment, data transfer, and connection release. First, a connection must be established before data can be transfered. Once the connection has been set up, data transfer can continue until either the provider or the responder releases the connection. Several interaction details are:

(1) A request for connection may be answered by either a connection confirmation or a disconnection indication from Responder user.

(2) During a data transfer phase, Initiator user can initiate a disconnect request if data transfer has been completed.

The Lotos specification of the INRES service is listed as follows:

```
Specification INRES(ini, res): noexit :=
    (Conestablishment(ini, res)
        >> Datatransfer(ini, res)
    ) >> INRES(ini, res)
where
```

```
proc Conestablishment(I, R): exit :=
    Connectrequest(I, R)
    >> (Connectaccept(I, R) ; exit
      [] ( Connectreject(I, R)
          >> Conestablishment(I, R))
       )
  where
    proc Connectrequest(I, R): exit :=
          I ! CONreq;  R ? CONind; exit
    endproc
    proc Connectaccept(I, R): exit :=
          R ! CONresp; I ? CONconf; exit
    endproc
    proc Connectreject(I, R): exit :=
          R ! DISreq; I ? DISind; exit
    endproc
 endproc /* Conestablishment*/
 Proc Datatransfer(I, R): exit:=
     (I ! DATreq; R ? DATind
     >> Datatransfer(I, R))
   [] Transferover; exit
  where
  proc Transferover(I, R): exit :=
      I ! DISreq; R ? DISind; exit
   endproc
endproc /* Datatransfer */
endspec /* INRES */
```

The Labelled rooted tree of the INRES service is shown in Figure A.2, where the black and grey states means that the service behaviours are equivalent at these states.

By projecting the system behaviour of INRES service into its two SAPs: ini and res, we obtain the two LRTs in Figure A.3 and A.4 which represent behaviours that can be observed and controlled at each service access point.

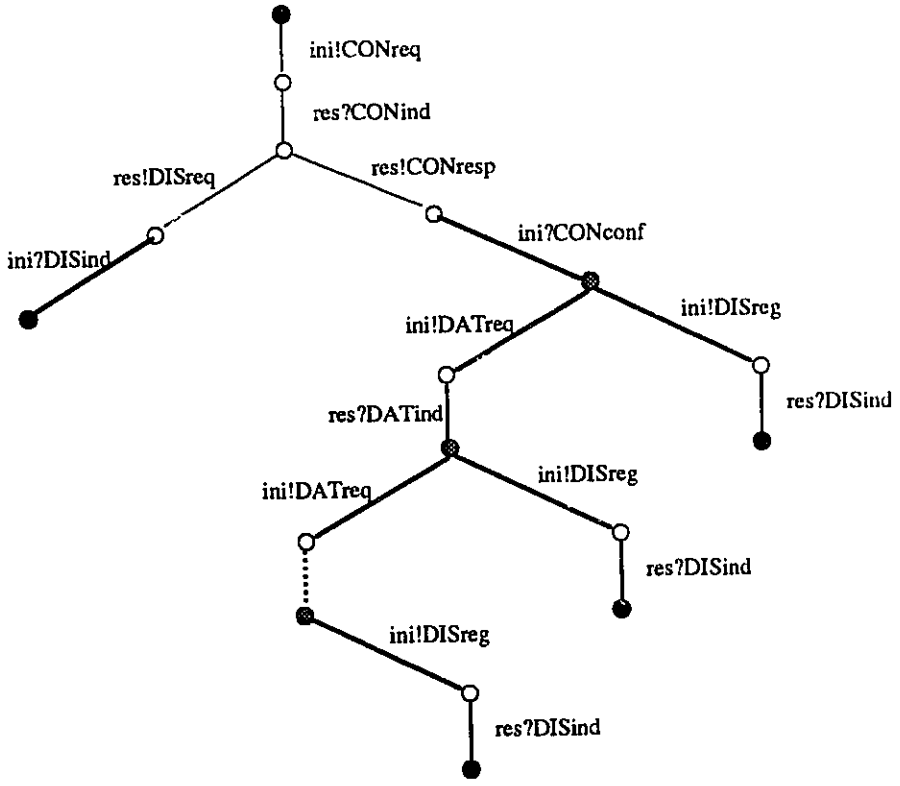


Figure A.2. Labelled rooted tree of the INRES service

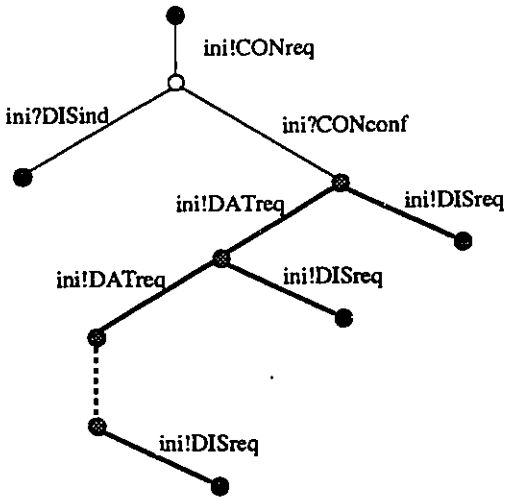


Figure A.3 Service behaviours observed at SAP: **ini**

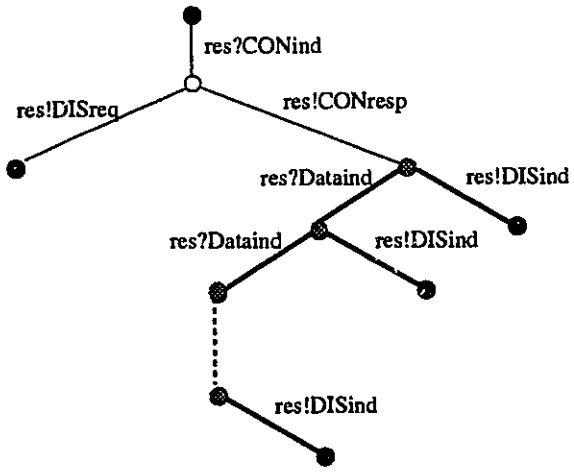


Figure A.4 Service behaviours at SAP: **res**