UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Extensions to coloured petri nets and their application to protocols

Jonathan Billington

Contents

III   Application to Specification

9   Communications Examples

10   Cambridge Fast Ring Service Specification

List of Figures

Summary

This dissertation develops a net theoretic specification technique for an area known as protocol engineering that covers the life-cycle of protocols. After surveying the application of net theory to protocol engineering, the fundamentals of the specification technique are presented. The technique is based on Jensen's Coloured Petri Nets (CP-nets).

To increase their expressive power, CP-nets are extended by including place capacities and an inhibitor function, leading to the definition of a class of extended CP-nets, known as P-nets. To allow the analysis techniques developed for CP-nets to be applied to P-nets, a transformation from P-nets to CP-nets is formalised and it is proved that it preserves interleaving behaviour. The transformation is based on the notion of complementary places (known from Place/Transition-nets) and involves the definition and proof of a new complementary place invariant for CP-nets. A class of P-nets is defined where true concurrency is preserved under the transformation.

A graphical form of P-nets, known as a P-Graph, is formally defined, drawing upon the notions developed for algebraic specification of abstract data types. Arc inscriptions are multisets of terms generated from a many-sorted signature. Transition conditions are Boolean expressions derived from the same signature. An interpretation of the P-Graph is given in terms of a corresponding P-net. In the P-Graph, concrete sets are associated with places, and likewise there are concrete initial marking and capacity multisets. P-Graphs are useful for specification at a concrete level, and allow classes of nets, such as CP-Graphs, many-sorted Algebraic nets and many-sorted Predicate/Transition nets, to be defined as special cases. They also provide the basis for a comparison with other high-level nets such as Predicate/Transition nets and Algebraic nets. An extended place capacity notation is developed to allow for the convenient representation of resource bounds in the graphical form.

Abstract P-Graphs are defined in a similar way to P-Graphs, but this time sorts are associated with places, and markings and capacities are defined at the syntactic level. This is useful for more abstract specifications (such as classes of communication protocols) and for their analysis.

Part of the motivation for the extensions to CP-nets has been to develop convenient constructs for the purging of a place's marking (or part of the marking), by the occurrence of a single transition. This is achieved by equating the inscriptions of the inhibitor and normal arc. Some convenient notation is developed for the P-Graph for purging parts of a place's marking.

Some simple communications-oriented examples are presented including queues and the Demon Game developed by the International Organisation for Standardisation as a test case for formal description techniques. A major case study of the M-Access Service of the Cambridge Fast Ring is specified with the P-Graph to illustrate the utility of a number of the extensions developed for P-nets.

Preface

Acknowledgements

thesis. Many thanks also to my children, David, Anna and Helena for their understanding. This dissertation is dedicated to them all.

Declaration

This dissertation is the result of my own work and is not the outcome of work done in collaboration. The extent to which the work of others has influenced this research is specifically stated in the dissertation. Moreover, this dissertation is not substantially the same as any that I have submitted for a degree, diploma or other qualification at any other University and no part of this dissertation has already been or is concurrently being submitted for any degree, diploma or other qualification.

Part I

Background

Chapter 1

Introduction

1.1 Background

Protocol Engineering was a term coined in the early 1980s to describe a field of study concerned with rigorous methods for the specification, design, verification, performance evaluation, implementation, testing and maintenance of communication protocols. Rigorous methods are required in the development of distributed systems where high quality products or services are required. In the provision of telecommunication services, customer satisfaction, reduction of costs associated with the elimination of specification errors after implementation, and ease of testing and maintaining systems are some of the reasons for developing rigorous approaches.

It is quite obvious that in a rigorous approach, mathematical techniques will be required to unambiguously specify requirements and protocol mechanisms; to allow the specifications to be analysed and transformed; and to provide rigorous methods for testing protocol implementations. There are a number of techniques that could be considered candidates, including state machines, Petri net based methods, process algebras, logics, set theory, programming languages, abstract data types and hybrids of these techniques.

In a particular environment, the choice of technique depends more on the background of the individuals concerned and the available tools rather than an objective choice based on a detailed comparison of techniques against some set of criteria. One reason for this is that a detailed comparison of the existing techniques would be a vast undertaking requiring the appropriate mathematical background and a good knowledge of protocols, a rare combination.

In my case the choice of technique has been influenced by a background in the application of net theory to the specification and verification of protocols. My interest in formal methods was stimulated by Professor Fred Symons some 10 years ago when he was then an Assistant Director of the Telecom Australia Research Laboratories. Fred had recently completed a PhD at the University of Essex (under a Telecom Australia Development Award) where he had developed the ideas of Numerical Petri Nets (NPNs) for the modelling and analysis of communication protocols during the years 1976 to 1978 [131].

During the late 1970's and early 1980's there was considerable interest in the development of layered protocol architectures, particularly the reference model for Open Systems Interconnection (OSI) [151]. The notion of a service specification [34] that expressed the requirements of the users of a protocol, arose. During 1981 and 1982, I made attempts to model the OSI Transport Service using NPNs [29,21]. This lead to the realisation that constructs were needed for the specification of complex queues and for the resetting of systems, corresponding to the emptying or purging of places in Petri nets. My colleague, Geoff Wheeler, developed NPNs further, incorporating ideas from Predicate-Transition nets and self-modifying nets, and made a first attempt at a formal definition in 1985 [145].

Concurrently with the NPN developments we were building an automated tool (known as PROTEAN) for the verification of protocols based on reachability analysis [149,32,147,31]. This tool has been used for the analysis of a number of complex protocols [129] with some success in detecting errors, mainly deadlocks.

Through the use of NPNs and PROTEAN, we had discovered a number of limitations of NPNs. Firstly, the queueing construct of NPNs was not formally defined and was rather adhoc allowing only FIFO (first-in-first-out) queues to be built. Clearly a more general approach was needed. Secondly, there was no clear mapping from NPNs to other nets. This meant that we could only employ reachability analysis techniques for verification, and could not take advantage of other analysis techniques (such as invariants, reductions) being developed for other high-level nets such as Coloured Petri Nets [87]. Thirdly, there was no means of specifying classes of protocols, as only concrete constructs were available. Finally there were other difficulties, such as the need for structuring mechanisms and formal refinement techniques.

This thesis addresses the first three of these problems by developing a high-level Petri net technique called P-nets with its associated graphical forms, the P-Graph and P-Graph Schema.

1.2 Research Aims

The principal aim of this thesis is to develop a high-level Petri net specification technique (P-nets) that has a sound mathematical basis and provides some useful constructs for the task of describing features found in protocols and other similar concurrent systems. (This includes constructs that allow classes of systems to be specified at an abstract level.)

A second aim is to provide transformations from P-nets to already existing high-level nets to enable their analysis techniques and automated tools to be used with the new technique.

A third aim is to test out the technique on a 'real-world' application, in this case the service provided by the Cambridge Fast Ring (CFR).

1.3 Scope

As is usual with research, the initial scope of the investigations are quite broad, in this case encompassing the field of protocol engineering, but with an emphasis on specification and verification. In order for the thesis to be of manageable size and for it to be completed within a reasonable time, it has been necessary to narrow the scope considerably.

Although a considerable amount of work had gone into specifying the Unison Data Link protocol [133], it was much more important to formalise the high-level net required for the specification. Thus the thesis does not contain this work, initial drafts of which can be found in [24].

Similarly work published on tools [31,30] and general comparisons with other techniques particularly the international standards, SDL [43], LOTOS [83] and ESTELLE [82], that appear in [31,25] have also not been included, although there is some overlap between chapter 2 of the thesis and [25,30].

1.4 Structure of the Thesis

The thesis consists of three parts. Part I contains background material with chapter 2 introducing the area of protocol engineering and then surveying the application of Petri nets to this area. Chapter 2 is a revision of a paper presented at the Eighth European Workshop on Application and Theory of Petri Nets, in June 1987 [28].

Part II, entitled 'Extending Coloured Petri Nets', provides the theoretical foundations required to develop the specification technique known as P-nets and its associated graphical forms, the P-Graph and Abstract P-Graph. It consists of six chapters (3 to 8) and is based on [22]. Chapter 3 provides some motivation for the development of P-nets, while chapter 4 defines Coloured Petri Nets (CP-nets), the technique on which P-nets are based. Chapter 5 provides the extensions we require and then defines P-nets in terms of CP-nets and the extensions. Chapter 6 describes the transformations from P-nets to CP-nets and the restrictions required on the P-net to ensure that true concurrency is preserved. Most of the material in Chapters 3 to 6 has been published recently [23]. The graphical form for the specification technique, known as the P-Graph, is defined in Chapter 7 which expands on [27]. The P-Graph is defined at a level appropriate for the specification of concrete systems. It is compared with several other high-level nets. The chapter concludes by defining, at a more abstract level, the P-Graph schema, which can be used for the specification of classes of systems. Interpretations of the P-Graph and P-Graph schema are given in terms of P-nets. Chapter 8 investigates a specialised construct useful for purging places and in general useful for manipulating markings with a single mode of a transition.

Applications of the P-Graph to the specification of communication systems is illustrated in Part III. Chapter 9 provides examples of queues and also specifies the 'Demon Game', an example used by the International Organisation for Standardisation in its work on formal description techniques. The main example

is provided in chapter 10, where the M-Access Service of the Cambridge Fast Ring networking system is specified. It revises the work published in [26]. This example illustrates the utility of a number of extensions and notations developed in Part II. The final chapter provides a summary of the contribution of the thesis and looks at areas of future work.

Chapter 2

Protocol Engineering and Nets: An Overview

2.1 Introduction

Communication protocols, the procedures that allow separate information processing systems to co-operate, are becoming more important as we enter the information age. These protocols are vital to the provision of advanced information services and the communications infrastructure required to support them, such as service networks, which carry customer traffic, and the management and signalling networks, which provide for their efficient operation. Important examples include those of Open Systems Interconnection (OSI), Integrated Services Digital Network Digital Subscriber Signalling System No. 1, Common Channel Signalling System No. 7, and many others standardised or being standardised by the International Telegraph and Telephone Consultative Committee (CCITT) and the International Organisation for Standardisation (ISO). These encompass the protocol families for wide area, local area and metropolitan area networks and for the interworking of networks and services.

As the provision of services becomes more sophisticated, the complexity of the communications protocols increases. It is now no longer possible to design high-quality protocols using engineering intuition and the cost of rectifying specification errors after implementation is considerable [85]. Errors may also lead to inadequate reliability of services and consequent customer dissatisfaction. It is in this environment that appropriate methodologies, techniques and computer aids become essential for the design and maintenance of communication protocols. The term *protocol engineering* [116] has been coined to describe the activities involved in the rigorous design and maintenance of protocols, using *formal* methods (i.e. those based on mathematics).

Early work on formal methods applied to protocol specification and verification is reported in [75,128]. The largest source of literature on aspects of protocol engineering is the proceedings of the annual symposium of IFIP Working Group 6.1 on Protocol Specification, Testing and Verification, published by North Holland [3]. Increasing interest in the formal specification of protocols over the last decade, has led to the development of international standards by ISO and

CCITT. A conference known as FORTE was initiated in 1988 [135], to provide a forum for publicising work on the standardised formal description techniques.

Protocol engineering is a part of software engineering, but it is hoped that a much more rigorous, mathematically based discipline can be found for protocol design, than presently exists for software engineering in general. This is because the processing is relatively straightforward. The main problems occur with communication, synchronisation and concurrency.

This chapter provides an overview of protocol engineering, including a description of the basic concepts and methodologies, and briefly surveys how net theory may be applied to the area as a whole. In section 2 we introduce the main protocol engineering activities and propose a methodology for protocol design in section 3. Section 4 addresses the requirements of mathematical techniques to be used for protocol engineering, while the merits of net theory for providing a possible approach are discussed in section 5. Finally we examine the types of computer aids that will be required by the protocol engineer.

The chapter builds on the work published in a number of papers, particularly [63,41,32,146] and is a revision of [28].

2.2   Protocol Engineering Activities

Protocol Engineering covers the whole spectrum of activities regarding the life-cycle of protocols. The major protocol design and development activities and how they are related are shown in figure 2.1. These activities are concerned with:

- Broad *requirements* of the users of distributed computing resources.

- High-level *architectural* design, normally developing a hierarchy of services to be provided by a set of protocols. Examples include the OSI Reference Model[42] and computer vendor architectures, such as IBM's Systems Network Architecture (SNA).

- Specification of the *service* to be provided at each level within the hierarchy, for example OSI Service Definitions.

- Specification of the protocols at each level of the hierarchy, usually subdivided into two parts: an *implementation-independent specification* suitable for international standardisation; and a refined *implementation specification*, taking into account environmental constraints.

- Target Implementations

The above is a list of tangible outputs, produced by the protocol engineer. In a rigorous approach, we also wish to show that the implementation does conform to the high-level requirements. There are two approaches to this problem: *synthesis* and *analysis*.

19

```
                    ┌─────────────────────────────────┐
                    │       USER REQUIREMENTS          │
                    └─────────────────────────────────┘
                                    │
                                    │ Structure
                                    ▼
                    ┌─────────────────────────────────┐
                    │      ARCHITECTURAL DESIGN        │
                    └─────────────────────────────────┘
                                    │
                                    │ Define
                                    │ Services
                                    ▼
                    ┌─────────────────────────────────┐
              ┌────▶│      SERVICE SPECIFICATION       │◀────┐
              │     └─────────────────────────────────┘     │
              │               │ Protocol                    │
      Verification            │ Synthesis           Performance
              │               ▼                     Analysis
              │     ┌─────────────────────────────────┐     │
              │ ┌──▶│     PROTOCOL SPECIFICATION       │─────┘
              │ │   └─────────────────────────────────┘
              │ │          │ Automatic
   Conformance │ │         │ Implementation
     Testing  │ │          ▼
              │ │   ┌─────────────────────────────────┐
              └─┴───│     TARGET IMPLEMENTATION        │
                    └─────────────────────────────────┘
```
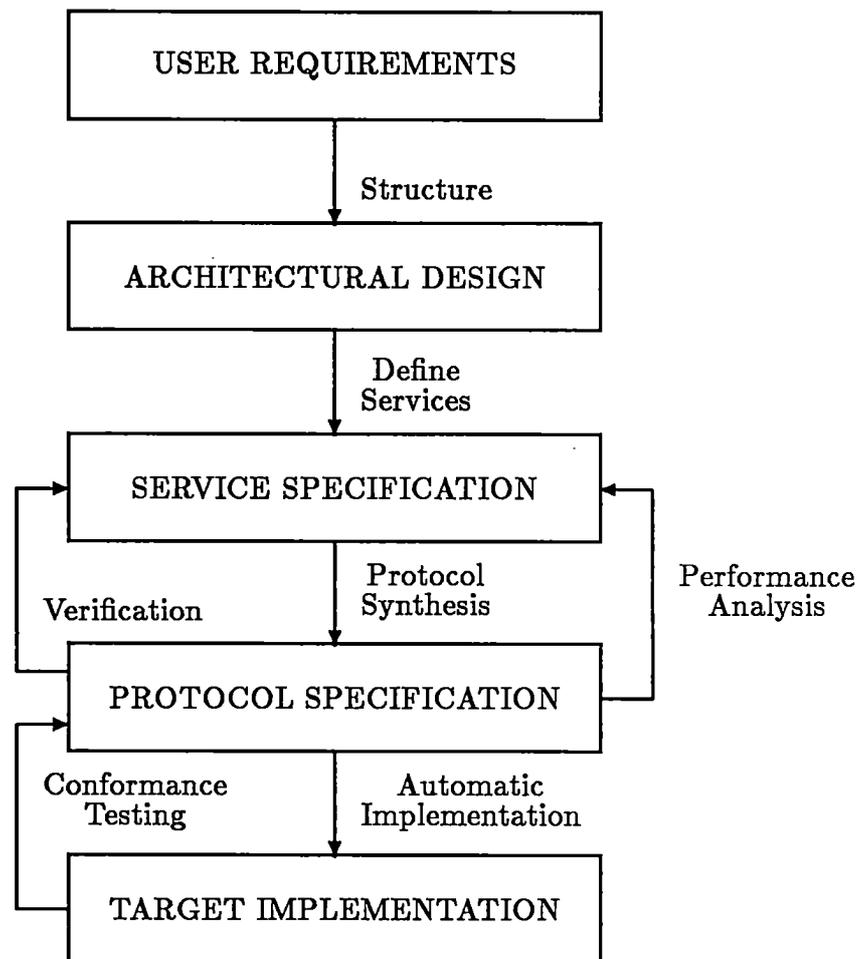
Figure 2.1: Major Protocol Engineering Design Activities

Synthesis provides the designer with a discipline of formal refinement of require-
ments, through specification to implementation, which preserves desired proper-
ties. Major areas of work include *Protocol Synthesis*, the discipline of deriving
a protocol specification from the adjacent service specifications, and *Automatic
Implementation*, the transformation of a protocol specification into an implemen-
tation (a compiler).

In a protocol hierarchy where we wish to check that a detailed specification is
consistent with a more abstract one (usually because we do not have a synthe-
sis procedure), analysis techniques are required to prove that desired properties
have been retained, and that undesirable ones have not been introduced. Ma-
jor activities include *Protocol Verification* and *Conformance Testing*. Protocol
Verification complements Protocol Synthesis. It is the process of proving that
the protocol specification provides the requirements stated in its service specifi-
cation. Conformance Testing complements Automatic Implementation. Various
forms of testing are undertaken to increase the confidence of the designer that
the implementation does conform to the protocol specification.

Other protocol engineering activities include *performance evaluation*, *mainte-
nance* and *protocol conversion*. Performance evaluation is concerned with the
real-time properties of throughput and delay as well as reliability. Protocol

maintenance is concerned with fixing errors in implementations and upgrading protocols as new versions or completely new protocols become available. Protocol maintenance requires the modification of the outputs listed above in a consistent manner. Protocol conversion [74] or *interworking* is required when it is necessary for two systems with incompatible protocol architectures to communicate, for example between an IBM SNA network and an Open Systems Interconnection environment. A method is required to translate between the two architectures. Similar problems of service definition, synthesis, specification, verification, performance analysis, direct implementation and testing occur in the development of a protocol converter.

2.3   Protocol Engineering Methodology

The diagram of figure 2.1 suggests the following top down methodology for protocol design, in analogy with the *waterfall* model of software development.

1. Gather and document requirements of the users of distributed applications.

2. Provide a high-level architectural design of a protocol hierarchy to meet the requirements. (Use existing architectures where these will satisfy 1.)

3. Define the service to be provided at each level of the protocol hierarchy. Two steps are required:

   - An *integrated* specification, where the behaviour of the users and provider of the service are specified jointly at a high level of abstraction. This defines the (global) set of sequences of *service primitives* (events representing communication between service user and service provider) and how they are related at each of the user interfaces.

   - A refinement of the integrated specification, where the behaviour of the users and the provider are separated by a well-defined but still implementation-independent interface.

4. Specify the protocol (or class of protocols) at each level of the hierarchy. This involves defining a protocol machine for each service user as a refinement of the service specifications. At this stage protocol verification and performance evaluation should be carried out. A further stage of refinement will be necessary to obtain an implementation specification. This will include specific details of the user/protocol entity interface as well as target implementation constraints and design decisions (what hardware, which operating system, what language, how parallel, what data structures, etc.).

5. The final stage of refinement is to produce the code for the target implementation from the implementation specification.

6. Conformance Testing and Debugging. The implementation is rigorously tested to see if it conforms to the protocol specification and any errors are corrected.

Top-down design has a fundamental flaw in that it assumes that requirements are well known and defined at the start of the design process. This is hardly ever the case for complex or large systems. However, top-down design augmented by iteration at all steps of the methodology, allows for the benefits of rapid prototyping in order to refine user requirements. Another problem with top-down design is the assumption that there are no parts of the system that already exist. The methodology should allow for the re-use of parts of existing protocol architectures where this is appropriate.

For complex protocols it is important to structure the service and protocol specifications in such a way as to maintain implementation independence (except for the implementation specification) and to increase readability and understanding. It will also be valuable from an analysis point of view.

2.4 Formal Techniques

This section addresses the needs for and requirements of formal techniques to support the protocol life-cycle.

2.4.1 Need for Formal Techniques

In order to provide a sound foundation for the design of protocols, it is essential that mathematical techniques are used, not only to provide unambiguous specifications, but also to allow specifications to be formally refined and analysed. This is required to ensure that internationally standardised specifications are of the highest quality to allow for the necessary interworking of heterogeneous systems. Ideally the techniques should facilitate all protocol engineering activities, particularly protocol synthesis, verification, automatic implementation and conformance testing. The use of mathematical techniques will allow the development of compilers for specification languages and the automatic derivation of test suites from specifications. This will provide for considerable productivity improvements in the development of implementations and their maintenance, thus reducing the cost of provisioning and maintaining information services.

2.4.2 Requirements of Formal Techniques

Surprisingly little has been published in the open literature on the requirements that need to be satisfied by a formal specification technique. A summary of important characteristics is as follows (further discussion can be found in Annex C of [4], in [53] and more generally in [51]):

- Well-defined syntax and semantics;

- Sufficiently expressive to describe the domain of protocol architectures, services and protocols;

- Analysable, to allow important properties (e.g. absence of deadlock) of protocols to be determined;

- Support the management of complex protocols (e.g. structuring capabilities);

- Support refinement;

- Support implementation independence (this implies support for concurrency and non-determinism and adequate abstraction mechanisms);

- Support all levels of the protocol life cycle, including verification, implementation and testing; and

- Support automation of design, verification, implementation and maintenance methodologies.

It is also most desirable that the technique can have several forms. A graphical form is desirable for readability. It allows all members of a protocol design and implementation team to communicate readily. Other forms more suited to mathematical manipulation are also desirable for analysis. Of course translations between the different forms are essential.

It is also desirable that a form of the technique exist which can be strongly related to the physical system that is to be modelled. The technique should not unnecessarily constrain the specifier, so that artificial constructs are avoided. This will lessen the difficulty of obtaining a realistic model of the desired system and will lead to intuitively appealing specifications.

2.5 Net Theory

Just as there are a large number of different programming languages, there are many different formal specification languages being developed and each one has strengths and weaknesses. Space precludes a comprehensive treatment of all the techniques and in order not to distract from the main purpose of the dissertation, I shall focus attention on the technique of choice: Petri nets. Some comments on other techniques, including the three international standards: CCITT's Specification and Description Language [43]; and ISO's Estelle (Extended State Transition Language) [82] and LOTOS (Language Of Temporal Ordering Specification) [83] are made by the author in [25].

Petri net theory [35,120,37,36,61] is founded upon a notion of concurrency; it expresses non-determinism simply and can be used to express system concepts at different levels of abstraction. Nets can be structured in a number of ways. For example, *Channel/Agency* nets [122], can be used to indicate a system structure. These nets can then be refined in various steps to define the dynamic behaviour of the system [6]. High-level nets also provide ways of grouping system features to indicate system structure in another way.

Nets may be presented in a graphical form which is easy to relate to physical systems. This makes learning and understanding the language relatively easy.

Petri nets have a solid mathematical foundation that has led to a large number of techniques being developed for their analysis. These include: reachability analysis; invariants analysis (a technique using linear algebra); transformations (including reductions) preserving desired properties; structure theory; formal language theory; synchronic distance; decomposition and equivalence of nets. The formal basis of nets allows them to be related to other models of concurrency that may be useful for the specification and analysis of distributed systems. For the latest information the reader is referred to [37,36,125,126].

The early experience gained in specifying protocols with Petri nets [131] revealed that place/transition nets (P/T-nets) [20] (also called Petri nets [115]) were too primitive to model complex protocols conveniently, as their use produced a proliferation of net elements. P/T-nets were also very inconvenient when representing information in message headers and compound state information (control state, sequence number, address and multiplexing data, time-out limits etc.). This problem was overcome with the development of 'high-level' nets [131,71,88,145,120] (i.e. nets where tokens are tuples of values, and arcs (and transitions) are inscribed by expressions). Later versions of Predicate/Transition nets [71] and Coloured Petri Nets [88] appear in [37]. More recent work [93,143,22,126] has shown how abstract data types can be incorporated within the high-level net framework. With the aim of increasing their expressive ability to specify protocols and services further, this dissertation develops extensions to Coloured Petri Nets, and formulates many-sorted high-level nets in Part II.

A weakness of nets is in the structuring of specifications and in providing equivalence transformations, but these problems are the subject of current research.

Net theory provides a suitable foundation for techniques and automated tools to support the protocol development life-cycle. The rest of this section briefly examines how net theory can be applied to each of the protocol activities mentioned in section 2.2, where *means/activity* nets [111] will be used to define the structure of architectures, services and protocols. (The development of an expressive high-level net for defining the behavioural aspects is addressed in Part II.)

2.5.1 Architectural Design

Architectural design is concerned with the definition of activities, the resources that are required, and the products that are produced as a result of the activities. A static structure evolves, showing the relationships between the activities and the resources/products.

Means/activity nets [111] or *channel-agency* nets [122] may be used to provide exactly this type of modelling. A simple example is given in figure 2.2, illustrating how the Open Systems Interconnection Protocol Architecture can be represented and refined. The general structure is given by Means/Activity nets. In Means/Activity nets, *activities* (actions) are represented by rectangles and *means* (resources) by ovals (or rectangles with rounded corners). An arrow from a *means* to an *activity* implies that the *means* is necessary for the *activity* to occur and arrow from an *activity* to a *means* implies that the *means* is modified

```
      ╭─────────────╮
      │    USER     │
      │  RESOURCES  │
      ╰─────────────╯
          │    ▲
          ▼    │
   ┌─────────────────────┐
   │ APPLICATION SERVICES │
   └─────────────────────┘
```
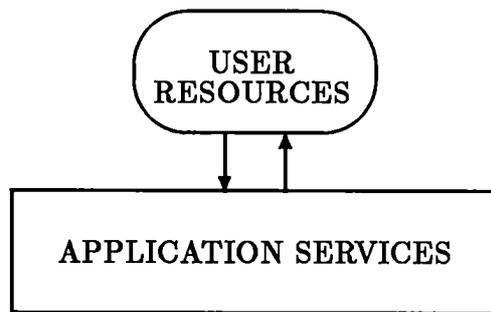
a: Provision of Application Services

```
      ╭─────────────╮
      │    USER     │
      │  RESOURCES  │
      ╰─────────────╯
          │    ▲
          ▼    │
   ┌─────────────────────┐
   │  APPLICATION LAYER  │
   │     ACTIVITIES      │
   └─────────────────────┘
          │    ▲
          ▼    │
      ╭─────────────╮
      │   P-SDUs    │
      ╰─────────────╯
          │    ▲
          ▼    │
   ┌─────────────────────┐
   │    PRESENTATION     │
   │  SERVICE-PROVIDER   │
   └─────────────────────┘
```
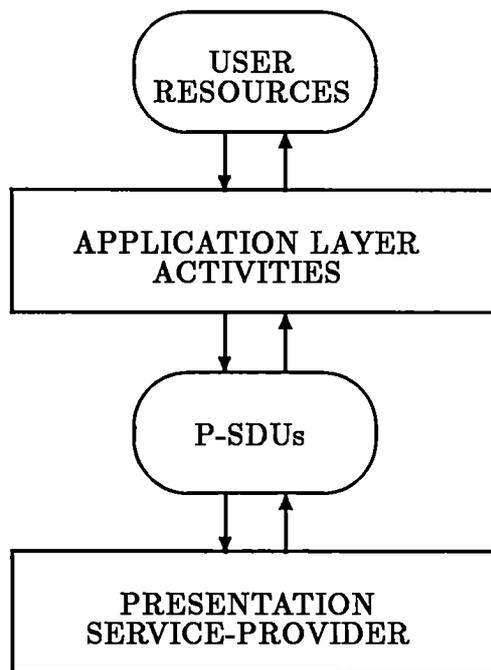
b: Refinement of a

Figure 2.2: Development of the Open Systems Interconnection Protocol Architecture using Means Activity Nets

by the *activity*, often by the production or consumption of a resource associated with the *means*.

Figure 2.2a shows that user resources (e.g. files) are required as input to a set of communication procedures called *application services*. In general these resources are distributed. Figure 2.2b shows a refinement of the application services, where it is now shown that application activities are supported by presentation services. Presentation Service-Data-Units (P-SDUs) are the resources that are produced and shared at the Application/Presentation boundary. In a similar way, the Presentation Service Provider may be refined into Presentation-Layer Activities supported by a Session Service Provider.

This process of refinement allows the complete OSI 7-Layer Reference Model [42] to be generated, where attention is focussed not only on the layer activities but also on the resources required for communication between the layers. Once the basic architecture has been designed, each of the component layers may be further refined.

An important part of the net representation is that communication across the interface is always brought to the attention of the designer. As the layer activities are refined, the abstract interface may also need to be refined. This is in contrast to other techniques, where the communication between blocks is hidden and assumed to be of a particular type (e.g. *rendez-vous*).

Of course architectural design is not limited to the specification of Protocol Architectures, but plays a major role in the specification of services and protocols.

2.5.2 Service Specification

Integrated Service Specification

A slightly different view to that presented above, is to consider each layer and the users of its service together. This can be done for each service separately.

The starting point is the naming of the service at say level N in the hierarchy (figure 2.3a). This may be refined into activities associated with interaction between user and provider, i.e. the occurrence of service primitives, and the N-level Service-Data-Units (SDUs) that are transferred between the service users. An example for two users is shown in figure 2.3b. This may be further refined into two state machines communicating with each other via two complex queues (figure 2.3c). Each state machine determines the set of sequences of service primitives at a local interface. The queue size and discipline governs the global set of sequences of service primitives. The state machines and their interaction via the two queues can be modelled using high-level nets. Examples for the OSI Transport and Network Service are given in [21,113].

Refined Service Specification

Figure 2.3a may be refined in a different way where the activities of the users may be separated out from the activity of the provider. An example for two users is given in figure 2.4a. It is similar to that of Fig 2.2b except that only two users are considered and they have been separated out. Also the users are very abstract activities that are only concerned with communication with the service provider. Figure 2.4a and figure 2.3b may be further refined into figure 2.4b. The purpose of this refinement is to provide a first step in the development of a protocol entity specification. Further refinement will allow a service primitive event to be refined into two events: control information will be passed between the user and its *local* provider to indicate that SDUs are ready for transmission or have been received. The event of transmitting the SDU into the service provider queue or of the user accepting the received SDU will signify the occurrence of the service primitive. This allows interleaving of many messages at the user/provider

(N)-SERVICE

a: Generic Service

User_A-
Provider
Interactions

SDUs in
Transit

User_B-
Provider
Interactions
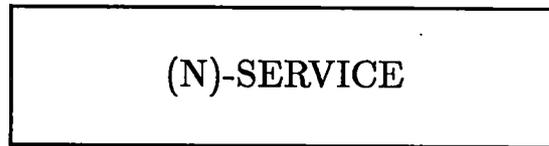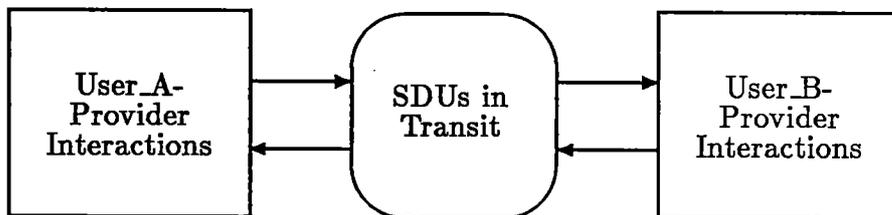
b: Refinement of a for two users (A and B).

A
state

A_Service
primitive
occurrences

SDU
queue
A to B

SDU
queue
B to A

B_Service
primitive
occurrences

B
state
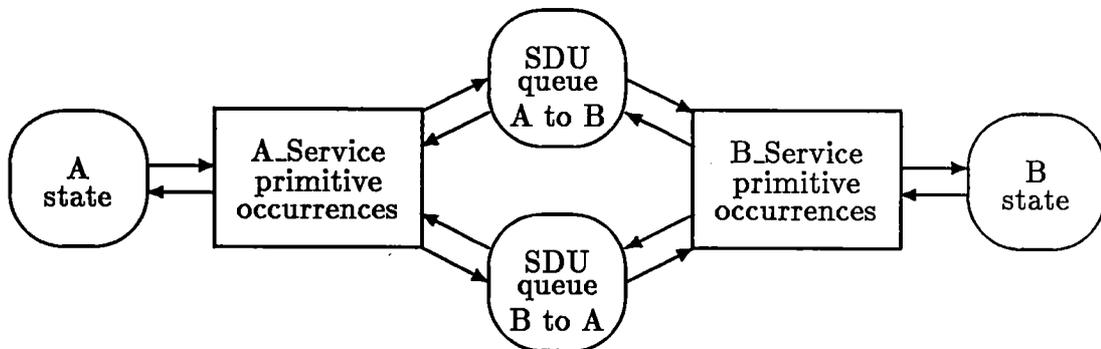
c: Further Refinement showing Service Primitive Occurrences.

Figure 2.3: Upper Level Refinement of a Generic Service

27
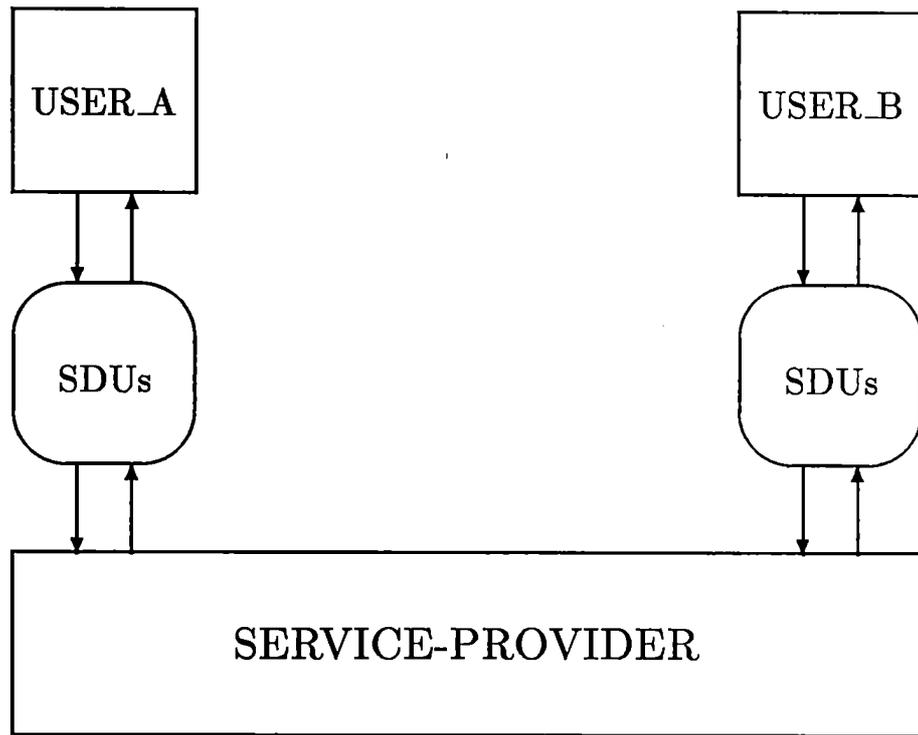
a: Separation of Service User and Provider
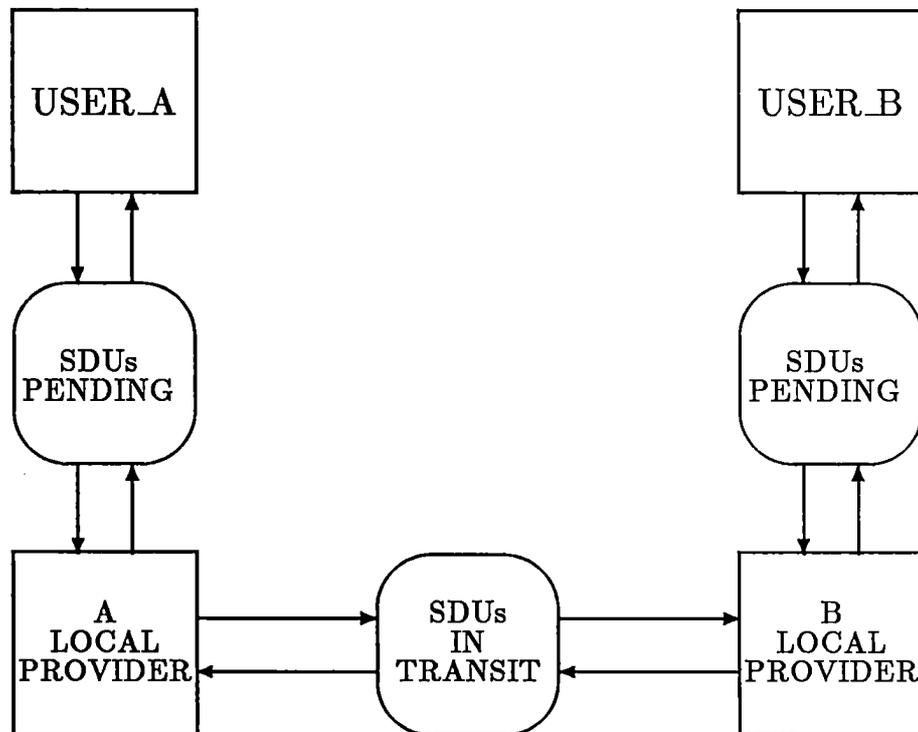
b: First Step for Protocol Entity Specification

Figure 2.4: The Service 'User/Provider' Model for Two Users

interface. (For example, a normal SDU can be signalled as ready, followed by an expedited SDU being signalled as ready. The local provider then has the option of acting on the expedited SDU before the normal SDU. Aborts and resets can be handled in a similar way.)

2.5.3 Protocol Synthesis

Given the service specifications for two adjacent layers of the hierarchy, these may both be refined into protocol entity activities communicating with their users and the underlying service provider. This is shown by the means/activity net of figure 2.5.

Considerable further refinement is then required to specify the structure and dynamics of the protocol entities. What is required is the definition of some synthesis rules, by which these refinements can be formally defined. Furthermore, the transformations should preserve the desired properties defined in the service specifications (e.g. sequences, safety, liveness, lack of duplication or loss).

General Net Theory may be able to provide some answers here via the notion of net morphisms and various notions of equivalence [35,150,144,107]. Recently published work on equivalence transformations and projections for Predicate/Transition nets [72], may prove to be useful. Further work in this area will be required before protocol synthesis is possible using net theoretic constructions.

2.5.4 Protocol Specification

Protocol specification is concerned with the refinement of figure 2.5. If we wish to define the protocol at level N of the hierarchy, the refined service specifications at levels N-1 and N, should be used as the starting point. The N-1 Service Provider and its interactions (interface) with the N protocol entity (a user) have already been defined, as have the interfaces with the N+1 users. The N service provider and N user specifications need to be refined to include an abstract description of protocol mechanisms [63] (to answer the question: *How are N-SDUs mapped onto (N-1)-SDUs (and vice-versa) to provide the required services at the desired quality?*).

The specification of protocols using nets (place/transition systems and high-level nets) is the most mature of the protocol engineering activities using net theory [59]. The earliest work on the modelling of protocols with nets was probably that of Merlin [105]. An early survey of the use of Petri net based techniques for the modelling of protocols is [59] with a more specific treatment in [17]. [41] presents a methodology for the specification of Open Systems Interconnection services and protocols using a high-level net technique called Product nets. A more recent survey of the use of nets in the specification and verification of protocols appears in [60]. High-level nets have been applied to the specification and analysis of many complex protocols and services [21,54,14,15,91,40,13,12,73,16,55,69,68,94].

Structural specification of protocol entities using means/activity nets has just begun [6]. A complete methodology for protocol specification using nets has yet
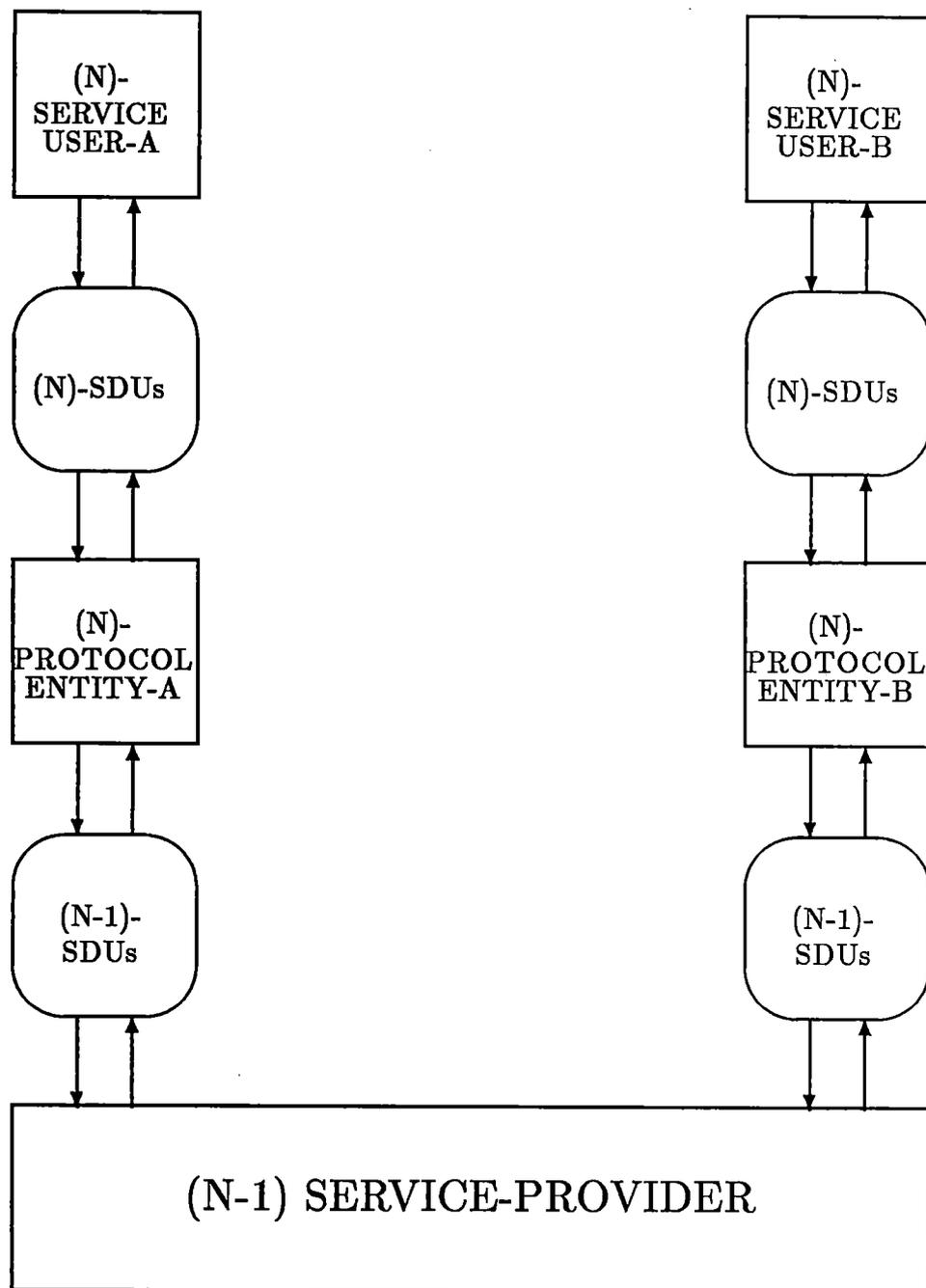
Figure 2.5: Environment for the Specification of the (N)-Protocol Entities

to be compiled. Guidelines for structural design and modelling are required as are the definition of a set of net building blocks for protocols [10].

Providing specifications at different levels of detail is also required. We would like to start with the most abstract specification, as implementation independent as possible (perhaps suitable for international standardisation) as the protocol may well be implemented on computing systems with very different characteristics. The abstract protocol and interface specifications then need to be refined to implementation specifications which are faithful to the implementation-independent specification. Rules for performing these refinement transformations appropriately are required. Again we appeal to General Net Theory for answers via

net morphisms. The work of Meseguer and Montanari [107] shows considerable promise here as they define refinement morphisms for place/transition nets.

2.5.5 Protocol Verification

A major purpose of developing a net specification of a protocol is to make it amenable to analysis. Protocol verification involves proving that the protocol entities operating over the underlying service provider is behaviourly equivalent to the integrated service specification (e.g. it preserves sequences of primitives and safety and liveness properties, and data to be transferred is not lost, duplicated or otherwise mutilated). Two techniques have been used with varying degrees of success: *Reachability Analysis* and *Invariants Analysis.*

Reachability analysis is readily understood and immediately reveals terminal states (e.g. deadlocks) as leaf nodes of the reachability graph, however considerable analysis of the reachability graph is required to discover whether or not other desirable properties exist. For example language analysis can be used to determine whether or not sequences of service primitives are preserved by the protocol [54,32]. Powerful logic-based query languages are being developed to interrogate the Reachability Graph [109,134,47,103] for general safety and progress properties.

Reachability analysis suffers from the problem of *state explosion*, but does provide results for moderately complex systems. In order to extend the scope of reachability analysis to more complex systems the state explosion problem is being tackled in a number of ways. Firstly, algorithms are being developed to reduce the reachability graph. The first of these methods uses equivalence classes of markings [79] which are derived from symmetries in the model. Another more general and very promising approach is the *stubborn set* method introduced by Valmari [139,138,141]. It takes advantage of the inherent concurrency of systems and does not rely on symmetries. It is also possible to combine these two approaches. Secondly, initial attempts are being made to simplify the analysis of systems composed of a set of closely coupled identical processes by reducing them to an equivalent smaller set [49]. Unfortunately these results only apply for closely coupled processes and cannot as yet be directly applied to network protocols where the communicating entities are normally loosely coupled via queues. What is required is some method of induction over protocol parameters, such as sequence numbers, number of retransmissions, buffer/queue sizes and the number of interacting parties. An initial attempt to provide a theory for this is given in [140]. Two other approaches are discussed in [48]: *hierarchical verification* which attempts to take advantage of the hierarchical nature of concurrent systems to abstract from details; and *lazy state generation* where only the states required to prove a particular property are generated. Finally some very recent results using symbolic model checking [39] show great promise in certain application domains.

Invariants analysis [96,71,88,120,70,87,104,142,143,76,56] provides an alternative to reachability analysis that avoids the state explosion problem. It has been used successfully in a number of examples [7,17,18,55]. Unfortunately, there is no guarantee that an assertion expressing a desired protocol property can

be proved using invariants [8]. There is also the complementary problem of interpretation of the invariants found, usually a large number when dealing with complex protocols.

Specification testing is an activity that attempts to discover protocol errors rather than prove correctness [77]. This may be achieved by simulating the net, where only parts of the reachability graph are investigated. This may be done automatically [77,86] or interactively [15].

The major problem with the analysis of protocols is their complexity. In order to analyse highly complex protocols, it will be necessary to have some form of compositional approach. The specification will be decomposed into a set of communicating entities. Each entity will be analysed separately for desired properties. The total system may then be synthesised by combining the entities according to some rules of composition which guarantee that the desired properties are maintained. Promising work in this area for Place/Transition nets using abstract algebra and category theory can be found in [150,107]. Very recent progress has been made on compositional state space generation [137] and compositional model checking [50]. Further research is required to apply these ideas to high-level nets and to the analysis of protocols.

2.5.6 Performance Evaluation

Performance evaluation involves the prediction of the performance of the protocol (in terms of delays, throughput and failure probabilities) from its specification. This requires the introduction of time and probabilities into the net model. Various stochastic and timed Petri net models have been introduced [117,106,130,108,119,99,2,101]. More recently these ideas have been incorporated into high-level nets [97,46,62]. Some of these nets have been used for the performance analysis of a number of protocols [67,119,2,100,102].

Unfortunately these models alter the firing rule of the net, thus reducing the possibilities for behavioural analysis. In [124], the concept of a clock has been introduced via the construction of a pulse generator using nets. The duration of an event can then be measured in terms of the number of ticks of the clock. This may lead to an integrated approach to the specification and analysis of both performance and behavioural aspects of protocols, at least to the extent where discrete time is involved.

2.5.7 Automatic Implementation

The field of automatic implementation is still in its infancy. Some reasonably successful attempts have been made to provide semi-automatic translation of protocol specifications based on state machine languages into compilable code on a single sequential processor [33]. The translation from the state machine language to code is a relatively straight forward task and produces about 40% of the code. The remainder is hand coded to suit the particular operating environment. (A summary of some of the tools for the International Standards, particularly SDL and Estelle compilers, is given in [25].)

Another example concerns a project to implement the OSI Transport Protocols (Classes 0,1,2 and 3) based on Numerical Petri Net specifications, which lead to the development of a prototype NPN interpreter by an Australian software house [148]. This has been developed further into a high-level net to C compiler, called PROMPT (PROtocol Manufacture, Prototyping and Testing) [30,81].

The more general problem of providing a translator from a concurrent specification in nets onto an implementation programming language (e.g. ADA) suitable for programming a multiprocessor appears to be an order of magnitude more difficult. Some attempts at this are reported in [110,38,52,146,132].

The importance of automatic implementation lies in the possibilities for rapid prototyping and rapid protocol maintenance. It is expected that progress in this area will be reasonably rapid as the rewards are considerable.

2.5.8 Conformance Testing

To provide comprehensive conformance testing a set of sequences of protocol interactions need to be defined. This set of sequences can be determined from the reachability graph generated from the composite protocol specification (figure 2.5). This information can be employed for rigorous testing of implementations. It is likely that the full reachability graph for complex protocols will not be available. In this case, interactive simulation of the net specification can be used to generate the test sequences [31]. Manual testing of OSI transport protocols based on Numerical Petri Net specifications using this approach, has been undertaken with considerable success [11].

This approach opens the door to semi-automatic testing, where the specification is run interactively and controls the sending of packets (or more generally Protocol Data Units) to the implementation under test. A trace is kept of all events. The sequence of message exchanges is continued until

- the received message is not one that is allowed by the specification;

- no message is received within an expected time; or

- the operator ends the session.

Alternatively this could be further automated, where the choice of the event occurrences could be made randomly, instead of by an operator.

2.5.9 Protocol Conversion

Formal methods are just starting to be applied to protocol conversion, with the first papers appearing in 1986 [95,112], using state machine techniques. As far as the author is aware, there are no papers using nets to tackle the problems of feasibility, specification, synthesis, optimum methods for overcoming mismatches in the protocols, performance analysis and implementation issues. It is an important problem as there are a large number of protocol architectures already

implemented and the desire to communicate between these heterogeneous networks will increase. Hopefully net theory will be able to provide the basis of a methodology for protocol conversion. This hope is based on the ability of net theory to provide some answers in the design and development of a single protocol architecture and that the techniques used in [95,112] are based on state machines and protocol verification techniques.

2.6 Computer Aided Tools

This section briefly discusses the range of computer-aided tools required to support protocol engineering activities. An integrated set of these tools constitutes a *protocol engineering workstation* or *workbench*. Perhaps the most advanced of these are: the tools being developed under ESPRIT project GRASPIN [92,84]; PACE (Prototyping, Analysis and Code-Generation Environment) [57]; and the CPN Palette package of Meta Software [1].

2.6.1 Specification Manager

With the design of any complex protocol a large number of specifications are required as the system is refined. The specifications include architectural, service and protocol descriptions at different levels of detail. It is important to be able to manage these specifications in order to provide a consistent set, particularly in the face of iteration. Automatic tools to assist the designer are essential and may include graphical and textual editors, syntax and static semantic checkers, morphism display, data dictionaries and specification libraries. Unfortunately no comprehensive system for nets is known to the author, however a number of tools are available covering parts of these requirements [65,89,1,118,114,31,80].

2.6.2 Simulator/Animator

A simulator is required to allow a specification to be executed. Graphical display of the dynamics (animation) also allows considerable increase in understanding of the system being designed. Two modes of simulation are valuable: interactive (where the user can choose which events are to occur) and automatic, where events can be chosen according to some previously set priorities and probabilities. A number of Petri net tools are available for this purpose [65,1,31,118,80].

2.6.3 Analyser/Verifier

This tool provides automatic assistance in the analysis of specifications. It will provide a set of analysis tools to allow the specification to be verified against its requirements and to allow the specification to be debugged when errors are found. There are a large number of Petri net tools being developed for this purpose [89,65,114,31,118,127].

34

2.6.4 Performance Analyser

An automatic tool is required which will allow the performance of a system to be predicted from a specification which includes stochastic information. A number of these analysers exist for various types of stochastic Petri net [2,65].

2.6.5 Translators

Facilities are required, based on net morphisms, for protocol synthesis and to assist with the software design. No tools are presently available. Tools for the direct implementation of specifications are also required [57,30,81].

2.6.6 Conformance Tester

Considerable computer assistance will be required to perform efficient and comprehensive conformance tests based on test suites derived from net specifications. Few tools are presently available [84].

2.6.7 Debugging

Once an error has been found it is essential to trace it back to an implementation or specification fault. Tools are required for this purpose. In an integrated tool set, which includes a compiler, only a symbolic debugger operating at the specification level is required.

2.7 Conclusions

The concerns of protocol engineering have been summarised and an attempt has been made to illustrate how net theory can be used to provide a rigorous approach to the field. The set of computer aided tools required for protocol engineering has been discussed. We are now at the stage where some sophisticated specification and analysis tools exist, however, considerable further work is required to provide a comprehensive protocol engineering workstation.

Further research is required in all aspects of protocol engineering, particularly protocol synthesis, the analysis of complex protocols, automatic implementation, conformance testing and protocol conversion. Net theory provides a suitable foundation for the specification of protocol architectures, services and protocol entities, although definition of a conveniently expressive high-level net, and details of structuring and a complete specification methodology still need to be determined.

The rest of this thesis is concerned with the definition of a suitable high-level net for the specification of the dynamic behaviour of protocols and services, but does not tackle the problems of structuring large specifications nor the need for specification methodologies.

Part II

Extending Coloured Petri Nets
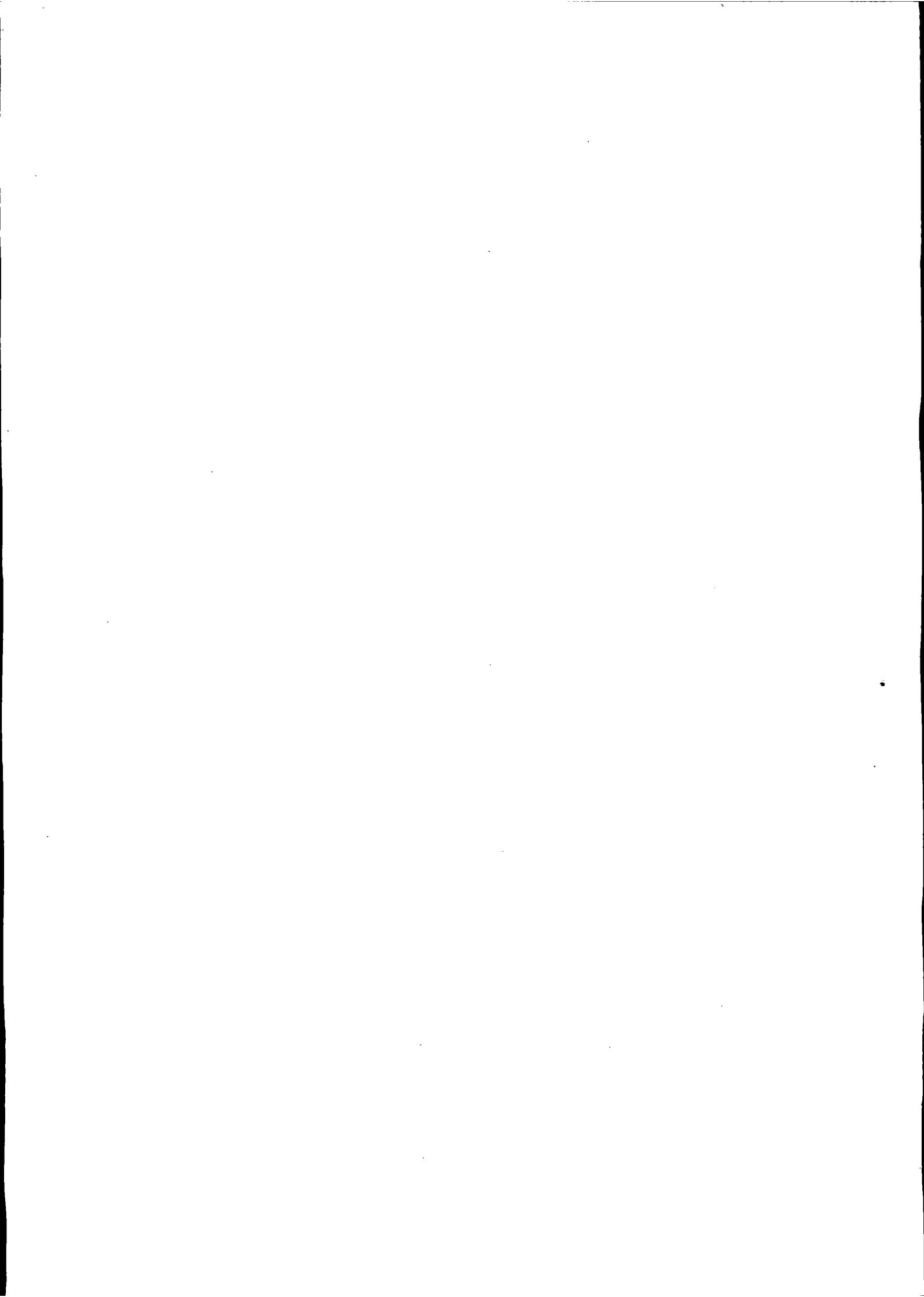
Chapter 3

Introduction

The limitations of Petri nets for specifying complex systems have been well known for the last 15 years. During the latter part of the 1970's and particularly during the 1980's *high-level* nets [131,71,88,120,70,87,123] have been developed in an attempt to overcome this problem. It is also recognised that high-level nets also have their limitations in expressive power and this part tackles this issue. To model many concurrent systems naturally, we would like to express bounds on the capacity of places (without having to use the net structure to do so) and also to express enabling conditions based on the absence of, or a limit on, the number of tokens in places. This idea is expressed by the *inhibitor arc* extension to Petri nets [115] and it can be generalised for high-level nets as will be shown in chapter 5.

The motivation for the introduction of the inhibitor function is not only to provide for more compact descriptions but also to allow for the atomic manipulation of markings. For example, we may wish to empty (*purge*) a place of all (or a subclass) of its tokens on the single occurrence of a transition mode. This aspect is the subject of chapter 8.

In order to be precise, it is necessary to provide formal definitions of these extensions. Further, it is also important to show how these extensions can be mapped back to the original high-level net, so that already existing results for analysis can be used.

Starting with Coloured Petri nets (CP-nets) [88,87] we add place capacities and an inhibitor function to define a class of extended CP-nets, called P-nets. We then define a transformation in chapter 6 that maps a P-net back to a CP-net, under the condition that the capacity of a place must be finite when there is an inhibitor associated with it. It is proved that the transformation preserves interleaving behaviour, in that the (single step) reachability graphs of the P-net and its equivalent CP-net (ECPN) are isomorphic.

A simple example shows that, in general, true concurrency is not preserved. It is shown that the enabling condition for the ECPN is either the same or more restrictive than that of the P-net. A theorem establishes the class of P-nets under which true concurrency is preserved by the transformation. As expected, this class includes *capacity* CP-nets as a special case.

The graphical form of nets provides strong support for the intuition of designers and specifiers, as it allows for the visualisation of flow of data and control. Hence an important goal of this part is to define a graphical form of P-nets and this is tackled in chapter 7. In the development of the graphical form, the ideas of signatures and many-sorted algebras are borrowed from the algebraic specification of abstract data types [64]. This allows us to develop two graphical forms, the P-Graph and P-Graph Schema, at different levels of abstraction. The P-Graph provides facilities for the development of concrete specifications where the sets and functions that are required are already known. On the other hand, the P-Graph Schema facilitates the specification of abstract systems where only the names of sets, variables and functions are used together with their declarations. A single P-net provides an interpretation for the P-Graph, whereas a class of P-nets can be used to interpret the P-Graph Schema. The P-Graph allows other forms of high-level nets, such as Predicate/Transition nets [70] and Algebraic nets [123] to be considered within its framework.

Some of the ideas formalised here stem from the author's association with the development of Numerical Petri Nets [131,130,29,145,31] for the specification of communication protocols and services. Specifically the idea of purging and the need for elegant representations of queues are due to the author [29]. The formalisation presented here is due to the author except where specifically noted otherwise.

The name **P-net** has been coined for the net defined here, to reflect its genesis in the specification of protocols (P-net is an abbreviation for Protocol-net), but also because it is sufficiently different from other *high-level* nets to warrant a new name. Although the design of P-nets has been influenced by the author's experience with the protocol domain, it is believed that they are generally applicable for the modelling of distributed systems including the functional specification of telecommunication systems and services.

This part assumes a familiarity with nets [120,115,20] and also makes extensive use of multisets. Multisets may be considered as a special class of vectors, sometimes called *weighted-sets*. Definitions of sets together with multiset and vector notation and operations are gathered together in Appendix A. Our definition of CP-nets is a little different from that of [87], taking a more global view and following that of [150]. We therefore proceed in a tutorial manner, introducing the notation and concepts gradually.

3.1   P-net Design

The aim of this part is the development of a formal technique to be used in systems engineering and in particular for protocol engineering. This section briefly discusses some of the requirements of such a technique.

The development of P-nets has been guided by the following conflicting requirements:

- Expressive ability

- Analytic power

- Simplicity

There are two parts to expressive ability: modelling power and modelling convenience. Modelling power is the ability of a technique to model a class of systems (see [115] chapters 7 and 8 for example) whereas modelling convenience refers to the elegance or conciseness of expression when representing a system's behaviour. It is well known [115] that there is a trade off between modelling power and analytic power. (We increase modelling power at the expense of our ability to analyse the model; more questions become undecidable.) In general we would like there to be just enough modelling power for our requirements (so that analytic power can be maximised).

It is essential that the technique be able to express all of the properties that we wish to express about protocols and their services. P-nets include the inhibitor extension which raises its modelling power to that of a Turing Machine. This allows us to model any system that is implementable. It turns out that most protocols do not require such modelling power. The technique allows for a whole range of modelling power, from state machines to Turing Machines, depending on the net structure and the use of inhibitors. The appropriate modelling power can then be chosen to suit the application.

We would also like to increase the modelling convenience of the technique, so that important elements in the application can be modelled relatively easily. We can add constructs (notation) to do this which can be defined in terms of the basic elements of the technique. The addition of too many constructs makes the language more complicated and difficult to learn and we need to strive for a few powerful constructs.

P-nets have been developed with these goals in mind. When considering a language for protocol engineering, we need to consider the perspectives of the specifier, analyser and implementer of protocols. It is hoped that P-nets provide a reasonable compromise.

3.2   The Nature of High-Level Nets

In applications we often find that we wish to model records (vectors) of information. For example, in protocol specification we wish to model messages (Service Data Units (SDUs) and Protocol Data Units (PDUs)) comprising many fields, and compound state information (major states, housekeeping variables, etc). Thus information is structured. It is useful to be able to express this structure in our specification language.

In place/transition nets (P/T-nets), this structure can only be expressed by complex labelling of places by values. One place is required for each value of the domain of a data structure and there is no means of grouping places associated with the same data structure. The structure is lost in an amorphous sea of net elements. This also leads to an explosion of P/T-net elements for even moderately complex applications, rendering the graphical form useless.

39

High-level nets [71,70,88,87] have overcome this problem by providing a mechanism for grouping sets of (P/T-net) places (transitions) together and considering them as entities in their own right, but still usually referred to as (high-level) places (transitions). The places in the high-level net are now typed (implicitly or explicitly) by the domain of the data structure and tokens residing in the place take on values from the domain. Tokens may now have a structure, and because they represent a value, are no longer anonymous and are often referred to as *individual* tokens. The grouping of transitions allows sets (schemes) of similar actions to be referred to by the one transition. The inscriptions on arcs are no longer integers but involve multisets of terms, which when evaluated are multisets over the domain associated with the place. The price paid for structuring the net is the increased complexity of net inscriptions. The considerable advantage is being able to model systems in a much more compact and intuitively appealing way. For example PDUs and SDUs, compound states and queues can all be represented by tokens in appropriately typed places.

Coloured Petri nets (CP-nets) [88,87] have been chosen as the basis for the development of P-nets because of their generality (arbitrary grouping of places and transitions, c.f. restricted arity of predicates in PrT-nets); their transparent relationship to P/T-nets; the possibility of using the ideas of algebraic specification of abstract data types within the same framework [143]; and the increasing range of analysis possibilities [142,104].

Chapter 4

Coloured Petri Nets

This chapter introduces the basic features of Coloured Petri nets, including the colouring of places and transitions, their pre and post maps, the net marking and transition rule and culminates in a formal definition. It also relates the definition provided here to the one used by Jensen [87].

4.1 Colouring Places and Transitions

Consider a set of (high-level) places, S, and transitions, T. We wish to associate a set with each place. Let there be a set, D, comprising the sets associated with the places. This set determines a structure on the underlying place/transition net. (We allow the sets associated with each place to be complex sets, such as unions of product sets, e.g. if $D \in D$ then we may have $D = (D_1 \times D_2) \cup D_3$.) We define a *place grouping function*, GP, which associates a set in D with a place in S.

$$GP : S \longrightarrow D$$

We also associate a set of *occurrence modes* with each transition and in a similar way, define the *transition grouping function*

$$GT : T \longrightarrow O$$

where O is the set of all the occurrence mode sets. (In most applications, we can derive O from D)

For economy of definition, we can use a single grouping function, called the *Colour Function* by Jensen [87].

$$C : S \cup T \longrightarrow C$$

where $C = D \cup O$ and for all $s \in S$, $C(s) = GP(s)$ and for all $t \in T$, $C(t) = GT(t)$. Jensen uses the term *occurrence-colours* for *occurrence modes* and the term *token-colours* to describe the members of a set in D, while we shall call them *place colours*. We shall refer to C as the *structuring* set and the sets in C as *colour sets*.

4.2 Pre and Post Maps

We follow the approach taken in [150] for defining pre and post mappings.
We define two sets:

$$TRANS = \{(t,m) \mid m \in C(t), t \in T\}$$

$$PLACE = \{(s,g) \mid g \in C(s), s \in S\}$$

TRANS is the set of transitions in the *unfolded* P/T-net and likewise *PLACE* is the unfolded set of places. We shall refer to elements of *TRANS* as *transition modes*, and elements of *PLACE* as *underlying* places or just simply places when the context is clear. (In this thesis, the terms *unfolded* and *underlying* are used interchangeably.)

We form the set of multisets over *TRANS* and *PLACE* and denote them by $\mu TRANS$ and $\mu PLACE$ respectively (see Appendix A).

We may now state the relationship between places and transitions by two mappings:

$$Pre, Post : TRANS \longrightarrow \mu PLACE$$

Thus *Pre* and *Post* return a multiset of underlying places for each transition mode.

4.3 Net Marking and Transition Rule

A marking multiset can now be defined for all places by

$$M \in \mu PLACE$$

and we shall denote the initial marking by M_0.

Sometimes it will be useful to consider the marking of a particular place, s. This can be achieved by partitioning the marking multiset according to places. We define, for $s \in S$ and for all $g \in C(s)$,

$$M_s \in \mu(\{s\} \times C(s))$$

such that $mult((s,g),M_s) = mult((s,g),M)$. ($mult(x,A)$ is the multiplicity of x in the multiset A - see Appendix A.) We have $\sum_{s \in S} M_s = M$, where $\sum$ refers to multiset addition (defined in Appendix A).

We can consider places to be *marked* by a multiset of tokens, $M(s) \in \mu C(s)$, where for all $g \in C(s)$, $mult(g, M(s)) = mult((s,g), M)$. (For any $s \in S$, a token is a member of $C(s)$.)

The transition rule follows immediately from [150]. We consider a finite multiset of transition modes, $T_\mu \in \mu TRANS$. By a linear extension of the above mappings, we have

$$Pre', Post' : \mu TRANS \longrightarrow \mu PLACE$$

where

$$P'(T_\mu) = \sum_{tr \in TRANS} mult(tr, T_\mu) P(tr)$$

with $P = Pre$ or $Post$. $\sum$ is multiset addition and juxtaposition is used for scalar multiplication of a multiset (see Appendix A).

We may now define a *step* (the simultaneous occurrence of a finite multiset of transition modes, T_μ) from marking M to marking M' as follows:

$$M[T_\mu\rangle M' \; \textit{iff} \; Pre'(T_\mu) \leq M$$

with

$$M' = M - Pre'(T_\mu) + Post'(T_\mu)$$

where '$\leq$' is multiset comparison, and '$-$' and '$+$' are interpreted as multiset subtraction and addition respectively (see Appendix A).

4.4 Definition of CP-nets

We are now in a position to provide a definition of CP-nets as a summary of the discussion above.

4.4.1 Definition

A **CP-net** is a structure $CP = (S, T, \mathcal{C}; C, Pre, Post, M_0)$ where

- S is a finite set of places

- T is a finite set of transitions disjoint from S ($S \cap T = \emptyset$)

- $\mathcal{C}$ is a finite set of non-empty colour sets, the *structuring* set

- $C : S \cup T \longrightarrow \mathcal{C}$ is the colour function used to structure places and transitions (of the underlying P/T-net)

- $Pre, Post : TRANS \longrightarrow \mu PLACE$ are the pre and post mappings with

$$TRANS = \{(t, m) \mid m \in C(t), t \in T\}$$

$$PLACE = \{(s, g) \mid g \in C(s), s \in S\}$$

- $M_0 \in \mu PLACE$ is a multiset known as the initial marking

4.4.2 Marking

A **Marking** is a multiset, $M \in \mu PLACE$.

4.4.3 Enabling

A finite multiset of transition modes, $T_\mu \in \mu TRANS$, is *enabled* at a marking M *iff*

$$Pre'(T_\mu) \leq M$$

Thus a multiset of transition modes is enabled if there are enough tokens on the input places to satisfy the linear combination of the pre maps for each transition mode in T_μ.

4.4.4 Transition Rule

Given that a multiset of transition modes, T_μ, is enabled at a marking M, then a *step* may occur resulting in a new marking M' given by

$$M' = M - Pre'(T_\mu) + Post'(T_\mu).$$

This is often denoted by $M[T_\mu\rangle M'$ or sometimes, for a single transition mode, $tr \in TRANS$, by $M \xrightarrow{tr} M'$.

4.4.5 Set of Reachable Markings

The set of reachable markings, $[M_0\rangle$, of CP is obtained inductively as follows.

- $M_0 \in [M_0\rangle$; and

- if $M_1 \in [M_0\rangle$ and for some $tr \in TRANS$, $M_1[tr\rangle M_2$, then $M_2 \in [M_0\rangle$.

4.5 Relationship to Jensen's CP-nets

CP-nets as defined above are very closely related to the way Jensen defines his CP-Matrix [87]. The main differences are:

1. The *structuring* set of colour sets, C, is included in the structure CP.

2. The empty net $(S \cup T = \emptyset)$ and isolated elements are allowed.

3. The *Pre* and *Post* mappings are defined in general for the whole net, rather than a family of functions, one for each place-transition pair $((s,t) \in S \times T)$.

The structuring set is included for completeness and as yet I have not found it useful to restrict the definition to exclude isolated elements or the empty net. This is in keeping with [150].

We shall now relate the pre and post mappings to Jensen's *positive* and *negative* *incidence* functions, $I_-(s,t)$ and $I_+(s,t)$.

We define the following functions

$$Pre(s,t), Post(s,t) : C(t) \longrightarrow \mu C(s)$$

and

$$Pre'(s,t), Post'(s,t) : \mu C(t) \longrightarrow \mu C(s)$$

so that for all $m \in C(t), t \in T$ and for all $g \in C(s), s \in S$ (and using $Pre(s,t;m)$ for $Pre(s,t)(m)$ and similarly for $Post(s,t)(m)$)

- $mult(g, Pre(s,t;m)) = mult((s,g), Pre(t,m))$ and

- $mult(g, Post(s,t;m)) = mult((s,g), Post(t,m))$

and similarly for the multiset extensions which are identical to Jensen's *positive* and *negative incidence* functions.

- $Pre'(s,t) = I_-(s,t)$

- $Post'(s,t) = I_+(s,t)$

As the multiset extension of a function includes the function itself, it is only necessary to use the multiset extension function. Jensen adopts this approach. I prefer to retain the original function and explicitly use it when the multiset extension is not required. This is the case when we wish to define the pre and post maps for example. It is hoped that this adds to the clarity of the presentation.
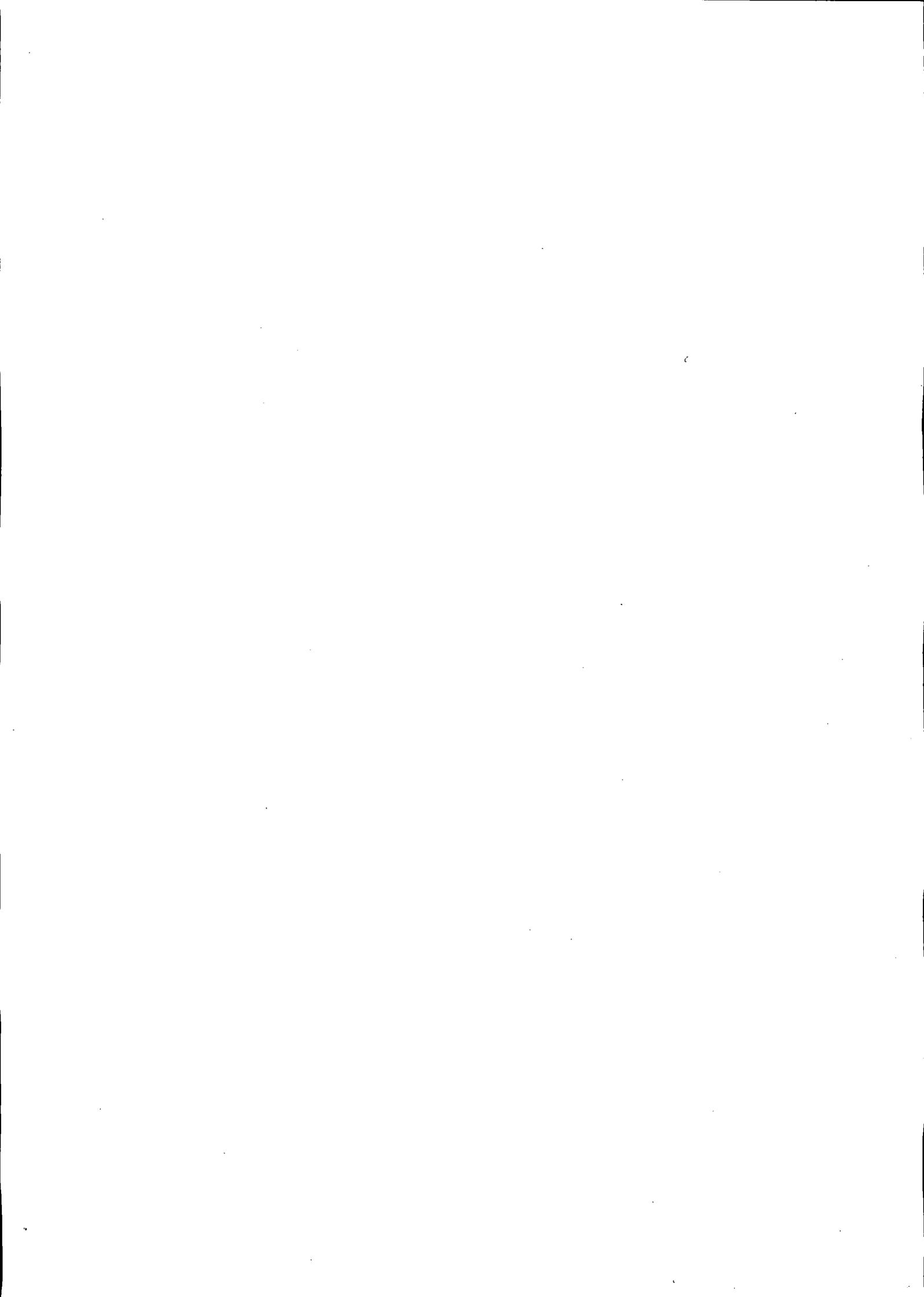
Chapter 5

Extensions to CP-nets

5.1   Place Capacity

In the definition of CP-nets it is assumed that all places have infinite capacity. We can generalise the notion of place capacity for P/T-systems [20] (and PrT-nets [71]) quite easily. We denote the capacity by K, representing a multiset of tokens for each place

$$K \in \mu_{\infty}^{+} PLACE.$$

This capacity cannot be exceeded by the marking: $M \leq K$. Specifically, the initial Marking, $M_0 \in \mu PLACE$ satisfies $M_0 \leq K$. Note that the capacity may contain elements with infinite multiplicities but zero multiplicities are not allowed (see Appendix A for the definition of $\mu_{\infty}^{+} A$).

To conform with the usual definition of P/T-systems [20] and for the reasons discussed in [58], the enabling condition now becomes

$$M[T_{\mu}\rangle M' \ \ iff \ \ Pre'(T_{\mu}) \leq M \leq K - Post'(T_{\mu})$$

and the transition rule is unchanged. The subtraction used above is vector subtraction and '$\leq$' is vector comparison (see Appendix A).

We may now define capacity CP-nets as follows:

Definition

A **CP$_{\mathbf{K}}$-net**, is a structure

$$CP_K = (CP, K)$$

where

- CP is a CP-net as defined in section 4.4.1 with initial marking restricted to comply with the capacity multiset: $M_0 \leq K$, and

- $K \in \mu_{\infty}^{+} PLACE$ is a multiset known as the place capacity.

Because of the enabling condition and transition rule, any marking $M \in [M_0\rangle$, will comply with the capacity constraint: $M \leq K$.

46

We can consider a partition of K according to places in the same way as we did for markings. We define for $s \in S$ and for all $g \in C(s)$,

$$K_s \in \mu_\infty^+(\{s\} \times C(s))$$

such that $mult((s,g), K_s) = mult((s,g), K)$ and $\sum_{s \in S} K_s = K$.

It will be useful to consider the capacity of a particular place, s, by defining

$$K(s) \in \mu_\infty^+ C(s)$$

as the multiset of tokens such that for each $g \in C(s)$

$$mult(g, K(s)) = mult((s,g), K).$$

5.1.1 Inhibitor Maps

Zero-Testing Inhibitor

There may be times when we would like to have the ability to test places for a null marking. This corresponds to the well known *inhibitor arc* extension to Petri nets which increases their modelling power to that of a Turing machine [115].

We shall denote the power set of a set A by $\mathcal{P}(A)$. We can generalise the notion of an inhibitor arc for high-level nets, by introducing a function

$$I_0 : TRANS \longrightarrow \mathcal{P}(PLACE)$$

which associates with each transition mode a subset of places that will be used for zero testing.

To obtain a suitable inhibitor condition for a multiset of transition modes we define the following function:

$$I_0' : \mu TRANS \longrightarrow \mathcal{P}(PLACE)$$

where for $tr \in TRANS$ and $T1_\mu, T2_\mu \in \mu TRANS$ we have

- $I_0'(\emptyset) = \emptyset$

- $I_0'(tr) = I_0(tr)$

- $I_0'(T1_\mu + T2_\mu) = I_0'(T1_\mu) \cup I_0'(T2_\mu)$

Thus for example for $n, m \in N^+$ and $tr, tr1, tr2 \in TRANS$

- $I_0'(ntr) = I_0'(tr) = I_0(tr)$

- $I_0'(ntr1 + mtr2) = I_0'(tr1) \cup I_0'(tr2) = I_0(tr1) \cup I_0(tr2)$

The enabling condition is then formulated as the conjunction of two predicates: the CP-net enabling predicate (section 4.4.3) and the inhibitor predicate, which is given by $M \cap I_0'(T_\mu) = \emptyset$. Here, $I_0'(T_\mu)$ is considered to be special multiset over $PLACE$, with multiplicities being either zero or one. Multiset intersection, $\cap$, is defined in Appendix A.

In summary, a finite multiset of transition modes, $T_\mu \in \mu TRANS$, is enabled by a marking M *iff*

$$(Pre'(T_\mu) \leq M) \wedge (M \cap I_0'(T_\mu) = \emptyset)$$

Thus a multiset of transition modes is enabled if there are enough tokens on the input places to satisfy the pre map, and no tokens reside on the inhibitor places.

The transition rule remains unchanged.

Threshold Inhibitor

For modelling convenience, we would like to introduce a *threshold* inhibitor condition, which instead of requiring that certain tokens must be absent from the marking as above, requires that certain tokens must not exceed a preset multiplicity, known as the *threshold*. We do this by generalising the above inhibitor function to associate a general multiset of (underlying) places (the thresholds) with each transition mode.

$$I : TRANS \longrightarrow \mu_\infty PLACE$$

Thus the multiplicity of place colours not having a threshold will be infinite, and the zero-testing inhibitor is simulated when the multiplicities are zero.

We can again extend this function to multisets of transition modes by defining

$$I' : \mu TRANS \longrightarrow \mu_\infty PLACE$$

where for $tr \in TRANS$, and $T1_\mu, T2_\mu \in \mu TRANS$ we have

- $I'(\emptyset) = \{(p, \infty) \mid p \in PLACE\}$
- $I'(tr) = I(tr)$
- $I'(T1_\mu + T2_\mu) = I'(T1_\mu) \cap I'(T2_\mu)$

The first item ensures that $I'(T1_\mu + \emptyset) = I'(T1_\mu)$.

Note that this extension to multisets is different from that for the pre map. Here the extension yields the minimum threshold, rather than the sum of thresholds.

As above, the enabling condition is then formulated as the conjunction of two predicates. A finite multiset of transition modes, $T_\mu \in \mu TRANS$, is enabled by a marking M *iff*

$$Pre'(T_\mu) \leq M \leq I'(T_\mu)$$

Thus a multiset of transition modes is enabled if there are enough tokens on the input places to satisfy the pre map and the thresholds are not exceeded for the inhibitor places. Again the transition rule remains unchanged.

Definition

A Threshold Inhibitor CP-net, **CP$_I$-net**, is a structure

$$CP_I = (CP, I)$$

where

- CP is a CP-net as defined in section 4.4.1; and

- $I : TRANS \longrightarrow \mu_\infty PLACE$ is the threshold inhibitor map.

The enabling condition has been defined above, and markings and the transition rule are as defined for CP-nets.

For the graphical form it will be convenient to define an inhibitor map for each place-transition pair as follows: for $s \in S$ and $t \in T$,

$$I(s,t) : C(t) \longrightarrow \mu C(s)$$

where for all $m \in C(t), t \in T$ and for all $g \in C(s), s \in S$

$$mult(g, I(s,t;m)) = mult((s,g), I(t,m))$$

5.1.2 P-nets

We are now in a position to define **P-nets**. P-nets are CP-nets extended by place capacities and the threshold inhibitor map. The motivation for choosing the threshold inhibitor is that it is symmetrical with the pre map and thus allows the two to be equated. This is convenient because it allows a marking to be purged by a single transition occurrence, useful when specifying the abortion of various procedures. Chapter 8 develops the theory and provides some examples.

As suggested in [90] we could use CP$_{KI}$-nets (Capacity/Inhibitor CP-nets) as the name to avoid a proliferation of names of high-level nets. After considering a number of options it was felt that the advantage of brevity; that P-net can be considered an abbreviation for CP$_{KI}$-net; and the mild link back to the application domain of protocols (P-nets could be thought of as an abbreviation of *Protocol Nets*) where some of the ideas for the extensions arose; were sufficient reasons to retain the name P-nets. One may also consider CP$_{KI}$-nets to be a broader class where the inhibitor extension may be defined differently (e.g. the zero-testing inhibitor), whereas with P-nets the inhibitor extension is the threshold inhibitor as defined above.

Definition

A **P-net**, is a structure

$$P = (CP, I, K)$$

where

- CP is a CP-net as defined in section 4.4.1 with initial marking restricted to comply with the capacity multiset: $M_0 \leq K$.

- $I : TRANS \longrightarrow \mu_\infty PLACE$ is the threshold inhibitor map; and

- $K \in \mu_\infty^+ PLACE$ is a multiset known as the place capacity.

Marking

A Marking is a multiset, $M \in \mu PLACE$, such that $M \leq K$.

Enabling

A finite multiset of transition modes, $T_\mu \in \mu TRANS$, is *enabled* at a marking M *iff*

$$(Pre'(T_\mu) \leq M \leq K - Post'(T_\mu)) \wedge (M \leq I'(T_\mu))$$

Thus a multiset of transition modes is enabled if there are enough tokens on the input places to satisfy the pre map, there is enough capacity left in the output places, and the inhibitor thresholds are not exceeded.

Transition Rule

The transition rule is the same as for CP-nets and is given in section 4.4.4.

$$M' = M - Pre'(T_\mu) + Post'(T_\mu)$$

Reachable Markings

This is again defined in exactly the same way as for CP-nets (see section 4.4.5).

Special Cases

There are three obvious special cases of P-nets.

1. Capacity CP-nets (CP_K-nets)

   When the thresholds are infinite, (that is for all $tr \in TRANS$ and for all $p \in PLACE$, $mult(p, I(tr)) = \infty$), the P-net becomes a capacity CP-net.

2. Inhibitor CP-nets (CP_I-nets)

   When the capacities of the places are infinite (for all $p \in PLACE$, $K(p) = \infty$), the P-net becomes an inhibitor CP-net.

3. CP-nets

   When the capacities of the places and the thresholds are infinite (that is for all $p \in PLACE$, $K(p) = \infty$ and for all $tr \in TRANS$, for all $p \in PLACE$, $mult(p, I(tr)) = \infty$) the P-net reduces to a CP-net.

The condition on the inhibitor function given in case 1 above is sufficient but not necessary for a P-net to be a CP_K-net. The conditions given in cases 2 and 3 are necessary and sufficient. In case 2, its obvious that the condition is sufficient. It is also necessary that $\forall p \in PLACE, K(p) = \infty$ to remove the initial marking capacity restriction $(M_0 \leq K)$. Once the necessary and sufficient conditions for case 1 have been determined, it becomes obvious that those stated above for case 3 are also necessary and sufficient.

We now explore the necessary and sufficient conditions for a P-net to be a CP_K-net.

Consider the following proposition.

Proposition 5.1 *If $\forall tr \in TRANS, I(tr) \geq K - Post(tr)$, then for any finite multiset, $T_\mu \in \mu TRANS$,*

$$I'(T_\mu) \geq K - Post'(T_\mu)$$

Proof:

Let $T_\mu = \sum_{i=1}^n t_i$ where n is a positive integer and for all $i \in In = \{1, \ldots, n\}, t_i \in TRANS$. (Note that the t_i need not be distinct.)

From the definition of I' (see section 5.1.1),

$$I'(T_\mu) = \bigcap_i I(t_i) \tag{5.1}$$

and from the definition of $Post'$ (see section 4.3),

$$Post'(T_\mu) = \sum_i Post(t_i). \tag{5.2}$$

Now, for any $p \in PLACE$, (and from the definition of multiset intersection - see Appendix A)

$$
\begin{aligned}
mult(p, \bigcap_i I(t_i)) &= min(mult(p, I(t_1)), \ldots, mult(p, I(t_n))) \\
&= mult(p, I(t_j)) \tag{5.3}
\end{aligned}
$$

where for all $k \in In, mult(p, I(t_j)) \leq mult(p, I(t_k))$.

From the proposition, we have for all $t_i \in T_\mu$

$$I(t_i) \geq K - Post(t_i)$$

and hence

$$
\begin{aligned}
mult(p, I(t_j)) &\geq mult(p, K - Post(t_j)) \\
&\geq mult(p, K - Post(t_j) - \sum_{i \neq j} Post(t_i)) \\
&\geq mult(p, K - \sum_i Post(t_i)) \\
&\geq mult(p, K - Post'(T_\mu)) \tag{5.4}
\end{aligned}
$$

Combining equations 5.1, 5.3 and 5.4

$$mult(p, I'(T_\mu)) \geq mult(p, K - Post'(T_\mu)) \qquad (5.5)$$

Since equation 5.5 is true for all $p \in PLACE$, we obtain the desired result:

$$I'(T_\mu) \geq K - Post'(T_\mu) \qquad (5.6)$$

$\square$

A corollary from proposition 5.1 is

Corollary 5.1 *If* $\forall tr \in TRANS, I(tr) \geq K - Post(tr)$, *then for any finite multiset,* $T_\mu \in \mu TRANS$,

$$M \leq K - Post'(T_\mu) \Rightarrow M \leq I'(T_\mu)$$

This corollary implies that if $\forall tr \in TRANS, I(tr) \geq K - Post(tr)$, holds for a P-net, then the threshold inhibitor has no effect on the enabling condition and is therefore redundant. The P-net is then a capacity CP-net.

The converse is also true.

Proposition 5.2 *If a P-net is a* CP$_K$*-net, then* $\forall tr \in TRANS, I(tr) \geq K - Post(tr)$

Proof:

Since the only effect of the inhibitor on the behaviour of the P-net concerns the enabling condition, the P-net enabling condition must reduce to that of a CP$_K$-net. This happens when for all finite $T_\mu \in \mu TRANS$

$$I'(T_\mu) \geq K - Post'(T_\mu)$$

which implies as a special case that $\forall tr \in TRANS, I(tr) \geq K - Post(tr)$ $\square$

We can now state the following theorem concerning the relationship between P-nets and CP$_K$-nets.

Theorem 5.1 *A P-net is a* CP$_K$*-net iff* $\forall tr \in TRANS, I(tr) \geq K - Post(tr)$

Proof: Follows directly from proposition 5.2 and corollary 5.1. $\square$

Given that a necessary and sufficient condition to remove the capacity condition is that for all $p \in PLACE, K(p) = \infty$, and that for all $tr \in TRANS$, for all $p \in PLACE, mult(p, Post(tr)) < \infty$, then the only way to remove the inhibitor (i.e. to satisfy theorem 5.1) is for the thresholds to be infinite everywhere. Hence the conditions stated in case 3 above for a P-net to be a CP-net are both necessary and sufficient.
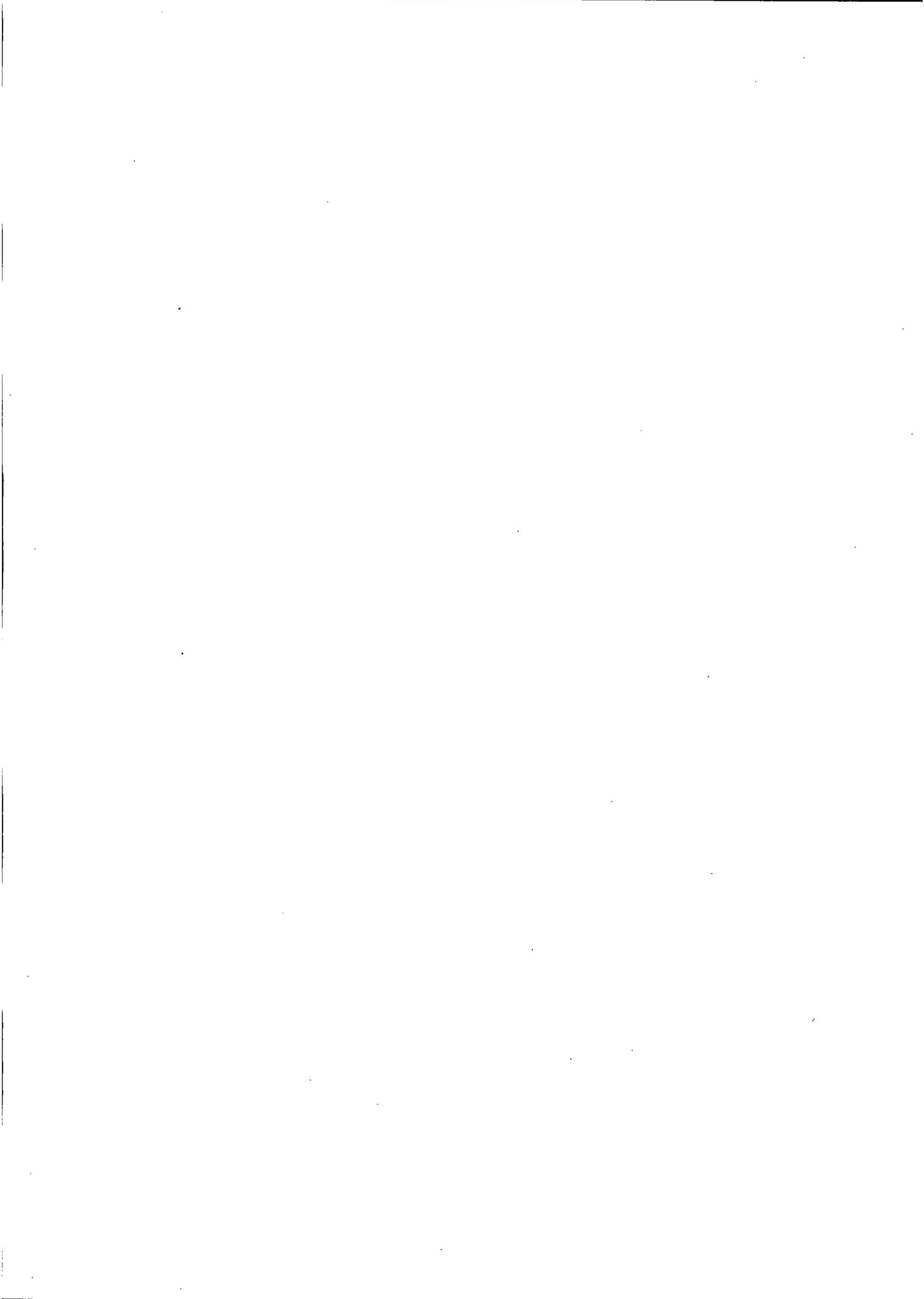
Chapter 6

Transforming P-nets to CP-nets

To allow the analysis techniques that have been and are being developed for CP-nets to be applied to P-nets, it is important to be able to transform P-nets to CP-nets and to know precisely under which circumstances these transformations are applicable. This chapter[1] describes transformations under which the interleaving behaviours of the P-net and CP-net are equivalent in the sense that their single-step reachability graphs are isomorphic. Further, it establishes the conditions on the P-net where the transformations do preserve true concurrency.

To motivate the transformations, it is first necessary to establish an *extended complementary place invariant* for CP-nets.

6.1 Extended Complementary Place Invariant

In this section we consider a class of CP-nets in which the set of places is partitioned into two sets of the same cardinality such that for each place in one set there is a corresponding place in the other. We also relate the pre and post maps restricted to one set of places to the pre and post maps restricted to the other set, in such a way that an invariant exists between the markings of the two sets of places.

This development has been inspired by the idea of complementarity for P/T-systems [120]. We shall therefore call the corresponding set of places, *complementary* places. When complementing a P/T-system the pre map (post map) on the complementary places is set equal to the post map (pre map) on the original set of places. This guarantees an invariant on each pair of complementary places. If p is an original place and $\hat{p}$ its complement, then for every reachable marking M, $M(p) + M(\hat{p}) = K(p)$ where $K(p) = M_0(p) + M_0(\hat{p})$ is a constant, the capacity of place p.

The following generalises this idea in two ways. Firstly we relax the relationship between the pre and post maps (we are only concerned with the equality of the differences in the pre and post maps) and secondly we raise these notions to the level of CP-nets.

---

[1]The first three sections (6.1 to 6.3) of this chapter have recently been published [23].

$$
\begin{aligned}
&S1 = S \cup \widehat{S} \\
&\text{where } \widehat{S} = \{\widehat{s} \mid s \in S\} \\
&\forall s \in S, C(\widehat{s}) = C(s) \\
&PLACE1 = PLACE \cup \widehat{PLACE} \\
&PLACE = \{(s,g) \mid g \in C(s), s \in S\} \\
&\widehat{PLACE} = \{(\widehat{s},g) \mid g \in C(s), s \in S\} \\
&\text{The pre and post maps are factored: } \forall tr \in TRANS \\
&Pre1(tr) = Pre(tr) + \widehat{Pre}(tr) \\
&Post1(tr) = Post(tr) + \widehat{Post}(tr) \\
&Pre, Post : TRANS \longrightarrow \mu PLACE \\
&\widehat{Pre}, \widehat{Post} : TRANS \longrightarrow \mu \widehat{PLACE} \\
&M1_0 = M_0 + \widehat{M}_0 \\
&\text{where } M_0 \in \mu PLACE \text{ and } \widehat{M}_0 \in \mu \widehat{PLACE}
\end{aligned}
$$

Figure 6.1: **CP1**: Basic Definition

6.1.1 Definitions

Let $\mathbf{CP1} = (S1, T, \mathcal{C}; C, Pre1, Post1, M1_0)$ be a coloured net with components defined in figure 6.1. The hat notation is used to indicate complementary places, or sets or functions associated with complementary places. An overbar notation is defined in figure 6.2. It is used to complement the marking or pre and post maps associated with the original places.

$$
\begin{aligned}
&\text{If } X \in \mu PLACE \text{ then } \overline{X} \in \mu \widehat{PLACE} \text{ such that} \\
&\forall p \in PLACE, \ \forall \widehat{p} \in \widehat{PLACE}, \ \overline{X}(\widehat{p}) = X(p)
\end{aligned}
$$

Figure 6.2: Overbar Notation

We now state the restriction on the pre and post maps which guarantees that the complementary place invariant holds.

$$\forall tr \in TRANS, \ \overline{Pre(tr)} - \overline{Post(tr)} = \widehat{Post}(tr) - \widehat{Pre}(tr) \tag{6.1}$$

6.1.2 Complementary Place Invariant

Let $M1 \in \mu PLACE1$, a reachable marking of **CP1**, be factored so that $M1 = M + \widehat{M}$ where $M \in \mu PLACE$ and $\widehat{M} \in \mu \widehat{PLACE}$. For convenience, let $\overline{M_0} + \widehat{M}_0 = \overline{K}$ where $\overline{K} \in \mu \widehat{PLACE}$, then the invariant is given in the following proposition.

Proposition 6.1 *For* **CP1** *above, satisfying equation 6.1,*

$$\forall M1 \in [M1_0), \overline{M} + \widehat{M} = \overline{K}$$

54

Proof:

The proof is by induction over the reachable markings. The proposition is true by definition for the initial marking. Given any reachable marking $M1 = M + \widehat{M}$, we assume that

$$\overline{M} + \widehat{M} = \overline{K} \tag{6.2}$$

and then prove it is true for any follower marking.

$\forall tr \in TRANS$ such that $Pre1(tr) \leq M1$, the follower marking, $M1'$, is given by the transition rule:

$$
\begin{aligned}
M1' &= M1 + Pre1(tr) - Post1(tr) \\
&= M + \widehat{M} + Pre(tr) + \widehat{Pre}(tr) - Post(tr) - \widehat{Post}(tr) \\
&= M + Pre(tr) - Post(tr) + \widehat{M} + \widehat{Pre}(tr) - \widehat{Post}(tr) \\
&= M' + \widehat{M}'
\end{aligned}
$$

where

$$
\begin{aligned}
M' &= M + Pre(tr) - Post(tr) \tag{6.3} \\
\widehat{M}' &= \widehat{M} + \widehat{Pre}(tr) - \widehat{Post}(tr) \tag{6.4}
\end{aligned}
$$

Thus we need to prove that given equation 6.2

$$\overline{M'} + \widehat{M}' = \overline{K} \tag{6.5}$$

Substituting for M' and $\widehat{M}'$ using equations 6.3 and 6.4, rearranging and using equation 6.1 gives the required result.

$$
\begin{aligned}
\overline{M'} + \widehat{M}' &= \overline{M + \overline{Pre(tr)} - \overline{Post(tr)}} + \widehat{M} + \widehat{Pre}(tr) - \widehat{Post}(tr) \\
&= \overline{M} + \widehat{M} + \overline{Pre(tr)} - \overline{Post(tr)} + \widehat{Pre}(tr) - \widehat{Post}(tr) \\
&= \overline{M} + \widehat{M} \\
&= \overline{K}
\end{aligned}
$$

$\square$

6.2 Interleaving Equivalence of P-nets and CP-nets

In this section we give transformations from P-nets to CP-nets which preserve their interleaving behaviour and show that there is an isomorphism between the single step (interleaving) reachability trees of P-nets and CP-nets, under weak assumptions.

The transformations are important as they allow the theory developed for the analysis of CP-nets (high-level reachability trees and invariants analysis [87]) to be applied to P-nets in most practical situations.

We firstly consider the most straightforward case where the capacities of all places in the P-net are finite and then relax this condition when there is no inhibitor condition associated with an underlying place (see section 6.2.3).

6.2.1 Complete Complementation Transformation

A P-net, $P = (S, T, C; C, Pre, Post, M_0, I, K)$, with the restriction that

$$\forall tr \in TRANS, \forall p \in PLACE, mult(p, I(tr)) < \infty \Rightarrow mult(p, K) < \infty \quad (6.6)$$

can be transformed into CP-net $CP = (\underline{S}, \underline{T}, \underline{C}; \underline{C}, \underline{Pre}, \underline{Post}, \underline{M_0})$, where we have used the underline to denote the CP-net elements.

Firstly we define a combined inhibitor-capacity function. For a P-net, the enabling condition for $tr \in TRANS$, at marking M, comprises

1. the *precondition* $Pre(tr) \leq M$

2. the *capacity condition* $M \leq K - Post(tr)$ and

3. the *inhibitor condition* $M \leq I(tr)$

It will be convenient to combine enabling conditions 2 and 3 above and we therefore define the following inhibitor-capacity function:

$$IK : TRANS \longrightarrow \mu_\infty PLACE$$

where $\forall tr \in TRANS$, $IK(tr) = (K - Post(tr)) \cap I(tr)$.

The enabling condition becomes

1. $Pre(tr) \leq M$

2. $M \leq IK(tr)$

We shall replace the second enabling condition by an equivalent pre map on a set of complementary places in the CP-net. The construction is as follows.

Firstly we impose the restriction that all places have finite capacity

$$\forall p \in PLACE, K(p) < \infty$$

For each $s \in S$, we create a complementary place $\hat{s}$ and gather them together into a set of complementary places, $\widehat{S} = \{\hat{s} \mid s \in S\}$. The set of places of the CP-net is then $\underline{S} = S \cup \widehat{S}$.

We denote the set of underlying complementary places by $\widehat{PLACE}$, given by

$$\widehat{PLACE} = \{(\hat{s}, g) \mid g \in C(s), s \in S\}$$

and use the notation defined in figure 6.2 for complementing multisets associated with P-net places.

If $\widehat{M}$ denotes the marking of the complementary places $\widehat{S}$, then $\widehat{M} \in \mu\widehat{PLACE}$. The initial marking and pre and post maps will be chosen so that the following invariant holds

$$\overline{M} + \widehat{M} = \overline{K} \quad (6.7)$$

$$\underline{S} = S \cup \widehat{S}$$
$$\underline{T} = T$$
$$\underline{C} = C$$
$$\underline{C} : S \cup \widehat{S} \cup T \longrightarrow C$$
where $\forall s \in S, \forall t \in T, \underline{C}(s) = C(s), \underline{C}(\widehat{s}) = C(s), \underline{C}(t) = C(t)$
The pre and post maps: $\forall tr \in TRANS$,
$$\underline{Pre}(tr) = Pre(tr) + \overline{K} - \overline{IK(tr)}$$
$$\underline{Post}(tr) = Post(tr) + \overline{Pre(tr)} - \overline{Post(tr)} + \overline{K} - \overline{IK(tr)}$$
$$\underline{M_0} = M_0 + \overline{K} - \overline{M_0}$$

Figure 6.3: P-net to CP-net Transformation

The _Pre_ and _Post_ maps are factored with respect to the original set of places and the set of complementary places and thus $\forall tr \in TRANS$

$$\underline{Pre}(tr) \quad = \quad Pre(tr) + \widehat{Pre}(tr) \tag{6.8}$$
$$\underline{Post}(tr) \quad = \quad Post(tr) + \widehat{Post}(tr) \tag{6.9}$$

where $\widehat{Pre}, \widehat{Post} : TRANS \longrightarrow \mu PL\widehat{AC}E$. From proposition 6.1, the above invariant holds if

$$\forall tr \in TRANS, \overline{Pre(tr)} - \overline{Post(tr)} = \widehat{Post}(tr) - \widehat{Pre}(tr) \tag{6.10}$$

If we complement the second enabling condition and substitute for $\overline{M}$ using equation 6.7, we obtain the equivalent precondition on the complementary places. Thus $\forall tr \in TRANS$

$$M \leq IK(tr) \quad \Leftrightarrow \quad \overline{M \leq IK(tr)}$$
$$\Leftrightarrow \quad \overline{K} - \widehat{M} \leq \overline{IK(tr)}$$
$$\Leftrightarrow \quad \overline{K} - \overline{IK(tr)} \leq \widehat{M} \tag{6.11}$$

Thus $\forall tr \in TRANS, \widehat{Pre}(tr) = \overline{K} - \overline{IK(tr)}$ which then gives us $\underline{Pre}(tr)$ from equation 6.8. Finally, knowing $\widehat{Pre}(tr)$, the post map is derived from equations 6.10 and 6.9.

The transformation is summarized in figure 6.3.

6.2.2 Proof of Interleaving Equivalence

The interleaving behaviours of the P-net and corresponding CP-net defined in figure 6.3 are equivalent in the sense that their reachability trees (transition systems) are isomorphic. This may be stated more precisely in the following theorem.

Theorem 6.1 (Interleaving Equivalence)   *1. For each reachable marking, $M \in [M_0\rangle$ of P, there is a corresponding reachable marking $\underline{M} \in [\underline{M_0}\rangle$ of CP and vice versa. That is there is a bijection:*

$$\rho : [M_0\rangle \longrightarrow [\underline{M_0}\rangle$$

where $\underline{M} = \rho(M) = M + \overline{K} - \overline{M}$; and

2. The single step occurrences of transition modes in P and CP are in one-to-one correspondence:

$$M \overset{tr}{\longrightarrow} M' \text{ iff } \rho(M) \overset{tr}{\longrightarrow} \rho(M')$$

Proof:

The proof is by induction over the reachable markings. Point 1 is true for the initial marking by definition: $\underline{M_0} = M_0 + \overline{K} - \overline{M_0}$ (see figure 6.3).

Assume

$$\underline{M} = M + \overline{K} - \overline{M} \tag{6.12}$$

We firstly need to prove that if a transition mode, tr, is enabled at M (in P), then it is also enabled at $\underline{M}$ (in CP) and vice versa. This is formally stated in the following lemma.

Lemma 6.1 (Enabling Lemma)

$$Pre(tr) \leq M \leq IK(tr) \text{ iff } \underline{Pre}(tr) \leq \underline{M}$$

Proof:

Starting with the CP-net, using equation 6.12 and substituting for the definition of $\underline{Pre}(tr)$ reveals

$$
\begin{array}{rl}
& \underline{Pre}(tr) \;\leq\; \underline{M} \\
\Leftrightarrow & \underline{Pre}(tr) \;\leq\; M + \overline{K} - \overline{M} \\
\Leftrightarrow & (Pre(tr) + \overline{K} - \overline{IK(tr)}) \;\leq\; (M + \overline{K} - \overline{M}) \\
\Leftrightarrow & Pre(tr) \leq M \text{ and } \overline{M} \leq \overline{IK(tr)} \\
\Leftrightarrow & Pre(tr) \leq M \leq IK(tr)
\end{array}
$$

which has proved the enabling lemma. $\square$

We now prove 1 and 2 together in two parts. Firstly we prove that ρ is an injection and the implication on the transition systems.

The enabling lemma tells us that if tr of P is enabled at M, then tr of CP is enabled at $\underline{M} = M + \overline{K} - \overline{M}$. Consider the successor markings

- $M \overset{tr}{\longrightarrow} M'$; and

- $\underline{M} \overset{tr}{\longrightarrow} \underline{M'}$

Assuming equation 6.12, we wish to prove that $\forall tr \in TRANS$

$$M \xrightarrow{tr} M' \Rightarrow M + \overline{K} - \overline{M} \xrightarrow{tr} M' + \overline{K} - \overline{M'}$$

From the transition rule and the definitions of the pre and post maps for the CP-net, we have

$$
\begin{aligned}
\underline{M'} &= M + \overline{K} - \overline{M} - \underline{Pre}(tr) + \underline{Post}(tr) \\
&= M + \overline{K} - \overline{M} - Pre(tr) + Post(tr) + \overline{Pre(tr)} - \overline{Post(tr)} \\
&= M - Pre(tr) + Post(tr) + \overline{K} - \overline{M} + \overline{Pre(tr)} - \overline{Post(tr)} \\
&= M' + \overline{K} - \overline{M'}
\end{aligned}
$$

Hence we have proved the one-way implication and also that ρ is an injection. We now prove the reverse implication of 2 and that ρ is surjective. The proof has exactly the same form as the previous proof.

Let R be the marking we get when a transition occurs in the P-net for mode tr at marking M. We need to prove $\forall tr \in TRANS$

$$M + \overline{K} - \overline{M} \xrightarrow{tr} M' + \overline{K} - \overline{M'} \Rightarrow M \xrightarrow{tr} M'$$

and hence that $R = M'$.

From the transition rule, the definitions of the pre and post maps for the CP-net and equation 6.12, we have

$$
\begin{aligned}
R &= M - Pre(tr) + Post(tr) \\
&= \underline{M} - \overline{K} + \overline{M} - Pre(tr) + Post(tr) \\
&= \underline{M} - \underline{Pre}(tr) + \underline{Post}(tr) - \overline{K} + \overline{M} - \overline{Pre(tr)} + \overline{Post(tr)} \\
&= \underline{M'} - \overline{K} + \overline{M'} \\
&= M'
\end{aligned}
$$

Thus for each successor marking $\underline{M'} = M' + \overline{K} - \overline{M'}$ in CP we have a corresponding marking M' in P, which completes the proof. $\square$

6.2.3   Less Restrictive Transformation

We now remove the restriction that all places must have finite capacity. If for some $p \in PLACE$, $mult(p, K) = \infty$, then we must have $\forall tr \in TRANS$, $mult(p, I(tr)) = \infty$, to satisfy the initial restriction (equation 6.6). In this case only an identity transformation is required.

As before we shall create complementary places in the underlying P/T-net to eliminate finite capacities and the inhibitor condition. The definition of $\widehat{PLACE}$ is modified to exclude an underlying place corresponding to p when $K(p) = \infty$.

$$\widehat{PLACE}' = \{\hat{p} \mid p \in PLACE \wedge K(p) \neq \infty\}$$

The set of complementary places now becomes

$$\widehat{S}' = \{\hat{s} \mid \exists(\hat{s}, g) \in \widehat{PLACE}', s \in S, g \in C(s)\}$$

$$\underline{S} = S \cup \widehat{S}'$$

where $\widehat{S}' = \{\hat{s} \mid \exists(\hat{s}, g) \in \widehat{PLACE}', s \in S, g \in C(s)\}$

and $\widehat{PLACE}' = \{\hat{p} \mid p \in PLACE \wedge K(p) \neq \infty\}$

$\underline{T} = T$

$\underline{C} = C \cup \widehat{C}$

where $\widehat{C} = \{\underline{C}(\hat{s}) \mid \hat{s} \in \widehat{S}'\}$

$\underline{C} : S \cup \widehat{S}' \cup T \longrightarrow \underline{C}$

where $\forall s \in S, \forall t \in T,$

$\underline{C}(s) = C(s)$

$\underline{C}(\hat{s}) = \{g \mid (\hat{s}, g) \in \widehat{PLACE}'\}$

$\underline{C}(t) = C(t)$

The pre and post maps: $\forall tr \in TRANS$

$\underline{Pre}(tr) = Pre(tr) + \overline{K} - \overline{IK(tr)}$

$\underline{Post}(tr) = Post(tr) + \overline{Pre(tr)} - \overline{Post(tr)} + \overline{K} - \overline{IK(tr)}$

$\underline{M_0} = M_0 + \overline{K} - \overline{M_0}$

Figure 6.4: Less Restrictive P-net to CP-net Transformation

where $\hat{p} = (\hat{s}, g)$. The corresponding colour sets are $\forall s \in S, \underline{C}(s) = C(s)$ and

$$\forall \hat{s} \in \widehat{S}', \underline{C}(\hat{s}) = \{g \mid (\hat{s}, g) \in \widehat{PLACE}'\}$$

so that the set of colour sets is $\underline{C} = C \cup \widehat{C}$ where $\widehat{C} = \{\underline{C}(\hat{s}) \mid \hat{s} \in \widehat{S}'\}$.

The overbar notation is changed accordingly so that if $X \in \mu PLACE$ then $\overline{X} \in \mu\widehat{PLACE}'$ such that for $p \in PLACE$, $\forall \hat{p} \in \widehat{PLACE}', \overline{X}(\hat{p}) = X(p)$. Note that when $K(p) = \infty$, there is no corresponding element in $\overline{X}$ (i.e. $\overline{X}(\hat{p}) = 0$).

The desired transformation is given in figure 6.4.

Because the transformation is of the same form as before the proofs carry through to the new transformation. Some care is needed with the enabling lemma in the implication proof when 'unbarring' where we need to note that for $p \in PLACE$ and $\forall tr \in TRANS$

$$mult(p, IK(tr)) = \begin{cases} mult(\hat{p}, \overline{IK(tr)}) & \text{if } \hat{p} \in \widehat{PLACE}' \\ \infty & \text{otherwise} \end{cases}$$

6.3 Example

The following example demonstrates that, in general, true concurrency is not preserved by the P-net to CP-net transformations. The example is essentially a simple P/T-system (we are using a instead of $\bullet$) with threshold inhibitor extension and is defined in figure 6.5. A graphical form is also given, following the usual conventions for P/T-systems. The threshold inhibitor is represented by the arc with a small circle at its end. The inscription defines the threshold for its associated place. For the initial marking shown it is obvious that

P-Net

$$S = \{s\}$$
$$T = \{t\}$$
$$C = \{\{a\}\}$$
$$C(s) = C(t) = \{a\}$$
$$Pre(t,a) = \{((s,a),1)\}$$
$$Post(t,a) = \emptyset$$
$$M_0 = \{((s,a),3)\}$$
$$I(t,a) = \{((s,a),4)\}$$
$$K = \{((s,a),5)\}$$
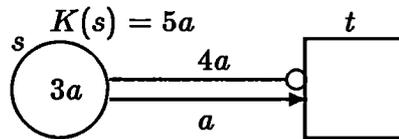
Graphical Form

Figure 6.5: Simple Inhibitor Example

CP-Net

$$S = \{s, \hat{s}\}$$
$$T = \{t\}$$
$$C = \{\{a\}\}$$
$$C(s) = C(\hat{s}) = C(t) = \{a\}$$
$$Pre(t,a) = \{((s,a),1),((\hat{s},a),1)\}$$
$$Post(t,a) = \{((s,a),0),((\hat{s},a),2)\}$$
$$M_0 = \{((s,a),3),((\hat{s},a),2)\}$$
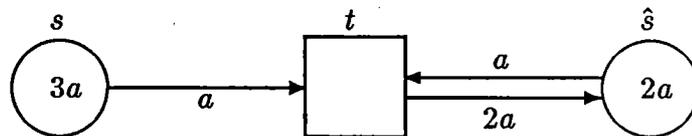
Graphical Form

Figure 6.6: CP-net equivalent for Simple Inhibitor Example

t is enabled, self concurrently 3 times. (More precisely, for $T_\mu = \{((t,a),3)\}$, $Pre'(T_\mu) = \{((s,a),3)\} = M_0$.)

The interleaving equivalent CP-net is shown in figure 6.6, together with a graphical representation. By inspection we can see that the interleaving behaviour of the two nets is equivalent, as they have the same reachability trees. (t occurs

three times, reducing the number of a tokens from 3 to zero in place s.) It is also obvious that truly concurrent behaviour is *not* preserved, since in the CP-net, t is only doubly concurrently enabled in the initial marking, compared with 3 times in the P-net.

6.4 Concurrency and P-net Transformations

The above example shows that the P-net to CP-net transformations do not guarantee the preservation of true concurrency. For P, a P-net, and its transformation, $\mathcal{T}(P)$, to a CP-net according to figure 6.4, and a finite multiset of transition modes, $T_\mu \in \mu TRANS$, then it is not necessarily true that if T_μ is enabled in P at a reachable marking M of P, then T_μ is enabled in $\mathcal{T}(P)$ at the corresponding marking, $\rho(M)$. More formally the proposition

$$\forall M \in [M_0\rangle, (Pre'(T_\mu) \leq M \leq (K - Post'(T_\mu)) \cap I'(T_\mu)) \Rightarrow (\underline{Pre}'(T_\mu) \leq \rho(M))$$

is false.

This leads to the investigation of the class of P-nets that can be transformed to CP-nets while preserving true concurrency. Firstly, we note that there is always a one-to-one correspondence between the reachable markings of the P-net, P, and those of $\mathcal{T}(P)$, as steps do not change $[M_0\rangle$, but only introduce extra arcs in the reachability tree. Thus we are only concerned about proving that the reachability trees have the same arcs if true concurrency is preserved. This is done by restricting P to ensure that if a multiset of transition modes is enabled in P at marking M, then the same multiset of transition modes is enabled in $\mathcal{T}(P)$ at marking $\rho(M)$ and vice versa.

A major class under which the transformation does preserve concurrency is CP_K-nets.

Theorem 6.2 *If a P-net is a CP_K-net, then the above transformations preserve true concurrency:*

$$\forall M \in [M_0\rangle, Pre'(T_\mu) \leq M \leq (K - Post'(T_\mu)) \Leftrightarrow \underline{Pre}'(T_\mu) \leq \rho(M)$$

Proof:

This theorem generalises the enabling lemma (lemma 6.1) to accommodate a finite multiset of transition modes (instead of a single mode), for the restricted case of a CP_K-net.

Let $T_\mu = \sum_{i=1}^n t_i$ where n is a positive integer and for all $i \in In = \{1, \ldots, n\}, t_i \in TRANS$. (Note that the t_i need not be distinct.)

From the definitions of Pre', $Post'$ and $\rho(M)$ (from theorem 6.1) and noting that for a CP_K-net $\forall t_i \in T_\mu, IK(t_i) = K - Post(t_i)$ (from theorem 5.1), then in a similar way to the proof of the enabling lemma we have for all $M \in [M_0\rangle$

$$\underline{Pre}'(T_\mu) \leq \rho(M)$$

$$\Leftrightarrow \quad \sum_i \underline{Pre}(t_i) \; \leq \; (M + \overline{K} - \overline{M})$$

$$\Leftrightarrow \quad \sum_i (Pre(t_i) + \overline{K} - \overline{IK(t_i)}) \; \leq \; (M + \overline{K} - \overline{M})$$

$$\Leftrightarrow \quad (\sum_i Pre(t_i) \leq M) \wedge (\overline{M} \leq (\overline{K} - \sum_i (\overline{K} - \overline{IK(t_i)})))$$

$$\Leftrightarrow \quad Pre'(T_\mu) \leq M \leq K - \sum_i (K - IK(t_i)) \qquad (6.13)$$

$$\Leftrightarrow \quad Pre'(T_\mu) \leq M \leq K - \sum_i Post(t_i)$$

$$\Leftrightarrow \quad Pre'(T_\mu) \leq M \leq K - Post'(T_\mu) \qquad (6.14)$$

$\square$

Now let us examine more carefully how the transformations can fail to preserve true concurrency. For a P-net, the enabling condition for a finite multiset of transitions, $T_\mu \in \mu TRANS$, at marking M, given in section 5.1.2, is equivalent to

$$Pre'(T_\mu) \leq M \leq (K - Post'(T_\mu)) \cap I'(T_\mu) \qquad (6.15)$$

The interleaving equivalent CP-net enabling condition at marking $\rho(M)$ has an equivalent condition on marking M given by equation 6.13. This allows a direct comparison of the two enabling conditions. The pre conditions are the same, but the *inhibitor-capacity conditions* (i.e. the '$M \leq expression$' part) are different in general. Of course, if we choose a multiset of transition modes (T_μ) where the inhibitor is not active $(\forall t_i \in T_\mu, K - Post(t_i) \leq I(t_i))$, then

$$\forall t_i \in T_\mu, IK(t_i) = K - Post(t_i)$$

and the enabling conditions (6.15 and 6.13 above) are the same, being identical to the CP_K-net enabling condition (6.14).

It is only transition modes where $\exists p \in PLACE : mult(p, I(t_i)) < mult(p, K - Post(t_i))$ that may cause a decrease in the amount of possible concurrency. It is therefore necessary to examine the inhibitor-capacity conditions component-wise as follows. For all $p \in PLACE$,

For the P-net

$$M(p) \leq min(mult(p, K - \sum_i Post(t_i)), mult(p, I(t_1)), \ldots, mult(p, I(t_n))) \qquad (6.16)$$

For the equivalent CP-net

$$M(p) \leq K(p) - \sum_{i=1}^{n} [K(p) - min(mult(p, K - Post(t_i)), mult(p, I(t_i)))] \qquad (6.17)$$

It is easy to verify that the above two conditions are equivalent when $n = 1$. Both 6.16 and 6.17 reduce to: for all $p \in PLACE$,

$$M(p) \leq min(mult(p, K - Post(t_1)), mult(p, I(t_1)))$$

They are also the same when no inhibitors are active, that is under the condition $\forall t_i \in T_\mu, p \in PLACE, mult(p, K - Post(t_i)) \leq mult(p, I(t_i))$ both 6.16 and 6.17 become

$$M(p) \leq mult(p, K - \sum_i Post(t_i)) \tag{6.18}$$

as expected.

Now given a $p \in PLACE$, such that for some $t_j \in T_\mu$,

$$mult(p, I(t_j)) = min(mult(p, K - \sum_i Post(t_i)), mult(p, I(t_1)), \ldots, mult(p, I(t_n)))$$
$$\tag{6.19}$$

then for the P-net, the inhibitor-capacity condition is (from 6.16 and 6.19)

$$M(p) \leq mult(p, I(t_j)) \tag{6.20}$$

and for the equivalent CP-net (from 6.17 and 6.19)

$$M(p) \leq mult(p, I(t_j)) - \sum_{i \neq j} [K(p) - min(mult(p, K - Post(t_i)), mult(p, I(t_i)))] \tag{6.21}$$

By comparing 6.20 and 6.21, it can be seen that the inhibitor-capacity condition is (in general) more restrictive for the equivalent CP-net, because the part in square brackets is always non-negative. This is demonstrated by proving the following proposition.

Proposition 6.2 $\forall p \in PLACE, \forall t_i \in TRANS$

$$[K(p) - min(mult(p, K - Post(t_i)), mult(p, I(t_i)))] \geq mult(p, Post(t_i)) \geq 0$$

Proof:

We shall use the notation [] to represent the part in square brackets. Note that $K(p) < \infty$ from equation 6.19 and the transformation restriction 6.6, and consider two cases: $\forall p \in PLACE$, and $\forall t_i \in TRANS$

1. For $mult(p, K - Post(t_i)) \leq mult(p, I(t_i))$,

$$\begin{aligned} [] &= K(p) - mult(p, K - Post(t_i)) \\ &= mult(p, Post(t_i)) \geq 0 \end{aligned} \tag{6.22}$$

2. For $mult(p, K - Post(t_i)) > mult(p, I(t_i))$,

$$\begin{aligned} [] &= K(p) - mult(p, I(t_i)) \\ &> mult(p, Post(t_i)) \\ &> 0 \end{aligned} \tag{6.23}$$

$\square$

Theorem 6.3 *The inhibitor-capacity condition for a P-net (P) is less restrictive than the inhibitor-capacity condition for the equivalent CP-net ($\mathcal{T}(P)$). That is, for a finite $T_\mu \in \mu TRANS$ with $T_\mu = \sum_{i=1}^n t_i$ where for all $i \in In, t_i \in TRANS$,*

$$(K - Post'(T_\mu)) \cap I'(T_\mu) \geq K - \sum_i (K - IK(t_i))$$

Proof:

There are several cases.

Case 1: The P-net is a CP_K-net, then from the proof of theorem 6.2 the inhibitor-capacity vectors are the same.

$$
\begin{aligned}
(K - Post'(T_\mu)) \cap I'(T_\mu) &= K - Post'(T_\mu) \\
&= K - \sum_i (K - IK(t_i))
\end{aligned}
$$

Case 2: $\exists p \in PLACE : mult(p, I(t_i)) < mult(p, K - Post(t_i))$.

Here we consider the inhibitor-capacity vectors component-wise from equations 6.16 and 6.17. Equation 6.18 shows that the vectors are equal when no inhibitor is active. Hence we only need to consider the case where there are active inhibitors.

Firstly we define the set of transition modes which have active inhibitors for place $p \in PLACE$

$$TRANSI(p) = \{tr \mid tr \in TRANS \text{ and } mult(p, I(tr)) < mult(p, K - Post(tr))\}$$
(6.24)

and the set of underlying places that have active inhibitors

$$PLACEI = \{p \mid p \in PLACE \text{ and } TRANSI(p) \neq \emptyset\} \qquad (6.25)$$

For each $p \in PLACE$, let the corresponding component of the P-net inhibitor-capacity vector (from equation 6.16) be

$$IC_P(p) = min(mult(p, K - \sum_i Post(t_i)), mult(p, I(t_1)), \ldots, mult(p, I(t_n)))$$
(6.26)

and let the corresponding equivalent CP-net inhibitor-capacity component (from equation 6.17) be

$$IC_{CP}(p) = K(p) - \sum_{i=1}^n [K(p) - min(mult(p, K - Post(t_i)), mult(p, I(t_i)))] \quad (6.27)$$

Hence we need to show that $\forall p \in PLACE, IC_P(p) \geq IC_{CP}(p)$. This has already been demonstrated for all $p \in PLACE \setminus PLACEI$.

Let t_j be the transition mode which has the most restrictive inhibitor on $p \in PLACEI$ (i.e. $t_j \in TRANSI(p)$ and $\forall t_k \in TRANSI(p), mult(p, I(t_j)) \leq mult(p, I(t_k))$).

Now there are two cases to consider for the P-net when there is at least one active inhibitor for place p.

65

Case 2.1: For all $p \in PLACEI$, $t_j \in TRANSI(p)$, for all $t_i \in T_\mu$, $t_j \in T_\mu$,

$$mult(p, K - \textstyle\sum_i Post(t_i)) < mult(p, I(t_j)) < mult(p, K - Post(t_j))$$

then

$$
\begin{aligned}
IC_P(p) &= mult(p, K - \textstyle\sum_i Post(t_i)) \\
&= K(p) - mult(p, Post(t_j)) - \sum_{i \neq j} mult(p, Post(t_i)) \quad (6.28)
\end{aligned}
$$

Case 2.2: For $t_j \in TRANSI(p)$, $t_i \in T_\mu$, $t_j \in T_\mu$ and for all $p \in PLACEI$,

$$mult(p, I(t_j)) \leq mult(p, K - \textstyle\sum_i Post(t_i))$$

then

$$IC_P(p) = mult(p, I(t_j)) \quad (6.29)$$

and for the equivalent CP-net (from 6.21)

$$IC_{CP}(p) = mult(p, I(t_j)) - \sum_{i \neq j} [K(p) - min(mult(p, K - Post(t_i)), mult(p, I(t_i)))]$$

$$(6.30)$$

From proposition 6.2, it follows directly that $IC_P(p) \geq IC_{CP}(p)$ for case 2.2.

In case 2.1, since $mult(p, I(t_j)) < K(p) - mult(p, Post(t_j))$, then $IC_P(p) > IC_{CP}(p)$ because $\forall t_i \in T_\mu, [] \geq mult(p, Post(t_i))$ from proposition 6.2.

□

Now let us examine the condition under which the inhibitor-capacity vectors are equal. This can only be true when case 1 of the proof of proposition 6.2 applies for all the transition modes in T_μ (except t_j) and all the images of the post maps with arguments taken from T_μ (except $Post(t_j)$) have zero multiplicities for p.

Let $T_\mu' = T_\mu - \{t_j\}$ and consider the case when

$$\forall t_i \in T_\mu', mult(p, K - Post(t_i)) \leq mult(p, I(t_i)) \quad (6.31)$$

The enabling condition for the equivalent CP-net becomes (from 6.21)

$$mult(p, \sum_i Pre(t_i)) \leq M(p) \leq mult(p, I(t_j)) - \sum_{i \neq j} mult(p, Post(t_i)) \quad (6.32)$$

The condition 6.31 implies that $t_j \notin T_\mu'$ and hence that t_j must not exhibit any self concurrency. It also means that t_j cannot be concurrently enabled with any other t_i, that has an active inhibitor on place p. For the equivalent CP-net to have the same enabling condition, we require further that for all $t_i \in T_\mu'$, $mult(p, post(t_i)) = 0$. Alternatively, if $mult(p, post(t_i)) \neq 0$, then t_j and t_i must not be concurrently enabled in the P-net.

This leads to the main theorem of this section, which states the necessary and sufficient conditions on the P-net for the transformation to preserve true concurrency.

Before stating the theorem, in order to refer to enabling conditions and inhibitor-capacity vectors in a convenient manner (in the theorem and proof), it is useful to introduce the following abbreviations.

$E_P = Pre'(T_\mu) \leq M \leq (K - Post'(T_\mu)) \cap I'(T_\mu)$
$E_{CP} = Pre'(T_\mu) \leq M \leq K - \sum_i(K - IK(t_i))$
$IC_P = (K - Post'(T_\mu)) \cap I'(T_\mu)$
$IC_{CP} = K - \sum_i(K - IK(t_i))$

Theorem 6.4 $\forall M \in [M_0), E_P \Leftrightarrow \underline{Pre}'(T_\mu) \leq \rho(M)$ *iff in the P-net, for all* $p \in PLACEI$ *and for all* $t_j, t_k \in TRANSI(p)$

1. t_j and t_k cannot be concurrently enabled, i.e.

$$Pre'(t_j + t_k) \not\leq (K - Post'(t_j + t_k)) \cap I'(t_j + t_k)$$

and

2. $\forall t_i \in TRANS \setminus TRANSI(p)$, *either*

   (a) $mult(p, post(t_i)) = 0$; *or*

   (b) t_j and t_i cannot be concurrently enabled, i.e.

$$Pre'(t_j + t_i) \not\leq (K - Post'(t_j + t_i)) \cap I'(t_j + t_i)$$

Proof of Implication (*only if*):

Firstly, note that $\underline{Pre}'(T_\mu) \leq \rho(M) \Leftrightarrow E_{CP}$ from equation 6.13.

From theorem 6.3, if E_P is false then so is E_{CP}. When E_P is true, we must have $IC_P = IC_{CP}$ to ensure that E_{CP} is also true (so that E_P and E_{CP} are equivalent). When $IC_P \neq IC_{CP}$, we may be able to choose a marking, M, such that E_P is true and E_{CP} is false. Thus when $IC_P \neq IC_{CP}$, we must ensure that E_P is false, for the two enabling conditions to be equivalent. This then implies certain restrictions on the P-net.

There are two ways in which the inhibitor-capacity vectors (IC_P and IC_{CP}) can be the same.

Case 1: There are no active inhibitors, $\forall tr \in TRANS, K - Post(tr) \leq I(tr)$.

The P-net is a CP_K-net, which ensures that $IC_P = IC_{CP}$ and $PLACEI = \emptyset$. Thus, in this case, $E_P \Leftrightarrow E_{CP}$ implies that in the P-net, $PLACEI = \emptyset$ and conditions 1 and 2 of the theorem follow trivially.

Case 2: There is no more than one active inhibitor per place $p \in PLACE$. The places that do have an active inhibitor are in the set $PLACEI$.

Consider the inhibitor-capacity vectors component-wise as defined by equations 6.26 and 6.27, where $T_\mu = \sum_i t_i$. There are two cases:

Case 2.1: Places not affected by inhibitors: for $p \in PLACE \setminus PLACEI$, we have $\forall t_i \in TRANS, mult(p, K - Post(t_i)) \leq mult(p, I(t_i))$

$$\begin{aligned}
IC_P(p) &= mult(p, K - \sum_i Post(t_i)) \\
&= K(p) - \sum_i mult(p, Post(t_i)) \\
&= IC_{CP}(p) \quad\quad\quad\quad (6.33)
\end{aligned}$$

Places not affected by inhibitors have no further restrictions as the enabling conditions on them for P and $T(P)$ are equivalent. This is the same as case 1 for vector components.

Case 2.2: Places affected by inhibitors. For $p \in PLACEI$ let the transition mode, $t_j \in TRANS$, have an active inhibitor on place p, $mult(p, I(t_j)) < mult(p, K - Post(t_j))$ and thus $\forall t_i \in TRANS \setminus \{t_j\}, mult(p, K - Post(t_i)) \leq mult(p, I(t_i))$.

There are two cases to consider for $IC_P(p)$, where we assume $t_j \in T_\mu$. (If $t_j \notin T_\mu$, then from equation 6.18, the enabling conditions are equivalent and concurrency is preserved.)

A. $mult(p, K - \sum_i Post(t_i)) < mult(p, I(t_j))$, then in a similar way to case 2.1 of the proof of theorem 6.3 (see equation 6.28)

$$IC_P(p) = K(p) - mult(p, Post(t_j)) - \sum_{i \neq j} mult(p, Post(t_i)) \qquad (6.34)$$

B. $mult(p, I(t_j)) \leq mult(p, K - \sum_i Post(t_i))$, then in a similar way to case 2.2 of the proof of theorem 6.3 (see equation 6.29)

$$IC_P(p) = mult(p, I(t_j)) \qquad (6.35)$$

Substituting the above conditions into equation 6.30 gives

$$IC_{CP}(p) = mult(p, I(t_j)) - \sum_{i \neq j} mult(p, Post(t_i)) \qquad (6.36)$$

We know from the proof of theorem 6.3 that in case A, $IC_P(p) > IC_{CP}(p)$ and hence $IC_P(p) \neq IC_{CP}(p)$.

Comparing equations 6.35 and 6.36 we see that the inhibitor-capacity components can only be equal (i.e. $IC_P(p) = IC_{CP}(p) = mult(p, I(t_j))$) if

$$\forall t_i \in T_\mu', mult(p, Post(t_i)) = 0 \qquad (6.37)$$

(Note that case A does not occur with the restriction of equation 6.37)

Since T_μ can be *any* finite multiset, the above condition 6.37 needs to span the whole set of transition modes, $TRANS$, of the P-net, if E_P can be true, or else we must ensure that E_P is false. For each $t_i \in TRANS \setminus \{t_j\}$, either

$$mult(p, Post(t_i)) = 0 \qquad (6.38)$$

or else t_j and t_i are not concurrently enabled

$$Pre'(t_j + t_i)) \not\leq (K - Post'(t_j + t_i)) \cap I'(t_j + t_i) \qquad (6.39)$$

If there are two or more active inhibitors on a place $p \in PLACE$, then the proof of proposition 6.2 (see equation 6.23) shows that $IC_P(p) > IC_{CP}(p)$.

In the above analysis, the assumption that $mult(t_j, T_\mu) = 1$ has been made. When $mult(t_j, T_\mu) > 1$ we must ensure that E_P is false, i.e. that t_j is not self-concurrently enabled. This is done by placing restrictions on the *Pre, Post*

and inhibitor maps in a similar way to inequality 6.39. Further, we may now allow other transition modes that have active inhibitors for place p, so long as the *Pre*, *Post* and inhibitor maps never allow any of them to be concurrently enabled. To guarantee this requires condition 1 of the theorem. (Note that only two modes need to be considered, as adding a mode would not reduce the value of the left hand side and would not increase the value of the right hand side of the inequality.)

We may now generalise the above to a set of active inhibitors for place p, $TRANSI(p)$. Since only one transition mode with an active inhibitor for place p can be included in the multiset of concurrently enabled modes, the post maps of transitions with active inhibitors are not relevant and hence equations 6.38 and 6.39 become that of condition 2 of the theorem. $\square$

Proof of Reverse Implication (*if*):

Case 1: $PLACEI = \emptyset$ (the conditions 1 and 2 of the theorem are not required)

$PLACEI = \emptyset$ implies that there are no active inhibitors and the P-net reduces to a CP$_K$-net. Hence the enabling conditions are equivalent by theorem 6.2.

Case 2: $PLACEI \neq \emptyset$

For all $p \in PLACE \backslash PLACEI$, the enabling conditions on place p are equivalent as indicated by equation 6.18 as there are no active inhibitors associated with place p.

For all $p \in PLACEI$, E_P is given by equation 6.15 for $T_\mu = \sum_{i=1}^{n} t_i$ as before. If $t_j, t_k \in TRANSI(p)$ and $t_j, t_k \in T_\mu$ or if $(T_\mu \cap TRANSI(p) = \{t_j\}) \wedge mult(t_j, T_\mu) \geq 2$ then from condition 1 of the theorem, E_P is false and from theorem 6.3 so is E_{CP} and they are therefore equivalent.

If $(T_\mu \cap TRANSI(p) = \{t_j\}) \wedge (mult(t_j, T_\mu) = 1)$, then when condition 2a of the theorem applies, (from equation 6.16) the component-wise E_P can be obtained from equation 6.20.

$$mult(p, \sum_i Pre(t_i)) \leq M(p) \leq mult(p, I(t_j)) \tag{6.40}$$

Like-wise for condition 2a of the theorem and from equation 6.32, the component-wise E_{CP} is the same and hence they are equivalent.

If for $t_i \in TRANS \setminus TRANSI(p), mult(p, Post(t_i)) > 0$, condition 2b of the theorem implies that E_P is false and again from theorem 6.3 so is E_{CP} and they are therefore equivalent. $\square$

6.5 Illustration of Transformations

This section provides a number of simple examples to illustrate the transformation from P-nets to CP-nets, particularly theorem 6.4. The graphical form that was used in the previous example is also adopted here. A formal definition of the graphical form is provided in the next chapter.

$$
\begin{array}{l}
S = \{s1, s2\} \\
T = \{t1, t2\} \\
C = \{\{a\}\} \\
C(s1) = C(s2) = C(t1) = C(t2) = \{a\} \\
Pre(t1, a) = \{((s1, a), 1), ((s2, a), 0)\} \\
Pre(t2, a) = \{((s1, a), 1), ((s2, a), 1)\} \\
Post(t1, a) = Post(t2, a) = \emptyset \\
M_0 = \{((s1, a), 3), ((s2, a), 1)\} \\
I(t1, a) = \{((s1, a), 3), ((s2, a), \infty)\} \\
I(t2, a) = \{((s1, a), \infty), ((s2, a), \infty)\} \\
K = \{((s1, a), 5), ((s2, a), \infty)\}
\end{array}
$$

Graphical Form
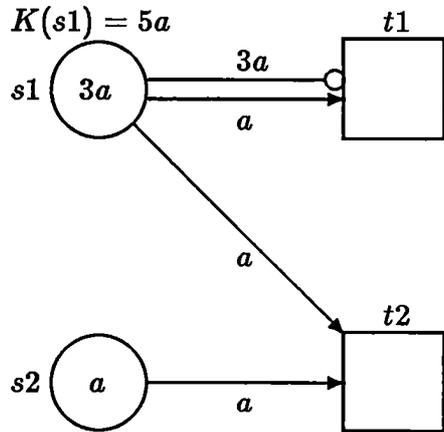
Figure 6.7: Self concurrency in P-net: P1

6.5.1 Example1: Self Concurrency

The fact that self concurrency is not preserved by the transformation, $\mathcal{T}$, has been illustrated in section 6.3. Here we introduce a slightly more complicated example that also demonstrates the lack of self-concurrency preservation. It is introduced now so that it may be compared with the next example where concurrency is preserved.

Consider the P-net, P1, of figure 6.7, which consists of two places, two transitions, 3 normal arcs and an inhibitor arc. There is only one colour set, the singleton $\{a\}$. It is associated with each place and transition. It can be seen that P1 is essentially a P/T-system with inhibitors. The capacity of place $s1$ is $5a$, and it is marked with $3a$ (complying with the capacity constraint), while place $s2$ has infinite capacity and is marked by a single a. Transition $t1$ has an active inhibitor on place $s1$ which means that $t1$ can be enabled only if the marking
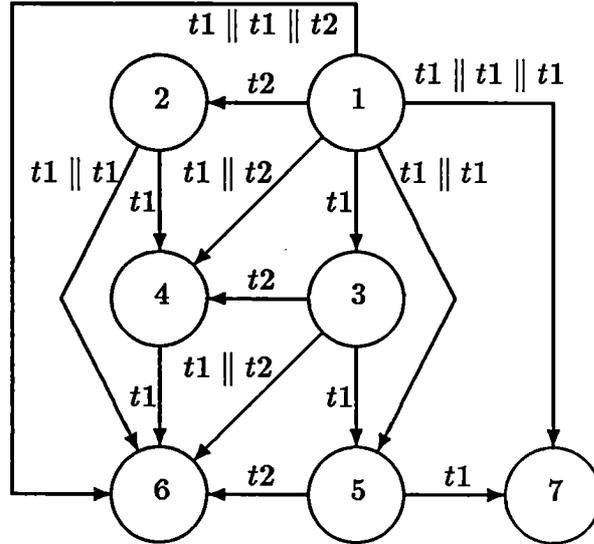
Figure 6.8: Reachability Graph for P-net: P1

of $s1$ is no more than $3a$. We may see from the definition of enabling that the following multisets of transition modes are enabled in the initial marking.

- $\{((t1,a),1)\}$

- $\{((t1,a),n)|n = 1,2\}$

- $\{((t1,a),n)|n = 1,2,3\}$

- $\{((t2,a),1)\}$

- $\{((t2,a),1),((t1,a),1)\}$

- $\{((t2,a),1),((t1,a),n)|n = 1,2\}$

P1 exhibits both self concurrency for mode $(t1,a)$ and 'mutual' concurrency for modes $(t1,a)$ and $(t2,a)$. The full reachability graph is given in figure 6.8. For this reachability graph we have used the convention that the transition mode $(t1,a)$ is identified with the transition $t1$ (and similarly $(t2,a)$ is identified with $t2$), since each transition has only one mode (as in P/T-systems). We have also used the notation $t1 \parallel t2$ to represent the multiset $T_\mu = t1 + t2$, to emphasize that the transitions occur in parallel (i.e. in one step). These conventions are followed for the remainder of this section.

If we apply the transformation to P1, we obtain the CP-net, $\mathcal{T}(P1)$, depicted in figure 6.9. The complementary place $\widehat{s1}$ and extra arcs have been added to simulate the capacity and inhibitor conditions. This net has the following multisets of transition modes enabled in the initial marking.

- $\{((t1,a),1)\}$

71

CP-Net:$\mathcal{T}$(P1)

$$S = \{s1, \widehat{s1}, s2\}$$
$$T = \{t1, t2\}$$
$$\mathcal{C} = \{\{a\}\}$$
$$C(s1) = C(\widehat{s1}) = C(s2) = C(t1) = C(t2) = \{a\}$$
$$Pre(t1, a) = \{((s1, a), 1), ((\widehat{s1}, a), 2), ((s2, a), 0)\}$$
$$Pre(t2, a) = \{((s1, a), 1), ((\widehat{s1}, a), 0), ((s2, a), 1)\}$$
$$Post(t1, a) = \{((s1, a), 0), ((\widehat{s1}, a), 3), ((s2, a), 0)\}$$
$$Post(t2, a) = \{((s1, a), 0), ((\widehat{s1}, a), 1), ((s2, a), 0)\}$$
$$M_0 = \{((s1, a), 3), ((\widehat{s1}, a), 2), ((s2, a), 1)\}$$

Graphical Form
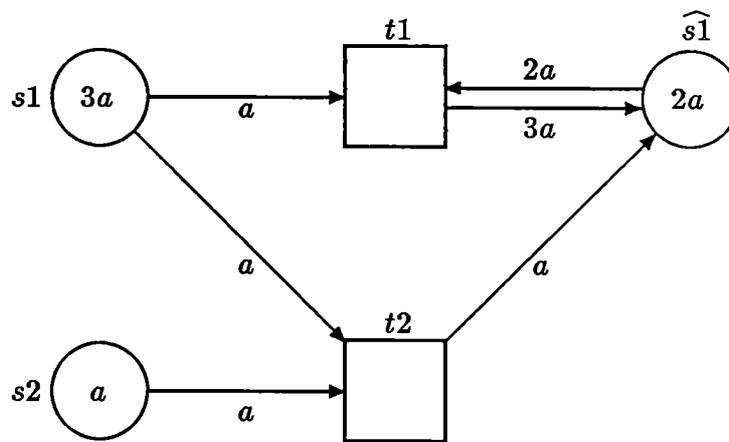
Figure 6.9: CP-net equivalent:$\mathcal{T}$(P1)
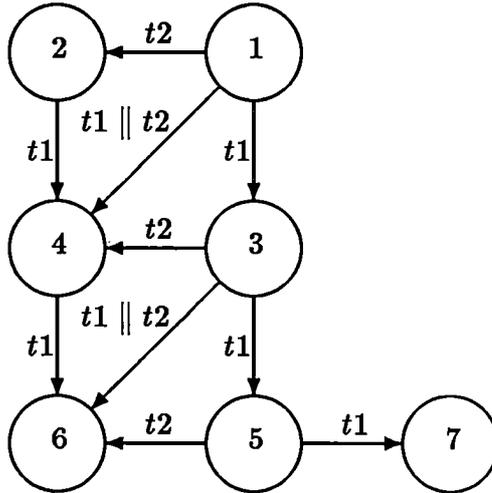
Figure 6.10: Reachability Graph for $T(\text{P1})$

- $\{((t2, a), 1)\}$

- $\{((t2, a), 1), ((t1, a), 1)\}$

The reachability graph for $T(\text{P1})$ is given in figure 6.10. We can see that in this case T does not preserve self concurrency, although mutual concurrency is retained. This is what we expect from theorem 6.4 as it states that for concurrency to be preserved, no self concurrency is allowed for transition modes that have active inhibitors. Here, transition $t1$ (mode $(t1, a)$) has an active inhibitor on place $s1$ (underlying place $(s1, a)$).

6.5.2 Example2: No Self Concurrency

An example where no self concurrency exists for the P-net is given by P2, depicted in figure 6.11. This example is the same as the previous one except that the *Pre* map for $t1$ now places a demand of $2a$ on $s1$ instead of a. Its transformation, $T(\text{P2})$, is shown in figure 6.12. It can be seen that both P2 and $T(\text{P2})$ have the following multisets of transition modes enabled in their initial markings.

- $\{((t1, a), 1)\}$

- $\{((t2, a), 1)\}$

- $\{((t2, a), 1), ((t1, a), 1)\}$

They have isomorphic reachability graphs. The basic reachability graph for both is depicted in figure 6.13. (The markings are different for the two nets, but related by ρ.)

This illustrates a case in which concurrency is preserved by T. This is because conditions 1 and 2(a) of theorem 6.4 are satisfied by P2.

73

$$S = \{s1, s2\}$$
$$T = \{t1, t2\}$$
$$C = \{\{a\}\}$$
$$C(s1) = C(s2) = C(t1) = C(t2) = \{a\}$$
$$Pre(t1, a) = \{((s1, a), 2), ((s2, a), 0)\}$$
$$Pre(t2, a) = \{((s1, a), 1), ((s2, a), 1)\}$$
$$Post(t1, a) = Post(t2, a) = \emptyset$$
$$M_0 = \{((s1, a), 3), ((s2, a), 1)\}$$
$$I(t1, a) = \{((s1, a), 3), ((s2, a), \infty)\}$$
$$I(t2, a) = \{((s1, a), \infty), ((s2, a), \infty)\}$$
$$K = \{((s1, a), 5), ((s2, a), \infty)\}$$

Graphical Form

Figure 6.11: No Self concurrency in P-net: P2

CP-Net:$\mathcal{T}$(P2)

$$S = \{s1, \widehat{s1}, s2\}$$
$$T = \{t1, t2\}$$
$$C = \{\{a\}\}$$
$$C(s1) = C(\widehat{s1}) = C(s2) = C(t1) = C(t2) = \{a\}$$
$$Pre(t1, a) = \{((s1, a), 2), ((\widehat{s1}, a), 2), ((s2, a), 0)\}$$
$$Pre(t2, a) = \{((s1, a), 1), ((\widehat{s1}, a), 0), ((s2, a), 1)\}$$
$$Post(t1, a) = \{((s1, a), 0), ((\widehat{s1}, a), 4), ((s2, a), 0)\}$$
$$Post(t2, a) = \{((s1, a), 0), ((\widehat{s1}, a), 1), ((s2, a), 0)\}$$
$$M_0 = \{((s1, a), 3), ((\widehat{s1}, a), 2), ((s2, a), 1)\}$$
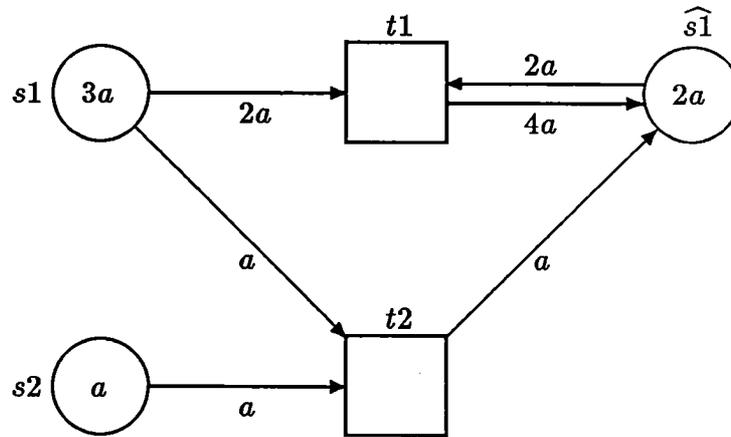
Graphical Form

Figure 6.12: CP-net equivalent:$\mathcal{T}$(P2)
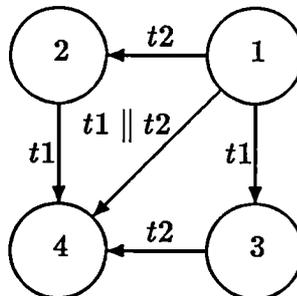
Figure 6.13: Reachability Graph for P2 and $\mathcal{T}$(P2)

P-Net:P3

$$
\begin{aligned}
&S = \{s1, s2\} \\
&T = \{t1, t2\} \\
&C = \{\{a\}\} \\
&C(s1) = C(s2) = C(t1) = C(t2) = \{a\} \\
&Pre(t1, a) = \{((s1, a), 2)\} \\
&Pre(t2, a) = \{((s2, a), 1)\} \\
&Post(t1, a) = \emptyset \\
&Post(t2, a) = \{((s1, a), 1)\} \\
&M_0 = \{((s1, a), 3), ((s2, a), 1)\} \\
&I(t1, a) = \{((s1, a), 3), ((s2, a), \infty)\} \\
&I(t2, a) = \{((s1, a), \infty), ((s2, a), \infty)\} \\
&K = \{((s1, a), 5), ((s2, a), \infty)\}
\end{aligned}
$$

Graphical Form
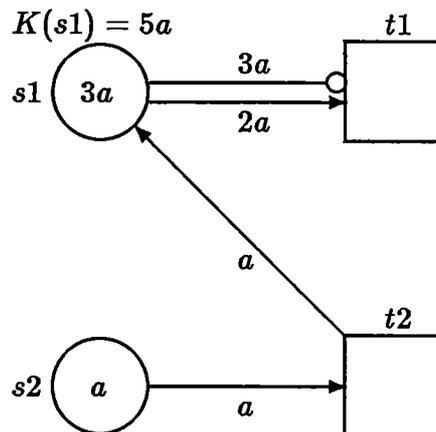
Figure 6.14: P-net: P3 with Post map involving an inhibitor place

6.5.3   Example3: Effect of Post Map

This example illustrates the effect of an output arc to a place that is affected by an active inhibitor. Consider the P-net, P3, of figure 6.14 which is the same as P2 except that the input arc from $s1$ to $t2$ has been replaced by an output arc from $t2$ to $s1$. In this case $Post(s1, t2; a) \neq \emptyset$.

The reachability graph is depicted in figure 6.15. It can be seen that mutual concurrency exists between modes $(t1, a)$ and $(t2, a)$, in the initial marking.

The corresponding CP-net, $T(P3)$, is shown in figure 6.16 and its reachability graph in figure 6.17, where it can be seen that there is no concurrency.

In this case, it is the non-empty $Post$ map, $Post(s1, t2; a) = a$, that prevents T from preserving concurrency.
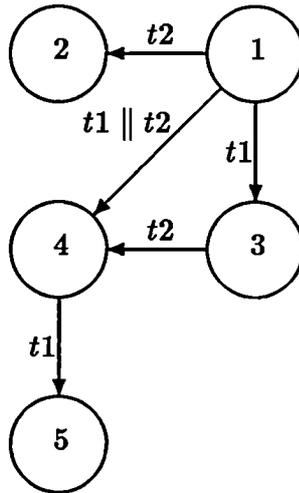
Figure 6.15: Reachability Graph for P3

CP-Net:$\mathcal{T}$(P3)

$$S = \{s1, \widehat{s1}, s2\}$$
$$T = \{t1, t2\}$$
$$C = \{\{a\}\}$$
$$C(s1) = C(\widehat{s1}) = C(s2) = C(t1) = C(t2) = \{a\}$$
$$Pre(t1, a) = \{((s1, a), 2), ((\widehat{s1}, a), 2), ((s2, a), 0)\}$$
$$Pre(t2, a) = \{((s1, a), 0), ((\widehat{s1}, a), 1), ((s2, a), 1)\}$$
$$Post(t1, a) = \{((s1, a), 0), ((\widehat{s1}, a), 4), ((s2, a), 0)\}$$
$$Post(t2, a) = \{((s1, a), 1), ((\widehat{s1}, a), 0), ((s2, a), 0)\}$$
$$M_0 = \{((s1, a), 3), ((\widehat{s1}, a), 2), ((s2, a), 1)\}$$

Graphical Form
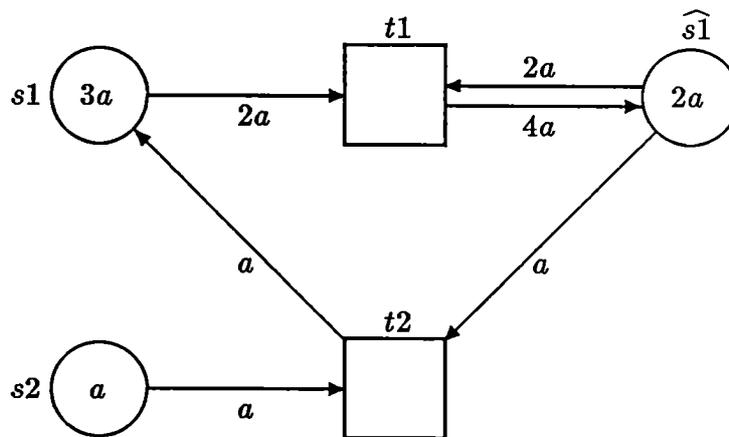
Figure 6.16: CP-net equivalent:$\mathcal{T}$(P3)

77
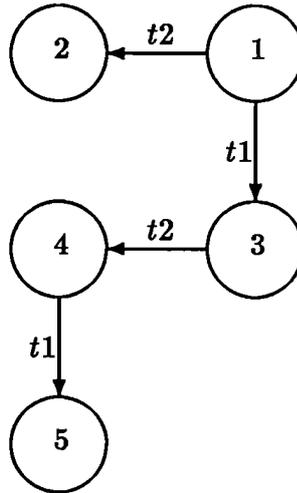
Figure 6.17: Reachability Graph for $\mathcal{T}(P3)$

6.5.4 Example4: Two Inhibitors

A similar situation occurs if two active inhibitors affect the same place. An example of this is shown in figure 6.18, which depicts a P-net, P4, that is again similar to P2, except we have now replaced the input arc by an inhibitor arc, with a threshold of $2a$. The initial marking of $s1$ has been decreased to $2a$, to allow modes $(t1, a)$ and $(t2, a)$ to be concurrently enabled in the initial marking. There are now two inhibitors for place $s1$.

The reachability graph for P4 is isomorphic to that of P2 (see figure 6.13).

When P4 is transformed under $\mathcal{T}$, we obtain the CP-net of figure 6.19. Its reachability graph is given in figure 6.20 which demonstrates that concurrency has not been preserved in this situation, as condition 1 of the theorem has been violated.

6.5.5 Example5: Two Inhibitors with No Concurrency

Finally, if we ensure that the modes that have active inhibitors on a place cannot be concurrently enabled, then (provided condition 2 of the theorem applies), whatever concurrency is left will be preserved. An example is given in figure 6.21. Here an extra input arc, $(s1, t2)$, has been included to make sure that $(t1, a)$ and $(t2, a)$ are not concurrently enabled. In this case, there is no concurrency in the P-net, and hence its transformation, $\mathcal{T}(P5)$, depicted in figure 6.22 also has no concurrency.

The reachability graphs for P5 and $\mathcal{T}(P5)$ are isomorphic and given in figure 6.23.

P-Net:P4

$S = \{s1, s2\}$
$T = \{t1, t2\}$
$C = \{\{a\}\}$
$C(s1) = C(s2) = C(t1) = C(t2) = \{a\}$
$Pre(t1, a) = \{((s1, a), 2), ((s2, a), 0)\}$
$Pre(t2, a) = \{((s1, a), 0), ((s2, a), 1)\}$
$Post(t1, a) = Post(t2, a) = \emptyset$
$M_0 = \{((s1, a), 2), ((s2, a), 1)\}$
$I(t1, a) = \{((s1, a), 3), ((s2, a), \infty)\}$
$I(t2, a) = \{((s1, a), 2), ((s2, a), \infty)\}$
$K = \{((s1, a), 5), ((s2, a), \infty)\}$

Graphical Form
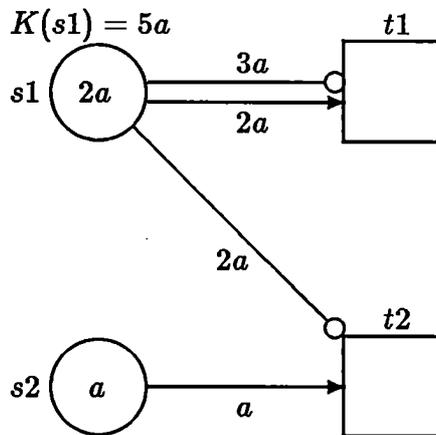
Figure 6.18: Two Inhibitors for Place S1

79

CP-Net: $\mathcal{T}$ (P4)

$$S = \{s1, \widehat{s1}, s2\}$$
$$T = \{t1, t2\}$$
$$C = \{\{a\}\}$$
$$C(s1) = C(\widehat{s1}) = C(s2) = C(t1) = C(t2) = \{a\}$$
$$Pre(t1, a) = \{((s1, a), 2), ((\widehat{s1}, a), 2), ((s2, a), 0)\}$$
$$Pre(t2, a) = \{((s1, a), 0), ((\widehat{s1}, a), 3), ((s2, a), 1)\}$$
$$Post(t1, a) = \{((s1, a), 0), ((\widehat{s1}, a), 4), ((s2, a), 0)\}$$
$$Post(t2, a) = \{((s1, a), 0), ((\widehat{s1}, a), 3), ((s2, a), 0)\}$$
$$M_0 = \{((s1, a), 2), ((\widehat{s1}, a), 3), ((s2, a), 1)\}$$

Graphical Form
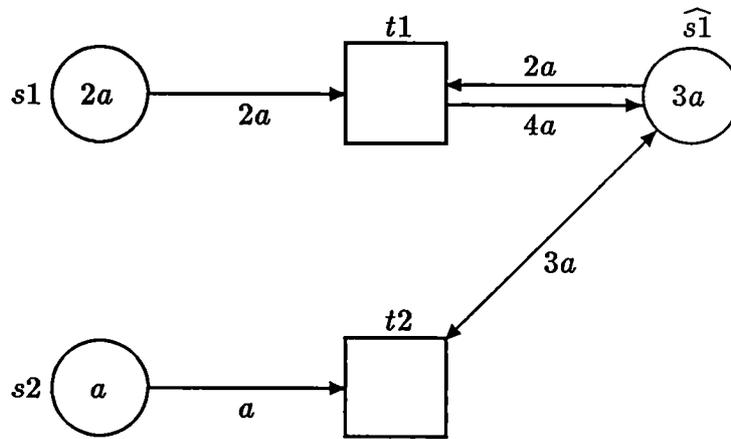
Figure 6.19: CP-net equivalent: $\mathcal{T}$ (P4)
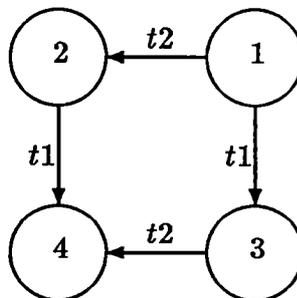
Figure 6.20: Reachability Graph for $\mathcal{T}$ (P4)

P-Net:P5

$$
\begin{aligned}
&S = \{s1, s2\} \\
&T = \{t1, t2\} \\
&C = \{\{a\}\} \\
&C(s1) = C(s2) = C(t1) = C(t2) = \{a\} \\
&Pre(t1, a) = \{((s1, a), 2), ((s2, a), 0)\} \\
&Pre(t2, a) = \{((s1, a), 2), ((s2, a), 1)\} \\
&Post(t1, a) = Post(t2, a) = \emptyset \\
&M_0 = \{((s1, a), 2), ((s2, a), 1)\} \\
&I(t1, a) = \{((s1, a), 3), ((s2, a), \infty)\} \\
&I(t2, a) = \{((s1, a), 2), ((s2, a), \infty)\} \\
&K = \{((s1, a), 5), ((s2, a), \infty)\}
\end{aligned}
$$
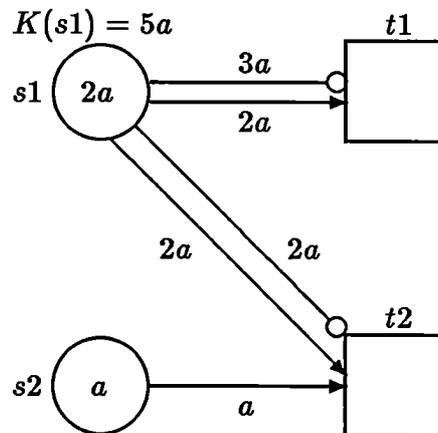
Graphical Form

Figure 6.21: Two Inhibitors for Place S1: No concurrency

$$S = \{s1, \widehat{s1}, s2\}$$
$$T = \{t1, t2\}$$
$$\mathcal{C} = \{\{a\}\}$$
$$C(s1) = C(\widehat{s1}) = C(s2) = C(t1) = C(t2) = \{a\}$$
$$Pre(t1, a) = \{((s1, a), 2), ((\widehat{s1}, a), 2), ((s2, a), 0)\}$$
$$Pre(t2, a) = \{((s1, a), 2), ((\widehat{s1}, a), 3), ((s2, a), 1)\}$$
$$Post(t1, a) = \{((s1, a), 0), ((\widehat{s1}, a), 4), ((s2, a), 0)\}$$
$$Post(t2, a) = \{((s1, a), 0), ((\widehat{s1}, a), 5), ((s2, a), 0)\}$$
$$M_0 = \{((s1, a), 2), ((\widehat{s1}, a), 3), ((s2, a), 1)\}$$
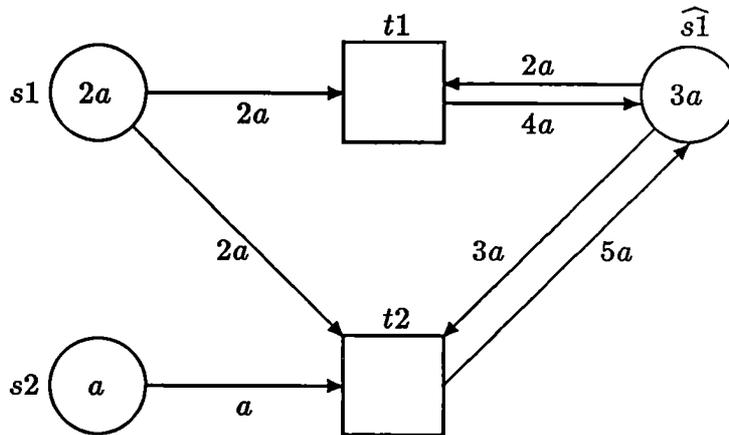
Graphical Form

Figure 6.22: CP-net equivalent:$\mathcal{T}$(P5)
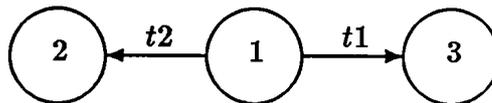
Figure 6.23: Reachability Graph for P5 and $\mathcal{T}$(P5)

Chapter 7

P-Graphs and P-Graph Schemas

7.1 Introduction

The main purpose of this chapter is to define a graphical form for P-nets. The work here combines Petri nets and Abstract Data Types (ADTs) within the same algebraic framework. It has been inspired by the work on Predicate/Transition nets [71,70], Coloured Petri Nets [87] and Algebraic nets [123]. The approach is similar to that of [143] but differs in a number of ways. Firstly we consider nets with inhibitors and capacities and more general arc inscriptions. (The axioms of ADTs are not considered here, but they can be added easily). Secondly the nets can be defined at two levels of abstraction. At the concrete level, places are typed by sets and markings and capacities are multisets over these sets. This has similarities with [123] and is appropriate for the specification of concrete systems such as a particular service or protocol. At the abstract level, places are many-sorted (i.e. a sort is associated with each place) and inscriptions, capacities and markings are defined on the level of terms. This is similar to [143,19,92,9] and is appropriate for specifying classes of systems. The concrete form is called a P-Graph, because it provides a definition for the graphical form of a P-net. Similarly, the abstract form is known as a P-Graph schema (following Vautherin's terminology) or abstract P-Graph (suggested by Jensen [90]).

The chapter concentrates on the (concrete) P-Graph and shows how subclasses such as CP-Graphs, Many-sorted PrT-nets (MPrT-nets) and Many-sorted Algebraic nets can be derived. Why do we need many-sorted versions? The many-sorted versions overcome some difficulties experienced with their singled-sorted predecessors [70,123]. To allow for the many-sorted nature of applications, the carrier of the single-sorted high-level net has to be a union of more basic sets, necessitating the use of partial functions. All variables are typed by the single carrier. This leads to a number of difficulties in typing and interpretation. These problems are investigated and it is shown how they are elegantly solved by allowing the terms to be built from a many-sorted signature with variables.

An early attempt at defining the P-Graph appears in [22]. This chapter expands on work recently published by the author [27].

7.2 Concepts from Algebraic Specification

In the P-Graph, we shall inscribe arcs with multisets of terms involving variables, and transitions with Boolean expressions. Many-sorted signatures provide an appropriate mathematical framework for this representation. Signatures provide a convenient way to characterise many-sorted algebras at a syntactic level. This section introduces the concepts of signatures, terms and many-sorted algebras that will be required for the definition of the P-Graph and abstract P-Graph. We make use of the ideas found in [64,98] for example.

7.2.1 Signatures

A *many-sorted* (or *R-sorted*) signature, Σ, is a pair:

$$\Sigma = (R, \Omega)$$

where

- R is a set of sorts (the **names** of sets, e.g. *Int* for the integers); and

- Ω is a set of operators (the **names** of functions) together with their *arity* in R which specifies the names of the domain and co-domain of each of the operators.

The arity is a function from the set of operator names to $R^* \times R$, where R^* is the set of finite sequences, including the empty string, ε, over R. Thus every operator in Ω is indexed by a pair (σ, r), $\sigma \in R^*$ and $r \in R$ denoted by $w_{(\sigma,r)}$. $\sigma \in R^*$ is known as the *input* or *argument* sorts, and r as the *output* or *range* sort of operator w. (The sequence of input sorts will define a cartesian product as the domain of the function corresponding to the operator and the output sort will define its co-domain - but this is jumping ahead to the many-sorted algebra.)

For example, if $R = \{Int, Bool\}$, then $w_{(Int.Int,Bool)}$ would represent a binary predicate symbol such as *equality* (=) or *less than* (<). Using a standard convention, the type of a constant may be declared by letting $\sigma = \varepsilon$. For example an integer constant would be denoted by $cons_{(\varepsilon, Int)}$ or simply $cons_{Int}$.

Types of variables may also be declared in the same way. This leads to the consideration of signatures with variables.

7.2.2 Signatures with Variables

A many-sorted signature with variables is the triple:

$$\Sigma = (R, \Omega, V)$$

where R is a set of sorts, Ω a set of operators with associated arity as before and V is a set of typed variables, known as an *R-sorted set of variables*. It is assumed that R, Ω and V are disjoint. The type of the variable is defined by

the arity function, in a similar way to that of constants, from the set of variable names to $\{\varepsilon\} \times R$. A variable in V of sort $r \in R$ would be denoted by $v_{(\varepsilon,r)}$ or more simply by v_r. For example, if $Int \in R$, then an integer variable would be $v_{(\varepsilon,Int)}$ or v_{Int}.

V may be partitioned according to sorts, where V_r denotes the set of variables of sort r (i.e. $v_a \in V_r$ iff $a = r$).

Including the variables in the signature is a convenient way of ensuring that they are appropriately typed.

7.2.3   Natural and Boolean Signatures

The term *Boolean Signature* is used to mean a many-sorted signature where one of the sorts is Boolean. Similarly, the term *Natural Signature* is used when one of the sorts corresponds to the Naturals (N).

7.2.4   Terms of a Signature with Variables

Terms of sort $r \in R$ may be built from a signature $\Sigma = (R, \Omega, V)$ in the following way. We denote a term, e, of sort r by $e : r$ and the set of terms of sort r by $TERM(\Omega \cup V)_r$, and generate them inductively as follows. For $r, r_1, \ldots, r_n \in R$ ($n > 0$)

1. $V_r \subseteq TERM(\Omega \cup V)_r$;

2. For all $w_{(\varepsilon,r)} \in \Omega$, $w_{(\varepsilon,r)} \in TERM(\Omega \cup V)_r$; and

3. If $e_1 : r_1, \ldots, e_n : r_n$ are terms and $w_{(r_1 \ldots r_n, r)} \in \Omega$, is an operator, then $w_{(r_1 \ldots r_n, r)}(e_1, \ldots, e_n) \in TERM(\Omega \cup V)_r$

Thus if *Int* is a sort, integer constants and variables, and operators (with appropriate arguments) of output sort *Int* are terms of sort *Int*.

We denote the set of all terms of a signature with variables by $TERM(\Omega \cup V)$, the set of all *closed* terms (those not containing variables, also known as *ground* terms) by $TERM(\Omega)$. Thus

$$TERM(\Omega \cup V) = \bigcup_{r \in R} TERM(\Omega \cup V)_r$$

7.2.5   Multisets of Terms

Multisets or *bags* of terms can also be built inductively from the signature if we assume that we have a Natural signature. We define multisets of terms this way to allow the multiplicities to be terms of sort *Nat*, rather than just the Naturals themselves. (This allows, for example, the introduction of conditions into arc expressions - see sections 7.3.2 and 7.8.5.)

Let $BTERM(\Omega \cup V)$ denote the set of multisets of terms, defined inductively as follows, using the symbolic sum representation for multisets defined in Appendix A. ($TERM(\Omega \cup V)$ is considered as a special set of multisets, where each member of $TERM(\Omega \cup V)$ is a multiset.)

- $TERM(\Omega \cup V) \subset BTERM(\Omega \cup V)$;

- if $b1, b2 \in BTERM(\Omega \cup V)$, then $(b1 + b2) \in BTERM(\Omega \cup V)$; and

- if $i \in TERM(\Omega \cup V)_{Nat}$ and $b \in BTERM(\Omega \cup V)$,
  then $i \times b \in BTERM(\Omega \cup V)$ where '$\times$' represents scalar multiplication.

Where there is no confusion the '$\times$' will be dropped and juxtaposition will be used for scalar multiplication (e.g. '$3 \times x$ can be replaced by $3x$ and $4 \times 3 \times x$ by $4 \times 3x$ which is distinctly different from $43x$.)

The set of bags with infinite multiplicities, $B_\infty TERM(\Omega \cup V)$, may now be defined as follows

- $BTERM(\Omega \cup V) \subset B_\infty TERM(\Omega \cup V)$; and

- if $b \in BTERM(\Omega \cup V)$, then $\infty \times b \in B_\infty TERM(\Omega \cup V)$.

where multiplication by ∞ is defined in appendix A.

7.2.6   Many-sorted Algebras

A many-sorted algebra, (or Σ-Algebra), H, provides an interpretation (meaning) for the signature Σ. For every sort, $r \in R$, there is a corresponding set, H_r, known as a *carrier* and for every operator $w_{(r_1 \ldots r_n, r)} \in \Omega$, there is a corresponding function

$$w_H : H_{r_1} \times \ldots \times H_{r_n} \to H_r.$$

In case an operator is a constant, w_r, then there is a corresponding element $w_H \in H_r$. They may be considered as functions of arity zero.

Definition: A many-sorted Algebra, H, is a pair

$$H = (R_H, \Omega_H)$$

where $R_H = \{H_r | r \in R\}$ is the set of carriers and
$\Omega_H = \{w_H | w_{\sigma,r} \in \Omega, \sigma \in R^* \text{ and } r \in R\}$ the set of corresponding functions.

For example, if $\Sigma = (\{Int, Bool\}, \{<_{(Int.Int,Bool)}\})$ then a corresponding many-sorted algebra would be

$$H = (Z, Boolean; lessthan)$$

where Z is the set of integers: $\{\ldots, -1, 0, 1, \ldots\}$
$Boolean = \{true, false\}$
and $lessthan : Z \times Z \to Boolean$ is the usual integer comparison function.

It could also be

$$\mathcal{B} = (N, Boolean; lessthan)$$

where N is the set of non-negative integers: $\{0, 1, \ldots\}$
$Boolean = \{true, false\}$
and $lessthan : N \times N \rightarrow Boolean$.

(The power of the signature is that it allows a class of algebras to be categorised.)

For signatures with variables, variables are R-sorted. In the algebra, the variable is typed by the carrier corresponding to the sort.

7.2.7 Assignment and Evaluation

Given an R-sorted algebra, H, with variables in V, an *assignment* [1] for H and V is a set of functions α, comprising an assignment function for each sort $r \in R$,

$$\alpha_r : V_r \rightarrow H_r.$$

This function may be extended to terms by considering the family of functions ass comprising

$$ass_r : TERM(\Omega \cup V)_r \rightarrow H_r$$

for each sort $r \in R$. The values are determined inductively as follows. For $\sigma \in R^* \backslash \varepsilon$, $\sigma = r_1 r_2 \ldots r_n$, with $r, r_1, \ldots, r_n \in R$ and $e, e_1, \ldots, e_n \in TERM(\Omega \cup V)$,

- If $e \in V_r$ is a variable, then $ass_r(e) = \alpha_r(e)$

- For a constant, $w_r \in \Omega$, $ass_r(w_r) = w_H \in H_r$.

- If $e = w_{(\sigma, r)}(e_1, \ldots, e_n)$, then
  $ass_r(w_{(\sigma, r)}(e_1, \ldots, e_n)) = w_H(ass_{r_1}(e_1), \ldots, ass_{r_n}(e_n)) \in H_r$, where $e_1 : r_1 \ldots e_n : r_n$.

Knowing the values of terms we can determine the value of multisets of terms by considering the multiset as a sum of scaled terms and evaluating each scalar and term for a particular assignment to variables. This is defined inductively for $a \in TERM(\Omega \cup V)$, $i \in TERM(\Omega \cup V)_{Nat}$ and $b1, b2 \in BTERM(\Omega \cup V)$ by

- $Val_H(i \times a) = ass(i) \times ass(a)$

- $Val_H(b1 + b2) = Val_H(b1) + Val_H(b2)$

7.3 P-Graphs

In this section a definition of a graphical form of P-nets is given by defining a **P-Graph**. A P-Graph consists of an inhibitor net where the arcs are annotated by multisets of terms. The multiplicities of the multisets are non-negative integer

---

[1]The terms *binding* and *valuation* are also used in this context.

terms. Transitions are annotated by Boolean terms. The terms are built from a Natural-Boolean signature which has an associated many-sorted algebra. The colour function restricted to places is included. It associates a carrier of the many-sorted algebra with a place. The capacity and initial marking are multisets over the place colour set as is the case for a P-net.

7.3.1 Definition

A P-Graph is a structure

$$\mathbf{PG} = (IN, \Sigma, C, AN, K, M_0)$$

where

- $IN = (S, T; F, IF)$ is an inhibitor net, with
    - S a finite set of places;
    - T a finite set of transitions disjoint from S;
    - $F \subseteq (S \times T) \cup (T \times S)$ a set of arcs; and
    - $IF \subseteq S \times T$ a set of inhibitor arcs.

- $\Sigma = (R, \Omega, V)$ is a Natural-Boolean signature with variables. It has a corresponding Σ-Algebra, $H = (R_H, \Omega_H)$.

- $C : S \to R_H$ is the colour function restricted to places, such that $\forall s \in S, C(s) \neq \emptyset$.

- $AN = (A, IA, TC)$ is a triple of net annotations.
    - $A : F \to BTERM(\Omega \cup V)$ such that for $C(s) = H_r$, then for all $(s, t), (u, s) \in F, A(s, t), A(u, s) \in BTERM(\Omega \cup V)_r$. It is a function that annotates arcs with a multiset of terms of the same sort as the carrier associated with the arc's place.
    - $IA : IF \to B_\infty TERM(\Omega \cup V)$ such that for $C(s) = H_r$, then for all $(s, t) \in IF, IA(s, t) \in B_\infty TERM(\Omega \cup V)_r$. It is a function that annotates inhibitor arcs with a multiset of terms of the same sort as the carrier associated with the arc's place.
    - $TC : T \to TERM(\Omega \cup V)_{Bool}$ where for all $t \in T, TC(t)$ belongs to $TERM(\Omega \cup V(t))_{Bool}$ and $V(t)$ is the set of free variables occurring in the arc inscriptions associated with t.
    TC annotates transitions with Boolean expressions.

- $K : S \to \bigcup_{s \in S} \mu_\infty^+ C(s)$ where $K(s) \in \mu_\infty^+ C(s)$ is the capacity function.

- $M_0 : S \to \bigcup_{s \in S} \mu C(s)$ such that $\forall s \in S, M_0(s) \leq K(s)$, is the initial marking.

7.3.2 Discussion

Concrete Colour Sets

In defining P-Graphs, we have intentionally associated a concrete colour set with each place. This colour set is a carrier of the chosen many-sorted algebra, H. This allows us to specify concrete systems where the sets and functions have already been determined.

There is also a need for a more abstract or syntactic form that allows classes of systems to be specified. In this case the places become R-sorted. This leads us to the notion of a P-Graph schema which is defined later in section 7.12.

Tupling

In an earlier work [22], the P-Graph definition included explicit tupling in the signature, in a similar way to PrT-nets [70]. This has the advantage that only relatively simple sorts need be included in the signature. It has the disadvantage that it makes for a more complex definition. If complex sorts (e.g. those corresponding to product sets in the algebra) are allowed in the signature, then tupling can always be done. The relationship between sorts and colour sets becomes more transparent as places may now be R-sorted with the associated carrier (in the algebra) being the corresponding colour set. This is a more elegant approach and is followed in [143,123].

Arc Annotations

When generating multisets of terms for the arc inscriptions, we allow the multiplicities to be natural number terms, so that the value can depend on the values of variables and operators of other types. In particular this includes as a special case, the *generalised Kronecker delta* extension to PrT-nets [70].

Subtyping

An early definition of the P-Graph [22] allowed there to be terms (tuples), annotating an arc, which on evaluation had to be a multiset over the colour set of the arc's place. This allows for subtyping. For the chosen algebra, there may be a number of sorts, associated with terms used in the arc's annotation, all of which have carriers which are subsets of the place's colour set. Strictly speaking, the above definition only allows for terms of the *same* sort (as the carrier associated with the arc's place). This was done to keep the definition as simple as possible.

Subtyping is easily incorporated if necessary as follows. For an algebra H, define the set R_r, which gathers together our desired sorts.

$$R_r = \{r' | r' \in R \text{ and } H_{r'} \subseteq H_r\}$$

Thus for $C(s) = H_r$, the restrictions on the terms annotating the arcs become:

$$\forall (s,t), (u,s) \in F : A(s,t), A(u,s) \in \bigcup_{r' \in R_r} BTERM(\Omega \cup V)_{r'}$$

89

and

$$\forall (s,t) \in IF : IA(s,t) \in \bigcup_{r' \in R_r} B_\infty TERM(\Omega \cup V)_{r'}$$

Strong Typing vs Weak Typing

The inclusion of the colour function, C, may be considered unnecessary. This is because the co-domain of the capacity function and initial marking function could be represented as the set of multisets of terms in $TERM(\Omega)$ evaluated in the Σ-Algebra, H. The colour set of a place would be determined by the sorts of the terms in the annotations of the surrounding arcs (evaluated in H) and the capacity and initial marking functions.

The inclusion of the colour function has a number of advantages. Firstly it encourages good design, as the typing of places needs to be considered early in the specification of a system. Secondly, it ensures that the initial marking, capacity function and arc annotations are all consistently typed. This can be used to great advantage for type checking specifications with automated tools. Finally, it allows a straightforward interpretation in terms of a P-net.

We shall use the term *strongly-typed* for P-Graphs in which the colour function is included and *weakly-typed* when it is not included.

Alternative Graphical Forms

A slightly less syntactic approach would be to replace the signature with the many-sorted algebra, a set of variables, and a typing function associating a variable with a particular carrier of the algebra. This would be closer to the approach in [123] for Algebraic Nets.

Another graphical form would be to just consider an annotated net (rather than an inhibitor net). The definition would be as before, except that the inhibitor net IN would be replaced by a net N and annotations of the input and output arcs would be separated. The ouput arcs would be annotated as before, but the input arcs would carry a pair as an inscription. The first element of the pair would refer to the pre map and the second to the inhibitor map. This may prove to be a more convenient graphical representation as less arcs are involved and it would tend to de-emphasize the rôle of the inhibitor. This is desirable when the inhibitor is acting as a way of increasing modelling convenience rather than modelling power, for example when purging places with finite capacities.

There are a number of alternative graphical forms and the choice of the most suitable form will depend on further experience in particular application domains. Present experience indicates that the above definition is at least a useful one.

7.4 Interpretation of the P-Graph as a P-net

The P-Graph may be given an interpretation as a P-net in the following way.

1. Places: S is the set of places in the P-net.

2. Transitions: T is the set of transitions in the P-net.

3. Colour Sets: The colour set for a transition is determined by the types of the variables occurring in the surrounding arc annotations restricted by its transition condition.

   Let there be n_t free variables associated with the arcs surrounding a transition $t \in T$. Let these have names $v_{r_1}(t), \ldots, v_{r_{n_t}}(t) \in V$. In the Σ-Algebra, H, for all $i \in \{1, 2, \ldots, n_t\}$, let the carrier corresponding to r_i, H_{r_i}, be denoted by G_i with typed variables $v_i(t) : G_i$. Following [87], let $g_i \in G_i$, then

   $$C(t) = \{(g_1, \ldots, g_{n_t}) \mid (\lambda(v_1(t), \ldots, v_{n_t}(t)).TC(t))(g_1, \ldots, g_{n_t})\}$$

   (The λ-expression provides a means for formally substituting values for the variables in the Transition Condition. Tuples which satisfy $TC(t)$ are included in $C(t)$.)

   The colour sets for the places are obtained from the colour function. Thus the structuring set (of colour sets) is given by $\mathcal{C} = \{C(x) \mid x \in S \cup T\}$.

4. The Colour Function: The colour function restricted to places is defined in the P-Graph and for all $t \in T$, $C(t)$ is given above.

5. Pre and Post Maps.

   The pre and post maps are given, for all $(s, t), (t, s) \in F$, by the following mappings from $C(t)$ into $\mu C(s)$

   $$Pre(s, t) = \lambda(v_1(t), \ldots, v_{n_t}(t)).A(s, t)$$

   $$Post(s, t) = \lambda(v_1(t), \ldots, v_{n_t}(t)).A(t, s)$$

   For $(s, t) \notin F$ and $\forall m \in C(t)$, $Pre(s, t; m) = \emptyset$ and for $(t, s) \notin F$ and $\forall m \in C(t)$, $Post(s, t; m) = \emptyset$.

6. Inhibitor Map

   The inhibitor map is a function from $C(t)$ into $\mu_\infty C(s)$ where for all $(s, t) \in IF$

   $$I(s, t) = \lambda(v_1(t), \ldots, v_{n_t}(t)).IA(s, t)$$

   and for $(s, t) \notin IF$, $\forall g \in C(s), m \in C(t), mult(g, I(s, t; m)) = \infty$.

7. Capacity Function.

   $K(s)$ is as defined in the P-Graph.

8. Initial Marking.

   $M_0(s)$ is as defined in the P-Graph.

With this translation from the P-Graph to P-nets in place, we may now use the definitions of marking, enabling and transition rule for P-nets to allow the P-Graph to be executed. (Alternatively, we could define the enabling condition and the transition rule directly for the P-Graph, by considering assignments for terms in a similar way to [123].)

7.5 Graphical Form of P-Graph

7.5.1 General

The graphical form comprises two parts: a *Graph* which represents the net elements graphically and carries textual inscriptions; and a *Declaration*, defining all the sets, variables, constants and functions that will be used to annotate the Graph part. The declaration may also include the initial marking, the capacity and the colour function if these cannot be inscribed on the graph part due to lack of space.

7.5.2 Places

In the usual way we shall represent places by circles (or ellipses). A place s may carry four inscriptions.

- the place name;

- the colour set associated with the place, $C(s)$;

- the place capacity, $K(s)$; and

- the initial marking, $M_0(s)$.

The first three would be inscribed close to the place, whereas the initial marking would be inscribed inside the circle representing the place. (As mentioned above, $C(s)$, $K(s)$ and $M_0(s)$ can be defined in the Declaration if there is insufficient space in the Graph part.) We shall adopt the convention that if a place $s \in S$ is not annotated by a capacity multiset, then it will have infinite capacity for all tokens in $C(s)$, unless specified otherwise in the Declaration.

Useful notation for $K(s)$ is given later in sections 7.10 and 7.11.

7.5.3 Transitions

Transitions are represented by rectangles, annotated by a name and may be inscribed by a boolean expression, known as the *Transition Condition*. The Transition Condition for transition t, $TC(t)$, only involves the variables of the inscriptions of its surrounding arcs. If a transition, t, is left blank, then the Transition Condition is true ($TC(t) = true$).

7.5.4 Arcs

As usual arcs are represented by arrows. For $(s,t) \in F$, an arrow is drawn from place s to transition t and vice versa for $(t,s) \in F$. If (s,t) and (t,s) have the same inscriptions (s is a side place of t), $A(s,t) = A(t,s)$, then this may be shown by a single arc with an arrowhead at both ends and annotated by a single inscription.

Linear P-Graph

$$
\begin{aligned}
&S = \{p1\},\ T = \{t1\},\ F = \{(p1, t1)\},\ IF = \emptyset \\
&\Sigma = (\{\mathbf{A}, Bool\}, \{true_{Bool}\}, \{x_{\mathbf{A}}\});\ H = (\{A, Boolean\}, \{true\}) \\
&C(p1) = A \\
&A(p1, t1) = x,\ IA = \emptyset,\ TC(t1) = true \\
&K(p1) = \{(a, \infty) | a \in A\} \\
&M_0(p1) = A
\end{aligned}
$$

Figure 7.1: Subset Consumption

An inhibitor arc, $(s, t) \in IF$, is represented by an edge from place s to transition t with a small circle instead of an arrow head at its destination.

The arcs will be annotated with multisets of terms. We therefore need a convenient representation for multisets. We use the symbolic sum or vector representation described in appendix A. In order to distinguish multiplicities from terms, the convention is adopted that terms may be enclosed in angular brackets.

7.5.5 Markings and Tokens

A token is a member of $\bigcup_{s \in S} C(s)$. A Marking of the net may be shown graphically by annotating a place with its multiset of tokens $M(s)$. We again use the symbolic sum representation and distinguish multiplicities from tokens, by enclosing tokens in angular brackets. Thus if $g \in M(s)$, g or $<g>$ could appear written in the circle representing place s. We use the natural numbers greater than one, to represent the multiplicity of the token in $M(s)$. Thus if $mult(g, M(s)) = m_g$ we would represent this by juxtaposition: $m_g<g>$ and this would be written inside the circle representing s. If $m_g = 1$, it would be omitted from the inscription. If g is an n-tuple (for example $g = (a, b, c)$), then we adopt the convention of dropping the parentheses (e.g. (a, b, c) would be represented by $<a, b, c>$ and not $<(a, b, c)>$.)

7.6 Simple Examples

This section provides an introduction to the graphical form of the P-Graph via some simple examples which illustrate some of the conventions adopted in the graph. An interpretation in terms of a P-net is also presented.

7.6.1 Consume any subset

Consider the following example which represents the consumption of any non-empty subset of a set of $n \in N^+$ elements, $A = \{a_1, \ldots, a_n\}$, followed by further consumption of a non-empty subset of the remainder and so on until all elements are consumed. The linear form of the P-Graph is given in figure 7.1.

P-Graph

Declarations

$$A = \{a_1, \ldots, a_n\}$$
$$M_0(p1) = A$$
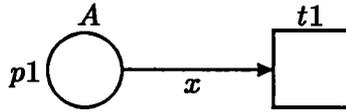
Graph

P-Net

$$S = \{p1\}$$
$$T = \{t1\}$$
$$C = \{A\}$$
$$C(p1) = C(t1) = A$$
$$\forall a \in A, Pre(t1, a) = \{(p1, a)\}$$
$$\forall a \in A, Post(t1, a) = \emptyset$$
$$K = \{((p1, a), \infty) | a \in A\}$$
$$M_0 = \{(p1, a) \mid a \in A\}$$

Figure 7.2: P-Graph and P-net corresponding to Figure 7.1

A graphical form of the P-Graph and its corresponding P-net are shown in figure 7.2.

For each value of $x : A$, there is an occurrence mode of $t1$. Consider the (multi)set $T_\mu = \{(t1, a) | a \in A\}$. Then

$$Pre'(T_\mu) = \{(p1, a) | a \in A\} = M_0$$

Thus all modes of the transition are enabled and any (non-empty) subset could occur simultaneously, consuming the corresponding subset of A.

This P-net represents a set of $|A|$ independent underlying input place/transition pairs, which are concurrently enabled.

This example illustrates a number of conventions that are adopted in the graphical form.

- Inhibitors: It is quite often the case that inhibitor arcs are not present so that $IF = \emptyset$ and hence IA is also empty.

- Omission of a capacity annotation or declaration indicates infinite capacity.

- Quite often it is not necessary to state the signature explicitly and we can operate at the level of the algebra. Thus we can just declare the sets and operators and type variables. In this case we have adopted the convention

that the type *Bool* can be considered primitive and that there is no need to explicitly declare the constant *true*. This is consistent with the use of the default transition condition as discussed below.

- Implicit typing of variables. When the colour set of a place is a simple product of carriers (or a union of products of different degree), then the type of a variable in an arc annotation is determined from its position in the tuple, the degree of the tuple and the colour set definition. (If the variable occurs in the argument of a function, then it is typed by the domain of the function.) In this example, $x : A$.

  If the variable is used in a number of arc inscriptions, then it is possible for mistakes to be made with implicit typing, so that the typing of a specific variable is inconsistent. Considerable care is required with implicit typing and ambiguity will be avoided if all variables are declared in the Declaration.

- Default Transition Condition. If for $t \in T, TC(t) = true$, t is left blank rather than annotating it with the constant *true*. This is the convention adopted for $t1$ in this example.

- Multisets as sets. We adopt the convention that when a multiset is a set (i.e. its multiplicities are chosen from $\{0, 1\}$), then it can be represented as a set. This has been followed for the initial marking and the images of the pre and post maps.

Remark: Choosing $C(t) = A$ is demanded by the above transformation (section 7.4) but this is not necessary. We could have chosen $C(t) = B$ with $|B| = |A|$ and defined *Pre* as a bijection

$$Pre : \{(t1, b)|b \in B\} \rightarrow \{(p1, a)|a \in A\}$$

Thus there is an isomorphism. We chose $C(t) = A$ as it provides the simplest way of defining the rule for *Pre*. Choosing $C(t) = B$ would be equivalent to renaming the transitions in the underlying P/T-net.

In the following examples we shall only give the graphical representation of the P-Graph.

7.6.2 Consume any token and create any token

The P-net of figure 7.3 shows an example of more complex folding, where each underlying place $\{(p1, a)|a \in A\}$ is an input to $|B|$ transitions each of which has a different place chosen from $\{(p2, b)|b \in B\}$ as an output place. The variables are implicitly typed with $x : A$ and $y : B$.

7.6.3 Information Flow

By replacing y by x in the above example, we can see how information can flow around a net. To ensure that the resulting net is a P-Graph we must have that

P-Graph

Declarations

A, B: Non-empty sets
$M_0(p1) = A$, $M_0(p2) = \emptyset$

Graph
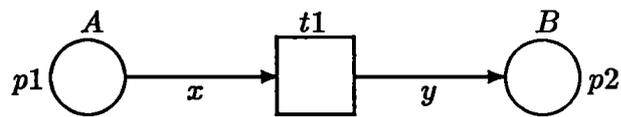
P-Net

$S = \{p1, p2\}$
$T = \{t1\}$
$C = \{A, B\}$
$C(p1) = A$
$C(p2) = B$
$C(t1) = A \times B$
$\forall a \in A, \forall b \in B, Pre(t1, a, b) = \{(p1, a)\}$
$\forall a \in A, \forall b \in B, Post(t1, a, b) = \{(p2, b)\}$
$K = \{((p1, a), \infty), ((p2, b), \infty) | a \in A, b \in B\}$
$M_0 = \{(p1, a) | a \in A\}$

Figure 7.3: More Complex Folding

P-Graph

Declarations

$\boxed{\begin{array}{l} A, B\text{: Non-empty sets} \\ A \subseteq B;\ x : A \\ M_0(p1) = A,\ M_0(p2) = \emptyset \end{array}}$

Graph

P-Net

$\boxed{\begin{array}{l} S = \{p1, p2\} \\ T = \{t1\} \\ C = \{A, B\} \\ C(p1) = A \\ C(p2) = B \\ C(t1) = A \\ \forall a \in A, Pre(t1, a) = \{(p1, a)\} \\ \forall a \in A, Post(t1, a) = \{(p2, a)\} \\ K = \{((p1, a), \infty), ((p2, b), \infty) | a \in A, b \in B\} \\ M_0 = \{(p1, a) | a \in A\} \end{array}}$

Figure 7.4: Information Flow

$A \subseteq B$ and $x : A$. In this case, including the type of x in the declaration is mandatory as implicit typing is ambiguous (is $x : A$ or $x : B$?). The corresponding P-net is shown in figure 7.4.

The underlying P/T-net is a set of $|A|$ identical input place, transition, output place subnets. There are also $|B \setminus A|$ isolated places.

7.6.4 Transition Condition

Consider the example of section 7.6.2 with the added constraint that $x < y$ is attached as a condition to transition $t1$. The P-Graph is given in figure 7.5. The comparison operator, $<$, must be defined in the Declaration. Infix notation is used when it is customary. The corresponding P-net is the same as that of figure 7.3 except that the occurrence modes are limited by the condition $x < y$ so that $C(t1) = \{(a, b) | a \in A, b \in B, \text{and } a < b\}$.

Declarations

A, B: Non-empty sets
$<: A \times B \to Boolean$
$M_0(p1) = A,\ M_0(p2) = \emptyset$

Graph
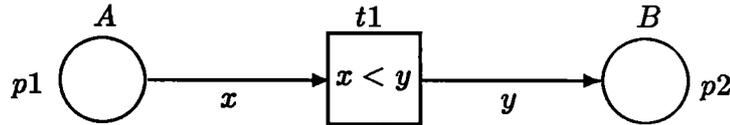
Figure 7.5: P-Graph with Transition Condition

7.7 Many-sorted Algebraic Nets

Algebraic nets were proposed as a reformulation of PrT-nets with an improved invariants calculus in [123], where a *partial* algebra over a single carrier was employed. The many-sorted nature of applications was captured by allowing the carrier to be the union of a number of sets. This then lead to the definition of partial functions and their associated operators to be used in the multiset of terms for arc inscriptions.

A colour function is not included in [123] and the net is therefore *weakly-typed*. We shall consider two *many-sorted* algebraic nets: one weakly-typed and the other strongly-typed.

7.7.1 Weakly-typed many-sorted algebraic nets

A weakly-typed many-sorted algebraic net, MAN, is one of the simplest special cases of a P-Graph, where the inhibitor arcs and annotations, the Transition Condition, and the colour and capacity functions are removed (i.e. $IF = \emptyset$; $(\forall t \in T)TC(t) = true$; all places have infinite capacity; and the colour function is not included). The arc annotations are also restricted to multisets where the multiplicities are constants rather than natural number terms.

Definition

A weakly-typed MAN is a structure

$$(N, \Sigma, A, M_0)$$

where

- $N = (S, T; F)$ is a net.

$$\boxed{\begin{array}{l} D = A \cup B \\ A = \{a_1, \ldots, a_n\}, n \in N^+ \\ B = \{b_1, \ldots, b_n\} \\ f : D \to D \text{ where} \\ f(a_i) = b_i \text{ for } i = 1, \ldots, n \\ f(b_i) \text{ is undefined for } i = 1, \ldots, n \\ M_0(p1) = B, \; M_0(p2) = \emptyset \end{array}}$$
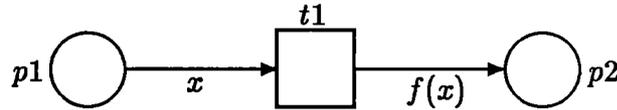
Figure 7.6: Algebraic Net with an undefined follower marking

- $\Sigma = (R, \Omega, V)$ is a an R-sorted signature with variables. It has a corresponding R-sorted algebra, D.

- $A : F \to \mu TERM(\Omega \cup V)$ is the arc annotation function.

- $M_0 : S \to \mu\{Val_D(\tau) \mid \tau \in TERM(\Omega)\}$ is the initial marking.

I believe that this net captures the spirit of Algebraic nets in terms of a specification language and it has the following advantages:

1. Functions are total.

   Because functions are partial in [123], it is possible to annotate arcs with terms that are not defined in the algebra. This leads to difficulties in interpreting the behaviour of such nets. An example of an Algebraic net illustrating the difficulty is shown in figure 7.6.

   Firstly, consider the situation when $M_0(p1) = A$. Using the terminology of Algebraic nets, a *valuation* (assignment) for x, $\beta(x) = a_1$ for example, will enable $t1$ in mode β. When $t1$ occurs in mode β, a_1 is removed from place $p1$ and $f(a_1) = b_1$ is added to $p2$, i.e. $M(p1) = A \setminus \{a_1\}$ and $M(p2) = \{b_1\}$. A similar situation occurs for any valuation, $\beta(x) \in A$. Any valuation, $\beta(x) \in B$, will not enable $t1$, due to the initial marking of $p1$.

   Now consider when $M_0(p1) = B$, (a perfectly legal initial marking as $M_0 : S \to \mu D$, where $D = A \cup B$). A valuation, $\beta(x) \in B$, will now enable $t1$. When $t1$ occurs in mode b_1, the follower marking for $p1$ is clear, $M_0(p1) = B \setminus \{b_1\}$, but the follower marking for $p2$ is undefined as the value $f(b_1)$ is not defined.

   This problem does not occur with many-sorted algebraic nets as defined above because functions are total. The intention of the designer of the above algebraic net is unclear. A possible interpretation would be that transition, $t1$, is only enabled when x is bound to an element of A. This

99

Declarations

Sorts: $R = \{r1, r2\}$
Carriers: $D_{r1} = A$ and $D_{r2} = B$
Operators: All constants from A and B
and unary operator $f = w_{(r1,r2)}$
Variable: $x = v_{r1}$ thus $x : A$
$A = \{a_1, \ldots, a_n\}, n \in N^+$
$B = \{b_1, \ldots, b_n\}$
$f_D : A \to B$ where
$f_D(a_i) = b_i$ for $i = 1, \ldots, n$
$M_0(p1) = A \cup B$, $M_0(p2) = \emptyset$

Graph
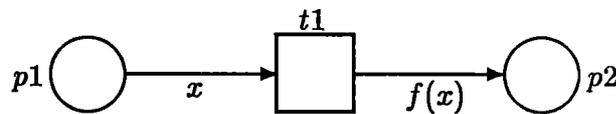
Figure 7.7: Weakly-typed MAN interpretation of above Algebraic Net

interpretation is easily handled with a many-sorted algebraic net (MAN) (see figure 7.7).

The MAN has essentially the same graphical form. The graph part and initial marking are identical. The main difference is that a signature with variables is explicitly included. The sorts $R = \{r1, r2\}$ have corresponding carriers $D_{r1} = A$ and $D_{r2} = B$. The set of operators includes a unary operator $f = w_{(r1,r2)}$ and enough constants of type $A \cup B$ to define the initial marking. The set of variables, V, is a singleton $x = v_{r1}$ and thus $x : A$. The function corresponding to the operator f is a bijection $f_D : A \to B$, where for $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$, $f_D(a_i) = b_i$ for $i = 1, \ldots, n$. To make the example more interesting we have set the initial marking to $M_0(p1) = A \cup B$ and $M_0(p2) = \emptyset$.

Transition, $t1$, is enabled in all modes, $m \in A$, and once all the $a's$ in $p1$ have been transformed into $b's$ in $p2$, $t1$ is dead. There is no possibility of binding x to an element of B, as it is of type $x : A$ as defined in the signature. Thus there are no difficulties of interpretation.

2. Sets can be simple.

The sets of the many-sorted algebra are simple (as opposed to complex unions of other component sets) and correspond to the sets of the physical world that is being modelled. This contrasts with Algebraic nets where there is only one carrier which needs to contain the union of all the simple sets. This is more than an aesthetic problem when developing automated tools, as valuations for each variable will be over the rather large set D,

instead of a much smaller domain corresponding to a carrier of the many-sorted algebra.

7.7.2 Strongly-typed many-sorted algebraic nets

Definition

A strongly-typed many-sorted algebraic net, includes a colour function and is given by

$$(N, \Sigma, C, A, M_0)$$

where

- $N = (S, T; F)$ is a net.

- $\Sigma = (R, \Omega, V)$ is an R-sorted signature with variables. It has a corresponding R-sorted algebra, $H = (R_H, \Omega_H)$.

- $C : S \to R_H$ is the colour function restricted to places.

- $A : F \to \mu TERM(\Omega \cup V)$ is the arc annotation function, where for $C(s) = H_r$, and for all $(s, t), (u, s) \in F$, $A(s, t), A(u, s) \in \mu TERM(\Omega \cup V)_r$.

- $M_0 : S \to \bigcup_{s \in S} \mu C(s)$ such that $\forall s \in S$, $M_0(s) \in \mu C(s)$ is the initial marking.

The strongly-typed many-sorted algebraic net has the advantage that static type checking can be done to eliminate errors as discussed before. In the above example, it may have been that place $p1$ should never be marked with tokens from B and that the initial marking was just a mistake. In this case it would be appropriate to set $C(p1) = A$ and, depending on the application, $C(p2) = B$. In this case, setting $M_0(p1) = B$, would violate the typing rules and be detected in a static check. This would not be the case in a weakly-typed MAN, where the error would be detected at run time when an attempt to execute the net would reveal that $t1$ was dead.

The P-Graphs of figures 7.2, 7.3 and 7.4 are examples of strongly-typed MANs, but figure 7.5 is not a strongly-typed MAN as it has a transition condition different from *true*. In the next section we define CP-Graphs which allow for transition conditions. It will be seen that a strongly-typed *MAN* is a special class of CP-Graph where $\forall t \in T, TC(t) = true$ and the multiplicities of terms in arc expressions are natural numbers rather than natural number terms.

7.8 CP-Graphs and Many-sorted PrT-Nets

7.8.1 CP-Graphs

On removing the inhibitor arcs and the place capacities from the P-Graph, we obtain a subclass that is very similar to Jensen's 'CP-graph' [87]. We shall

distinguish our class, called CP-Graphs, from that of Jensen by using an upper case 'G' in 'Graph'. The CP-**graph** differs from the CP-Graph defined here in two respects:

- it is a multigraph (i.e. multiple arcs are allowed between places and transitions); and

- the arc inscriptions and transition conditions ('guards') are not explicitly defined.

Jensen [87] states that the expressions and guards may be defined by means of a many-sorted algebra (but excludes this from his scope of concern) and that has provided part of the stimulus for the definition of P-Graphs. For CP-Graphs to be a subclass of P-Graphs they include a signature rather than the algebra. This seems to be the simplest approach, since a signature is always required to build terms and it can also be used to type variables.

Definition

A CP-Graph (**CPG**) is a P-Graph

$$(IN, \Sigma, C, AN, K, M_0)$$

with the following restrictions

- $IN = (S, T; F, \emptyset)$ i.e. no inhibitor arcs.

- $AN = (A, \emptyset, TC)$ i.e. no inhibitor arc annotations.

- For all $s \in S$, $K(s) = \{(g, \infty) | g \in C(s)\}$ i.e. the capacities of the places are infinite.

7.8.2 Many-sorted PrT-nets

Predicate/Transition Nets (PrT-nets) have been developed over the last decade with the latest definition appearing in [70]. PrT-nets are defined on a syntactic level accompanied by a *relational structure* that provides an interpretation at the concrete level of sets, functions and relations. In this section we shall consider a many-sorted PrT-net as a form of P-Graph, and return to PrT-nets defined at the syntactic level in a later section.

In [70], Genrich mentions the use of many-sorted structures and a 'formalism for abstract data types' (many-sorted algebras) but does not pursue these ideas. PrT-nets are single-sorted, do not include the inhibitor extension nor the colour function, all variables range over a single carrier, and in [70] a capacity function is not defined. A predicate associated with a place has a fixed index, so terms annotating arcs associated with the place can only be multisets of tuples of the same length as the index.

In the following we define a subclass of the CP-Graph, known as a many-sorted PrT-net which includes the colour function (i.e. it is strongly typed), and types

variables via the signature, but retains the PrT-net flavour of restricting the colour sets to products. Hence tuple lengths for terms annotating arcs are constant.

Definition

A many-sorted PrT-net (MPrT-net) is a CP-Graph with the restriction that for all $s \in S$, $C(s)$ is a simple set or a cartesian product of simple sets, where a simple set comprises elements that are singletons. This ensures that the tuples annotating arcs are of the same length.

The advantages of many-sorted PrT-nets over the PrT-nets of [70] are the same as those for many-sorted algebraic nets over Algebraic nets (see section 7.7.1). These problems may be overcome with PrT-nets by typing variables in transition conditions, but as this is an **option** of the specifier, mistakes can easily arise. An example is the resource management scheme represented as a PrT-net in [70], page 217 (the variables m, r, and $r1$ to rL may be bound to any value from D, but from the text it is clear that this is not intended). The need to type variables in the transition condition may unduly clutter the graph with information that is best handled in a declaration to allow for static checks.

7.8.3 Simple Examples

The four examples of section 7.6 are MPrT-nets.

7.8.4 Train Example

In [70], Genrich describes the operation of two trains travelling in the same direction on a circular track of seven sections. For safe ,operation, the trains must never be on the same section or even on adjacent sections. A MPrT-net is given in figure 7.8 where any number of sections greater than 4 is allowed.

The model is a little different from that in [70]. Apart from the minor difference of generalising the number of track sections, the marking of place $p2$ represents which track sections are vacant. In the original model, the same place represented the predicate that sections i and i$\oplus$1 were vacant. As a minor modelling point, the simpler meaning for a place is preferred. The less intuitive predicate also necessitates the definition of two functions, the modulo 7 successor and predecessor functions, whereas only one (modulo n addition) is required in the MPrT-net. There is also no need for the transition condition and extra variables.

The drawback of the PrT-net is that the successor functions are partial, whereas the variables all range over I $\cup$ T. Thus there are legal substitutions for the variables for which the transition condition is undefined. This situation does not arise with the MPrT-net.

It can be seen that this net is also a strongly-typed MAN.

Declarations

Set of Trains:$T = \{a,b\}$
Set of track sections:$I = \{0, 1, \ldots, n - 1 \mid n > 4\}$
n: number of sections
Variables x:T; i:I
Function $\oplus$:$I \times I \rightarrow I$ is modulo n addition
Place $p1$: Sections occupied by trains
Place $p2$: Vacant sections
$M_0(p1) = \{<0,\text{a}>, <2,\text{b}>\}$
$M_0(p2) = I \setminus \{0, 2\}$

Graph
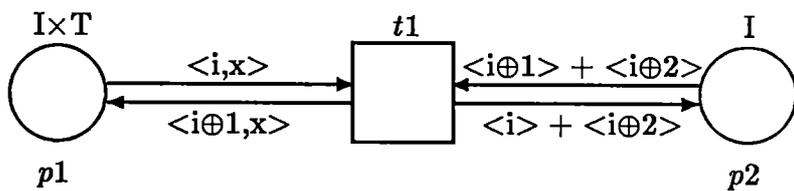
Figure 7.8: MPrT-Net of Safe Train Operation

7.8.5 Example of Conditionals in arc expressions

In this example we use a variant of the readers/writers problem to illustrate the use of conditionals in arc expressions. It is essentially the same as the resource management scheme example of [70], but the model is considerably simplified by removing unnecessary states and colours. The identities of the agents wishing to access the common resource have been retained, but the access 'tickets' are not distinguished.

A number (N) of agents (processes) wish to access a shared resource (such as a file). Access can be in one of two modes: shared (s), where up to L agents may have access at the same time (e.g. reading) and exclusive (e), where only one agent may have access (e.g. writing). No assumptions are made regarding scheduling. An MPrT-net model is given in figure 7.9.

It has been assumed that the initial state is when all the agents are idle or waiting to gain access to the shared resource (with no queueing discipline assumed). Place *Wait* is marked with all agents; *Access* is empty and the *Control* place contains L ordinary tokens. An agent can obtain access in one of two modes: if shared (m=s), then a single token is removed from *Control* (as m=e is false) when *enter* occurs in a single mode; if exclusive (m=e), then all L tokens are removed preventing further access until the resource is released (transition *Leave*). Shared access is limited to a maximum of L agents as transition *enter* is disabled when *Control* is empty.

Declarations

Set of Agents:$A = \{a_1, \ldots, a_N\}$
Set of Access Modes:$M = \{s,e\}$
Control: $C = \{\bullet\}$
Positive integer constants: N,L
Variables x:A ; m:M
Function [ ]:*Bool* $\rightarrow \{0,1\}$ where
$[true] = 1$ and $[false] = 0$
$M_0(\text{Wait}) = A$
$M_0(\text{Control}) = L<\bullet>$
$M_0(\text{Access}) = \emptyset$

Graph
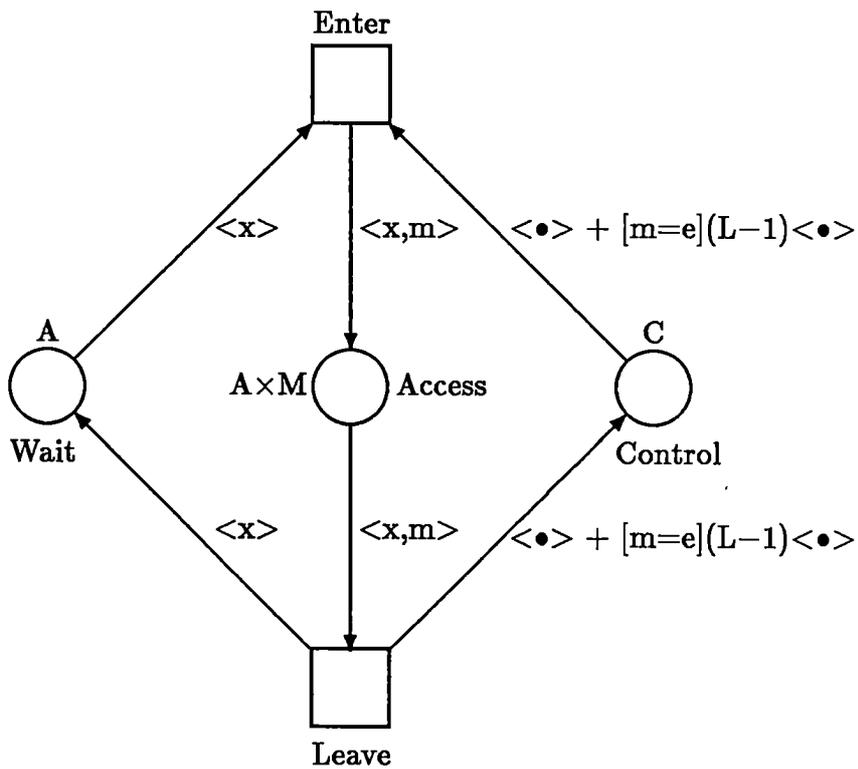
Figure 7.9: MPrT-Net of Resource Management

Following [70] outfix notation has been used for the function $Bool \rightarrow \{0, 1\}$ and this will be used as a standard convention. It is assumed that integer addition and subtraction and the equality predicate are primitive and do not need to be defined in the Declaration.

Remark: The net of figure 7.9 is very like a PrT-net. If the places were annotated by predicates rather than colour sets and the domain was formed as the union $D = A \cup M \cup C$, with variables x and m of type D, it would be a PrT-net. The indices of the predicates annotating *Wait* and *Control* would each be one, and that of *Access*, two. It is important to note that the behaviour of the two nets is not the same. In the PrT-net m can be bound to any element of D. Hence an agent could gain access to the resource in a meaningless mode • or a_i for example. The meaning of this is unclear and contrary to the intention of the specification.

7.9 P-Graph Example: Genrich's Train revisited

The train example above provides us with a very simple illustration of the use of the inhibitor arc to provide a more compact, and I think more intuitive, model of the trains travelling on a circular track. Given that this is the first example of the use of the inhibitor and capacity extensions, we shall describe it in full detail.

7.9.1 Linear P-Graph

The linear P-Graph for the safe train is given in figure 7.10.

In this example we have explicitly shown how tupling (in this case pairing) can be achieved with a suitable tupling operator declared in the signature.

7.9.2 Graphical Form

The graphical form of the P-Graph for the operation of the train is given in figure 7.11. As usual we include only the information about the algebra in the Declaration and type variables with the appropriate carrier. The tupling operator and function are considered primitive without any need to define them each time in a Declaration. I have also been less formal with the use of operator names and functions in not distinguishing between them (i.e. $\oplus$ has been used as an operator and also as a function). Also infix notation has been used as it is customary.

For inhibitor arcs we use the convention that zero multiplicities are shown explicitly, whereas infinite multiplicities are assumed for any term that is not shown explicitly (c.f. pre map arcs which assume that zero multiplicities are not shown in the sum). We have also used '*' notation to represent sums of tuples. It is defined as follows:

Linear P-Graph

$S = \{p1\}$, $T = \{t1\}$, $F = \{(p1, t1), (t1, p1)\}$, $IF = \{(p1, t1)\}$

$R = \{r1, r2, r3, Nat, Bool\}$

$\Omega = \{\oplus_{r1r1,r1}, (\_, \_)_{r1r2,r3}\} \cup Natconst \cup \{a_{r2}, b_{r2}, true_{Bool}\}$

where $Natconst$ is the set of natural constants including infinity

$V = \{i_{r1}, x_{r2}\}$

$H = (R_H, \Omega_H)$; $R_H = \{H_{r1}, H_{r2}, H_{r3}, H_{Nat}, H_{Bool}\}$

$H_{r1} = I = \{0, 1, \ldots, n-1 \mid n > 4\}, n \in N$

$H_{r2} = T = \{a, b\}$

$H_{r3} = I \times T$

$H_{Nat} = N_\infty$; $H_{Bool} = \{true, false\}$

$\Omega_H = \{\oplus_H, (\_, \_)_H, a_H, b_H, true_H\}$

$a_H = a; b_H = b; true_H = true$

$\oplus_H : I \times I \to I$ is modulo n addition

$(\_, \_)_H : I \times T \to I \times T$ is a pairing function where

$\forall j \in I, \forall t \in T, (j, t)_H = (j, t)$

$C(p1) = I \times T$

$A(p1, t1) = (i, x)$, $A(t1, p1) = (i \oplus 1, x)$

$IA(p1, t1) = 0 \sum_{j=1}^{2} \sum_{u \in U} (i \oplus j, u) + \infty \sum_{j \in J} \sum_{u \in U} (i \oplus j, u)$

where $J = \{0, 3, 4, \ldots, n-1\}$ and $U = \{a, b\}$

$TC(t1) = true$

$K(p1) = \{((j, u), 1) \mid (j, u) \in I \times T\}$ (i.e. the set $I \times T$)

$M_0(p1) = \{(0, a), (2, b)\}$

Figure 7.10: Linear P-Graph of Safe Train Operation

Declarations

Set of Trains:$T = \{a, b\}$

Set of track sections:$I = \{0, 1, \ldots, n-1 \mid n > 4\}$

$n \in N$: number of sections

Variables x:T; i:I

Function $\oplus$:I$\times$I$\to$I is modulo n addition

Place $p1$: Sections occupied by trains

$K(p1) = $ I$\times$T

$M_0(p1) = \{<0,a>,<2,b>\}$

Graph
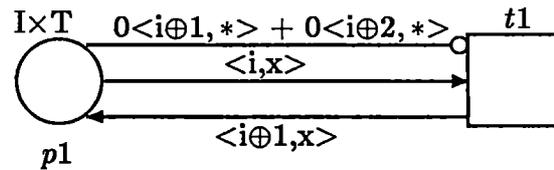
Figure 7.11: P-Graph of Safe Train Operation

$S = \{p1\}, T = \{t1\}$
$C = \{I \times T\}$
$C(p1) = C(t1) = I \times T$
$Pre(p1,t1) = \lambda(i,x).(i,x)$, the identity function on $I \times T$
$Post(t1,p1) = \lambda(i,x).(i \oplus 1, x)$, a permutation of $I \times T$
$I(p1,t1) : I \times T \rightarrow \mu(I \times T)$
$I(p1,t1) = \lambda(i,x).(0 \sum_{j=1}^{2} \sum_{u \in U} (i \oplus j, u) + \infty \sum_{j \in J} \sum_{u \in T} (i \oplus j, u))$
where $J = \{0,3,4,\ldots,n-1\}$
$K(p1) = I \times T$
$M_0(p1) = \{(0,a),(2,b)\}$

Figure 7.12: P-net of Safe Train Operation

Let $(x,y) : A \times B$, then $(x,*) = \sum_{b \in B}(x,b)$.

This can be generalised to tuples of any length, by allowing the sum to be over the domains of all the variables replaced by stars.

The graphical form provides an intuitively appealing specification of the behaviour of the trains on the track. The occurrence of $t1$ again indicates the movement of a train from section i to section i$\oplus$1. This is possible if there is a train on section i, (pre condition) and there are no trains on sections i$\oplus$1 and i$\oplus$2 (inhibitor condition). Of course the concurrent moving of trains is allowed, so long as the conditions are met for different trains on different sections of track. For example, on a 10 section track ($n = 10$) if train 'a' is on section 4 and train 'b' on section 9, then the bindings of i=4 and x=a and i=9 and x=b, both satisfy the enabling condition when taken together.

7.9.3 Equivalent P-net

The equivalent P-net is given in figure 7.12.

The P-net maps can be rewritten in function notation as
$Pre(p1,t1) = \{((i,x),(i,x))|(i,x) \in I \times T\}$
$Post(p1,t1) = \{((i,x),(i \oplus 1, x))|(i,x) \in I \times T\}$
$I(p1,t1) = \{((i,x),\{((i\oplus1,u),0),((i\oplus2,u),0),((i\oplus j,u),\infty)|u \in T, j \in J\}|(i,x) \in I \times T\}$
so that for all $(i,x) \in I \times T$
$Pre(t1,i,x) = (p1,(i,x))$ and so forth.

7.10 Notation for Capacity

The capacity of a particular place, s, is given by the function

$$K(s) : C(s) \longrightarrow N_{\infty}^{+}$$

Declarations

> H: Set of Host Addresses
> Me: Set of Host Messages
> Variables: $s, d : H; m : Me$
> $M_0(\text{BUFFER}) = \emptyset$
> $M_0(\text{FREE}) = H$
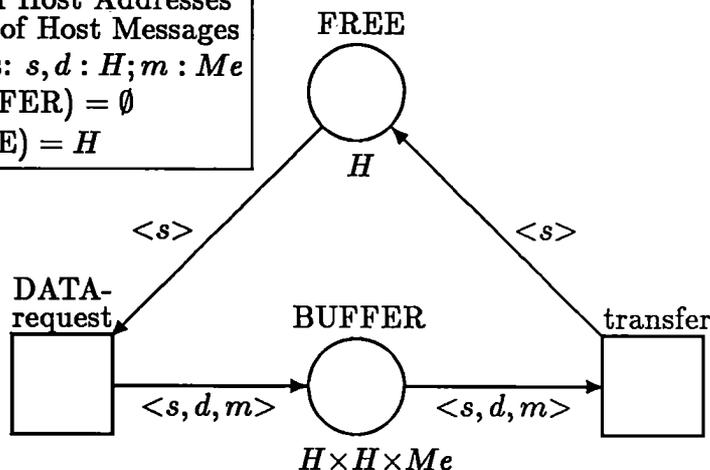
Figure 7.13: LAN Access Buffer

It is convenient to use a shorthand notation for this function when annotating places of the P-Graph, as the place is indicated by the proximity of the annotation to the place. Thus for the capacity of token $g_1 \in C(s)$, we may write (for $n_1 \in N_\infty^+$) $K(g_1) = n_1$ next to place s, instead of $K(s; g_1) = n_1$. Of course, this will only be practical when $C(s)$ is a very small set, or when most of the capacities are the same.

A special case is when the capacity for each token $g \in C(s)$ is the same, say $n \in N^+$. This is the same as the capacity defined for PrT-nets [71], and we use the same notation. Thus if place s is annotated by $K = n$ in the P-Graph, then this means $\forall g \in C(s), K(s; g) = n$.

7.11 Extended Capacity Notation

Although the P-net capacity function and the above notation may be of use in some applications, for others a much richer capacity notation is required. It is often the case that a limit needs to be placed on the cardinality of multisets over (elements of partitions of) a place's colour set. For example, we would like to be able to express constraints like $|M(s)| \leq n$. This represents the *total* capacity of a place (i.e. the sum of all tokens in the place) which could be a resource bound, e.g. a buffer capacity. Here we are not placing a direct limit on the multiplicity of each element of the colour set but a limit on the sum of multiplicities of elements and thus the capacity function (by itself) is inadequate.

As a further illustration, consider the following example encountered while modelling the M-Access Service of the Cambridge Fast Ring [26] (see chapter 10).

Figure 7.14: LAN Access Buffer illustrating extended capacity notation

A network interconnects a set of computers, known as hosts. Hosts can send messages to each other via the network. Each host has an address. When a host wishes to send a message it appends its own address (source address) and that of the destination (destination address) to the message to form a packet. Each host accesses the network via a one packet buffer. When this buffer is free, the host can store a new packet in the buffer. When network resources are available the packet is transferred into the network for routing and delivery, thus freeing-up the buffer for a new packet.

A P-Graph of the access procedure (for all hosts) is shown in figure 7.13. Place BUFFER represents the set of access buffers, one for each host. Place FREE indicates which buffers are available. (Initially all the buffers are free: M_0(BUFFER) = $\emptyset$.) If this place contains a token with the value of host a's address, then host a's buffer is free and can be used for the next packet host a wishes to submit to the network (transition DATA-request occurs). Host a's buffer will not be free again until the network accepts the packet (transition transfer occurs). Hence place FREE provides the control necessary to ensure a capacity limit of one buffer per host.

When visualization of this control mechanism is not required, we would like to replace the capacity control for place BUFFER by an extended capacity inscription. This is shown in figure 7.14, where place BUFFER is inscribed by '$K(s, *, *) = 1$'. We may interpret this to mean that there is one buffer available for each host, i.e. that the sum of tokens over the set of (destination) host addresses, H, and messages, Me, in place BUFFER for a particular value of s, is at most one. The $*$'s indicate sums over the domains of the variables they replace. This may be viewed as an extended capacity condition on the marking of the place concerned: for all markings of BUFFER, and for each host, $j \in H$, $\sum_{h \in H} \sum_{g \in Me} M(\text{BUFFER}; j, h, g) \leq 1$.

More generally, a place, s, with $C(s) = G_1 \times \ldots \times G_n$, may be annotated by an inscription $K(a_1, \ldots, a_n) = k$ with $k \in N^+$. The syntax of $a_i, i \in In = \{1, 2, \ldots, n\}$ is given by the production rule $a_i ::= < v_i > | *$ where angular brackets denote non-terminals and $v_i : G_i$. (The syntax for variables is left open, but it would normally be a finite string of alphanumeric characters.)

We shall now give the meaning of this notation in terms of a P-Graph without it.

7.11.1 Interpretation of Extended Capacity Notation

When there are no stars present in the argument of $K(a_1, \ldots, a_n)$, it has the same meaning as K, defined in the previous section. This notation is therefore redundant and would not be used.

We now consider two cases:

- A. when there is at least one star but less than n stars

- B. when all arguments are stars.

For case A, for each place, s, inscribed by $K(a_1, \ldots, a_n) = k$, we remove the inscription and replace it by a *projected* complementary place, $\overline{s}$, and associated arcs in the following manner.

1. From the argument of K create a tuple consisting of only the variables by deleting the stars. This will be of the form $< v_i, \ldots, v_j >$ with $i \leq j \leq n$.

2. Create a place, $\overline{s}$, with colour set $C(\overline{s}) = G_i \times \ldots \times G_j$ derived from the types the variables of the above tuple, where G_i is the type of the variable v_i and so forth.

3. Create an arc $(\overline{s}, t)$ for each arc (t, s), $t \in T$ and an arc $(t', \overline{s})$ for each arc (s, t'), $t' \in T$.

4. Annotate each arc by the tuple $< v_i, \ldots, v_j >$.

5. The initial Marking, $M_0(\overline{s})$ is related to $M_0(s)$ and the value of k in the following way. For every $g_i \in G_i \ldots g_j \in G_j$

$$mult((g_i, \ldots, g_j), M_0(\overline{s})) + \sum mult((g_1, \ldots, g_n), M_0(s)) = k$$

where the sum is over the domains of the variables that have been replaced by stars in the argument of K.

For case B, for each place, s, inscribed by $K(*, \ldots, *) = k$, we remove the inscription and replace it by a *completely-projected* complementary place, $\overline{s}$, (a P/T-net place) and associated arcs in the following manner.

1. Create a place, $\overline{s}$, with colour set $C(\overline{s}) = \{\bullet\}$.

2. Create an arc $(\bar{s}, t)$ for each arc (t, s), $t \in T$ and an arc $(t', \bar{s})$ for each arc (s, t'), $t' \in T$.

3. Annotate each arc by the singleton $< \bullet >$.

4. The initial Marking, $M_0(\bar{s})$ is related to $M_0(s)$ and the value of k in the following way.

$$mult((\bullet), M_0(\bar{s})) + \sum mult((g_1, \ldots, g_n), M_0(s)) = k$$

where the sum is over the domains of all the variables.

Case B corresponds to a resource limit and the notation K^* will be adopted for it (i.e. $K^* = K(*, \ldots, *)$) as in Numerical Petri Nets [145].

In this section the colour sets have been restricted to a single product set. No attempt is made to generalise to unions of product sets as the complexity and infrequent usage do not justify it.

7.12 Abstract P-Graphs or P-Graph Schemas

The P-Graph defined previously included concrete colour sets, markings and capacities. This is often the level at which telecommunication and other systems are specified. However it is very useful to have a more abstract specification that allows classes of systems to be specified. For example the range of sequence numbers or window sizes in protocols may be left open. The hope is that it will be possible to prove properties about systems for a whole range of parameter values by just considering the more abstract specification.

This is the approach adopted by Vautherin [143] where he defines a Petri net-like schema, Σ-schema, and provides an interpretation for it with a class of CP-nets. Vautherin does not include capacity or inhibitor functions, only allows equations to be associated with transitions, does not allow conditionals in arc expressions and does not type variables in his definition (although he does in examples). The following defines a schema addressing these points. The term *Abstract P-Graph* used for the schema, was suggested by Jensen in [90].

7.12.1 Definition

An **Abstract P-Graph, (APG)** or P-Graph Schema is a structure

$$(IN, \Sigma, \tau, AN, \mathcal{K}, \mathcal{M}_0)$$

where

- $IN = (S, T; F, IF)$ is an inhibitor net.

- $\Sigma = (R, \Omega, V)$ is a Natural-Boolean signature with variables.

- $\tau : S \to R$ is a function that types places. (Places are R-sorted.)

- $AN = (A, IA, TC)$ is a triple of net annotations.

  - $A : F \rightarrow BTERM(\Omega \cup V)$ such that for $s \in S$, $(s, y), (x, s) \in F$, $A(s, y), A(x, s) \in BTERM(\Omega \cup V)_{\tau(s)}$. Arcs are annotated with a multiset of terms that are of the same sort as the associated place.

  - $IA : IF \rightarrow B_\infty TERM(\Omega \cup V)$ where for each $(s, t) \in IF$, $IA(s, t)$ belongs to $B_\infty TERM(\Omega \cup V)_{\tau(s)}$. Inhibitor arcs are annotated with a multiset of terms that are of the same sort as the associated place.

  - $TC : T \rightarrow TERM(\Omega \cup V)_{Bool}$ where for each $t \in T$, $TC(t)$ belongs to $TERM(\Omega \cup V(t))_{Bool}$ with $V(t)$ the set of variables occurring in the arc inscriptions associated with t. TC annotates transitions with Boolean expressions.

- $K : S \rightarrow \mu_\infty^+ TERM(\Omega)$ where $\forall s \in S$, $K(s) \in \mu_\infty^+ TERM(\Omega)_{\tau(s)}$ is the capacity function associating a multiset of closed terms with each place.

- $\mathcal{M}_0 : S \rightarrow \mu TERM(\Omega)$ such that $\forall s \in S$, $\mathcal{M}_0(s) \leq K(s)$, is the initial marking at a syntactic level which respects the capacity.

7.12.2 Discussion

The definition mirrors that of the P-Graph, where the colour function is replaced by the typing function and the capacity and initial markings are defined at the syntactic level of terms rather than at the concrete level of sets. We may also include subtyping as has been discussed for the P-Graph (see section 7.3.2).

7.12.3 Interpretation as a P-net

For a many-sorted algebra, H, satisfying the R-sorted signature, (i.e. $H = (R_H, \Omega_H)$) the interpretation of the abstract P-Graph as a P-net is given in section 7.4, with the following exceptions.

1. Place Colour Sets. For each place $s \in S$, $C(s) = H_{\tau(s)}$.

2. Capacity Function. For all $s \in S$, $K(s) = Val_H(K(s))$.

3. Initial Marking. For all $s \in S$, $M_0(s) = Val_H(\mathcal{M}_0(s))$.

7.12.4 Abstract CP-Graphs

Like the CP-Graph, the Abstract CP-Graph does not include inhibitor arcs and has infinite capacities for places.

Definition

An Abstract CP-Graph (**ACPG**), or CP-Graph schema is a P-Graph schema with the following restrictions

Declarations

$$R = \{item, queue\}$$
$$\Omega = \{empty, enq, deq\}$$
$$empty :\rightarrow queue$$
$$enq, deq : item \times queue \rightarrow queue$$
Variables $x : item; q : queue$
$$\tau(Queue) = queue$$
$$M_0(Queue) = \{empty\}$$

Graph Schema
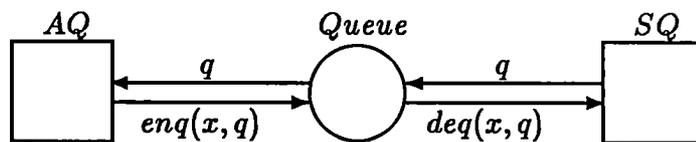
Figure 7.15: Generic Queue Specification with a CP-Graph Schema

- $IN = (S, T; F, \emptyset)$ i.e. no inhibitor arcs.

- $AN = (A, \emptyset, TC)$ i.e. no inhibitor arc annotations.

- $\forall s \in S$, $K(s) = \{(term, \infty)|term \in TERM(\Omega)_{\tau(s)}\}$ i.e. the capacities of the places are infinite.

We may also have classes such as a many-sorted PrT-net schema or an algebraic net schema by restricting the above structure in a similar way to that explored for P-Graphs.

7.12.5 CP-Graph Schema Example: A Generic Queue

Queues with different service disciplines (e.g. first-in-first-out (FIFO), last-in-first-out (LIFO), arbitrary) are important components of computer and communication systems. There may be times in an early part of a design when the service discipline and the items to be queued are undecided. It is at this stage when a class of queues can be specified by using a P-Graph Schema.

A CP-Graph schema of a generic queue is shown in figure 7.15. There are two sorts: *item*: the name of the set of items to be queued and serviced; and *queue*: the name of the queue structure (such as strings of items, or sets of items). We define a constant *empty* and enqueueing and dequeueing operations which compose items and queues to form queues. Variables are typed (x of sort *item* and q of sort *queue*) as is the place *Queue*. The capacity of place *Queue* is infinite and it is marked by the constant *empty*.

In the graph, the transition AQ models the arrival of items and SQ their servicing. On the arrival of an item, the current queue is removed, the item is added, and the new queue placed in *Queue*. If there is an item in the queue, then on servicing, this item is removed. The way in which items are added to and removed from the queue is not specified. This is the role of the algebra. For an arbitrary queueing discipline, the enqueueing and dequeueing operations correspond to multiset addition and (as we shall see later in chapter 9) for FIFO and LIFO queues they correspond to concatenation.

7.13 Conclusions

A graphical form of P-nets, P-Graph, has been defined as an inhibitor net that includes a many-sorted signature. Variables can now be appropriately typed and functions are total removing any difficulties in interpretation that can arise with single-sorted high-level nets.

A hierarchy of high-level nets can be defined by restricting the structure of the P-Graph to include CP-Graphs, many-sorted PrT-nets and many-sorted Algebraic nets. If partial functions are allowed, then PrT-nets and Algebraic nets can also be included in the hierarchy, but there appears to be little motivation to do so.

The abstract P-Graph provides a vehicle for the specification of classes of systems and the possibility of their analysis via a single member of the class as has been demonstrated by Vautherin [143]. This opens up exciting possibilities which need to be investigated in various application domains.

Chapter 8

Resetting Markings

When modelling applications it is convenient to be able to manipulate markings atomically (i.e. with the occurrence of a single mode of a transition). The purging of a queue [29] or aborting a broadcast [26] (see chapter 10) are relevant examples.

The manipulation of the marking of a place can be achieved with P-nets by setting the pre map equal to the inhibitor map and hence equal to the current marking when the transition is enabled. The post map then allows the desired manipulation of the marking as it can be defined in terms of the pre map (i.e. the current marking).

The idea of removing a place's marking on the single occurrence of a transition has arisen in the context of P/T-nets [5], where reset nets were defined and later by Valk [136], who defined a more general class of nets known as self-modifying nets. The work of this section investigates the reset idea in the context of high-level inhibitor nets, making it more expressive and suitable for complex applications.

The term *reset* will be used to connote the emptying of a place of all its tokens on the occurrence of a single transition mode. A *complete reset* implies the resetting of *all* places on the occurrence of a single transition mode.

Firstly we investigate P-nets with the *complete reset* property, the class of P-nets where the pre map and inhibitor map are the same. This class of P-net turns out not to be suitable for our purposes, but it is the simplest case conceptually and hence it is dealt with first. Having dispensed with this curiosity, we consider P-nets with the *reset* property where at least one place may be emptied by the occurrence of a single transition mode. The term, *purging* a place, is introduced, to describe the resetting of a place irrespective of its marking. A graphical representation of *purging* is given where a *reset arc* is defined as the superposition of the inhibitor and normal arcs. It is inscribed by a variable typed by the set of multisets over the colour set of the input place. The idea is generalised to purging submultisets (subbags) of markings and further to considering partitions and subsets of partitions of the input place's colour set.

8.1 Completely resetting P-nets - a curiosity

Definition: A *completely resetting* P-net is a P-net with $Pre = I$ and for all $tr \in TRANS$, $pre(tr) \neq \emptyset$.

The possibility of having a transition mode with zero-testing inhibitors for all places is excluded. This is because it leads to either the transition always being dead or, if enabled by the empty marking, then the number of times it is self-concurrently enabled is unbounded. Further, an occurrence of the transition mode leaves the marking empty. It appears that neither situation is of practical value and the restriction leads to a simple statement for the following proposition.

Proposition 8.1 *A completely resetting P-net is an interleaving model in that no two transition modes can be concurrently enabled.*

Proof: The proof is by contradiction. Assume that $tr1, tr2 \in TRANS$ are concurrently enabled at any reachable marking, $M \in [M_0\rangle$, then the following must hold from the definition of enabling

$$Pre'(tr1 + tr2) \leq M \leq I'(tr1 + tr2) \tag{8.1}$$

From the definitions of Pre' and I'

$$
\begin{aligned}
Pre'(tr1 + tr2) &= Pre(tr1) + Pre(tr2) &\tag{8.2}\\
I'(tr1 + tr2) &= I'(tr1) \cap I'(tr2) \\
&= I(tr1) \cap I(tr2) \\
&= Pre(tr1) \cap Pre(tr2) &\tag{8.3}
\end{aligned}
$$

Inserting equations 8.2 and 8.3 into 8.1

$$Pre(tr1) + Pre(tr2) \leq M \leq Pre(tr1) \cap Pre(tr2) \tag{8.4}$$

From the definition of multiset intersection (see Appendix A), this inequality can only be satisfied if $Pre(tr1) = \emptyset$ and $Pre(tr2) = \emptyset$ which is forbidden by the above definition. Hence two transition modes cannot be concurrently enabled. $\square$

Proposition 8.2 *For $tr \in TRANS$ enabled at a marking M, the transition rule is given by*

$$M' = Post(tr)$$

Proof: From Proposition 8.1, only one transition mode can occur at any instant (interleaving) and the transition rule becomes for all $tr \in TRANS$

$$M' = M - Pre(tr) + Post(tr) \tag{8.5}$$

From the enabling condition and definition of *completely resetting* P-nets

$$
\begin{aligned}
&Pre(tr) \leq M \leq I(tr) \\
\Rightarrow\ &Pre(tr) \leq M \leq Pre(tr) \\
\Rightarrow\ &\quad M = Pre(tr) \tag{8.6}
\end{aligned}
$$

P-Graph

A, B: Non-empty sets
$A \subseteq B$
$x : A, x_m : \mu A, y_m : \mu B$
$M_0(p1) = A, M_0(p2) = \emptyset$

P-Net

$S = \{p1, p2\}$
$T = \{t1\}$
$C = \{A, B, A \times \mu A \times \mu B\}$
$C(p1) = A$
$C(p2) = B$
$C(t1) = A \times \mu A \times \mu B$
$Pre(p1, t1) = \pi_1 + \pi_2$
$I(p1, t1) = Pre(p1, t1)$
$Post(p1, t1) = \pi_2$
$Pre(p2, t1) = \pi_3$
$I(p2, t1) = Pre(p2, t1)$
$Post(p2, t1) = \pi_1$
where π_1, π_2, π_3 are projection functions:
$\pi_1 : C(t1) \to A$
$\pi_2 : C(t1) \to \mu A$
$\pi_3 : C(t1) \to \mu B$
the capacities of $p1$ and $p2$ are infinite
$M_0 = \{(p1, a) \mid a \in A\}$

Figure 8.1: Completely Resetting P-net simulating Figure 7.4

118
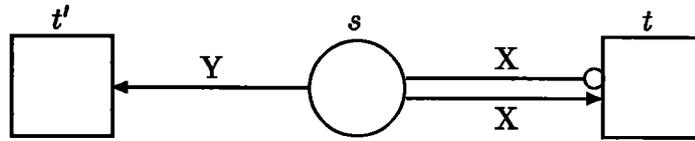
Figure 8.2: Transitions t and t' cannot occur concurrently

and thus from equation 8.5, for all $tr \in TRANS, M' = Post(tr)$. $\square$

Thus the follower marking is independent of the initial marking and it is this property that gives rise to the term *reset*.

Although it is possible to model applications with completely resetting P-nets, it is far too unwieldy and often will not match with the intuition of the designer as every transition affects every place. A simple example is given in figure 8.1. It models the situation in figure 7.4, where an item from a store is transferred to another store. With the net in figure 7.4, as many items as exist in $p1$ may be transferred as all the modes of $t1$ are concurrently enabled. This is not the case in figure 8.1 as only one mode is enabled for any given marking. The net is much more complex and rather convoluted and counter-intuitive. For these reasons completely resetting P-nets will not be investigated further.

What we would like is to be able to reset only the places of interest, not every place. This is investigated in the rest of this chapter.

8.2 P-nets with the reset property

Instead of resetting the whole marking of a P-net, we now consider the resetting of the marking of individual places. Consider the P-Graph of figure 8.2 where X and Y are any legal arc insciptions. Transition t has the reset property over place s. We assert that a mode of transition t cannot occur concurrently with any other transition mode, (including its own), that has a pre map dependent on place s.

Proposition 8.3 *For a P-net with transitions $t, t' \in T$, and place $s \in S$ with $Pre(s,t) = I(s,t)$, a mode of t, $m \in C(t)$ where $Pre(s,t;m) \neq \emptyset$, and a mode of t', $m' \in C(t')$ with $Pre(s,t';m') \neq \emptyset$, can not occur concurrently.*

Proof: The proof is essentially the same as the proof of proposition 8.1. From the enabling condition restricted to place s and the reset condition $(Pre(s,t) = I(s,t))$ it follows that for any $m \in C(t)$ and $m' \in C(t')$

$$Pre(s,t;m) + Pre(s,t';m') \leq M(s) \leq Pre(s,t;m) \cap I(s,t';m')$$

A necessary condition for this inequality to be satisfied is $Pre(s,t';m') = \emptyset$. Hence transitions t and t' cannot occur concurrently when $Pre(s,t';m') \neq \emptyset$. $\square$

An immediate specialisation of this proposition is:

Proposition 8.4 *For a P-net with transition $t \in T$ and place $s \in S$ such that $Pre(s,t) = I(s,t)$ and $\forall m \in C(t), Pre(s,t;m) \neq \emptyset$, the modes of t can never be concurrently enabled.*

Proof: The proof follows from proposition 8.3 by setting $t' = t$. $\square$

The reset condition ensures that the only marking that can satisfy the precondition is equality with the pre map. More precisely

Proposition 8.5 *For a P-net with transition $t \in T$ and place $s \in S$ such that $Pre(s,t) = I(s,t)$ and a mode, $m \in C(t)$, enabled at marking $M(s)$, then $Pre(s,t;m) = M(s)$.*

Proof: The proof follows immediately from the reset property and the enabling condition. For a mode $m \in C(t)$ enabled at $M(s)$,

$$Pre(s,t;m) \leq M(s) \leq I(s,t;m)$$
$$\Rightarrow \quad Pre(s,t;m) \leq M(s) \leq Pre(s,t;m)$$
$$\Rightarrow \qquad M(s) = Pre(s,t;m) \tag{8.7}$$

$\square$

Corollary 8.1 *A consequence of proposition 8.5 is that if transition, t, occurs in mode m, then the follower marking of place s is given by $M'(s) = Post(s,t;m)$.*

Proof: Follows immediately from the transition rule and equation 8.7. $\square$

8.3 Purging

This section describes how to empty a place of its current marking. This will be referred to as *purging*.

If we wish to purge a place s with a single occurrence of a mode of transition t, we need a mode, $m \in C(t)$, for every marking of s, $M(s) \in \mu C(s)$. $C(t)$ will in general be a product. To guarantee a mode for each marking, let $\mu C(s)$ be one of the sets in the product. We also need the pre map, $Pre(s,t)$, to select the current marking. From proposition 8.5 this can be done by using a reset: $I(s,t) = Pre(s,t)$. $Pre(s,t)$ and $I(s,t)$ will then be projection functions that select out a marking in $\mu C(s)$ from $C(t)$.

Let $D_s = \mu C(s)$ and $C(t) = D_s \times D'$ where D' is in general a product set dependent upon the inscriptions of the other arcs connected to t. (If there is no requirement for a product, then $C(t) = D_s$ or equivalently let D' contain a single nondescript element so that the product $D_s \times D'$ is isomorphic to D_s. The latter approach is used here, to include this special case in the general presentation.)

A projection function is defined as

$$\pi_{st} : C(t) \longrightarrow \mu C(s)$$

where $\pi_{st}(d_s, d') = d_s$ with $d_s \in D_s$ and $d' \in D'$.

We set $I(s, t) = Pre(s, t) = \pi_{st}$.

Corollary 8.1 ensures that whenever transition t occurs, place s will be emptied of its current marking, so long as $Post(s, t; m) = \emptyset$.

The following propositions summarise the above discussion.

Proposition 8.6 *Given a P-net with $t \in T$, $s \in S$, $C(t) = \mu C(s) \times D'$ and $Pre(s, t) = \pi_{st}$, then $\exists m \in C(t)$, such that $Pre(s, t; m) = M(s)$ for any $M(s) \in \mu C(s)$.*

Proof: Firstly $M(s) \in \mu C(s)$. From the definitions of $C(t)$ and π_{st}, noting that π_{st} is surjective, $Pre(s, t)$ ranges over $\mu C(s)$. Hence we can always choose an appropriate argument, m, of $Pre(s, t)$ such that $Pre(s, t; m) = M(s)$. $\square$

The conditions under which this proposition holds ensure that a mode, m, of transition t can always be chosen so that its image under the pre map $Pre(s, t; m)$ is equal to the current marking.

Now we need to ensure that a mode is only enabled by one marking, that being the current marking. This is achieved with a reset.

Proposition 8.7 *Given a P-net as in the previous proposition with $Pre(s, t) = I(s, t)$ and $\forall m \in C(t), Post(s, t; m) = \emptyset$, then when t occurs in any mode, place s is purged, i.e. $M'(s) = \emptyset$.*

Proof: Follows immediately from corollary 8.1. $\square$

8.3.1 Graphical Representation

In the P-Graph, $\mu C(s)$ is a carrier of the many-sorted algebra, H, (i.e. $H_r = \mu C(s)$) and a variable that ranges over multisets over $C(s)$ is included in the signature, for example $Y : \mu C(s)$. The arc (s, t) and inhibitor arc are annotated by a tuple consisting of this variable: $A(s, t) = IA(s, t) = Y$. Any variable name could be chosen and the declaration part would provide its type. However, a capital letter can be used to alert readers of the graph part that this variable is of higher order type.

To save space in the P-Graph, the normal and inhibitor arcs are superimposed and only one inscription is needed, because $A(s, t) = IA(s, t)$. This may be referred to as a *reset arc*.

8.3.2 Example: Purging a place

An example of a simple net where all tokens are removed from a place, irrespective of its marking, is shown in figure 8.3.

Transition $t1$ is always enabled in one of its modes but never concurrently with itself, except when $M(p1) = \emptyset$. When $t1$ occurs, $p1$ is emptied ($M'(p1) = \emptyset$).

P-Graph

A: Non-empty set
$Y : \mu A$
$M_0(p1) = A$

P-Net

$S = \{p1\}$
$T = \{t1\}$
$C = \{A, \mu A\}$
$C(p1) = A$
$PLACE = \{(p1, a) \mid a \in A\}$
$C(t1) = \mu A$
$TRANS = \{(t1, m) \mid m \in \mu A\}$
Pre and *I* are bijections: $TRANS \to \mu PLACE$
where for all $m \in \mu A, I(t1, m) = Pre(t1, m) = \{((p1, a), m(a)) \mid a \in A\}$
For all $tr \in TRANS$, $Post(tr) = \emptyset$
$\forall a \in A, K(p1, a) = \infty$
$M_0 = \{(p1, a) \mid a \in A\}$

Figure 8.3: Purging a place

When $M(p1) = \emptyset$, $t1$ is enabled by a mode in which $I(tr) = Pre(tr) = \emptyset$. An unusual phenomenon occurs where the degree of self-concurrency of the transition mode is unbounded and its occurrence has no effect on the current state. This possibility may be excluded by restricting the set of multisets over A to be non-empty.

The situation is the same as in figure 8.3 with the following modifications. In the P-Graph and in the P-net we replace μA by $\mu A \setminus \{\emptyset\}$ so that Pre and I become injections. Hence for all $tr \in TRANS$, $Pre(tr) \neq \emptyset$. Thus $t1$ is not enabled when the marking is null, but is enabled for all other markings.

8.4 Transferring a Marking

There are situations when we would like to be able to transfer the contents of one place to another place; for example in resetting. This may be achieved by setting the post map for the receiving place to the pre map for the place that is being purged.

Let s be the place that will be purged and $s1$ the one that will receive its contents. We require that $C(s) \subseteq C(s1)$.

As above we have $I(s,t) = Pre(s,t) = ID$, where ID is the identity function

$$ID : \mu C(s) \to \mu C(s)$$

and we now set $Post(s1,t) = Pre(s,t)$.

In the P-Graph, we shall have a reset arc from s to t annotated by $Y : \mu C(s)$, as will be the (normal) arc from t to $s1$ $(A(s,t) = IA(s,t) = A(t,s1) = Y)$.

8.5 Purging subbags of Markings

In this section the previous results are extended to purging multisets over a subset of a place's colour set. Consider a transition, t, with an input place, s. Let G_s be a subset of $C(s)$ and form the set of multisets over G_s, μG_s.

Now include μG_s as part of the product set comprising $C(t)$ in a similar way to section 8.3. Let $D_s = \mu G_s$, $C(t) = D_s \times D'$, $d_s \in D_s$, $d' \in D'$, so that $m = (d_s, d')$ as before. We can define $Pre(s,t)$ and $I(s,t)$ as follows. For $g \in C(s)$ and all $m \in C(t)$,

$$mult(g, Pre(s,t;m)) = \begin{cases} mult(g, d_s) & \text{if } g \in G_s \\ 0 & \text{otherwise} \end{cases}$$

$$mult(g, I(s,t;m)) = \begin{cases} mult(g, d_s) & \text{if } g \in G_s \\ \infty & \text{otherwise} \end{cases}$$

Thus the pre and inhibitor maps are determined by the subset G_s.

We may now generalise the propositions in section 8.3.

Proposition 8.8 *Given a P-net with* $t \in T$, $s \in S$, $G_s \subseteq C(s)$, $C(t) = \mu G_s \times D'$ *and* $Pre(s,t)$ *as above, then* $\exists m \in C(t)$, *such that for* $g \in C(s)$ *and any* $M(s) \in \mu C(s)$,

$$mult(g, Pre(s,t;m)) = \begin{cases} mult(g, M(s)) & \textit{if } g \in G_s \\ 0 & \textit{otherwise} \end{cases}$$

Proof: For any $M(s)$, consider the subbag, SUB, where the multiplicities of all elements not in G_s are set to zero and the others are as they were in $M(s)$, i.e.

$$mult(g, SUB) = \begin{cases} mult(g, M(s)) & \text{if } g \in G_s \\ 0 & \text{otherwise} \end{cases}$$

Now define $SUB' \in \mu G_s$ where for all $g \in G_s$, $mult(g, SUB') = mult(g, SUB)$. Hence for $d' \in D'$, we may set $m = (SUB', d')$ so that

$$mult(g, Pre(s,t;m)) = \begin{cases} mult(g, SUB') & \text{if } g \in G_s \\ 0 & \text{otherwise} \end{cases}$$

For $g \in G_s$, $mult(g, SUB') = mult(g, SUB) = mult(g, M(s))$ and the proposition follows immediately. $\square$

Proposition 8.9 *Given a P-net as in the previous proposition with* $m \in C(t)$, $I(s,t;m)$ *as defined above and* $\forall m \in C(t), Post(s,t;m) = \emptyset$, *then when* t *occurs in mode* m, *place* s *is purged of all tokens in* G_s, *i.e.*

$$mult(g, M'(s)) = \begin{cases} 0 & \textit{if } g \in G_s \\ mult(g, M(s)) & \textit{otherwise} \end{cases}$$

Proof: To prove this proposition we need the following lemma.

Lemma 8.1 *Given the P-net of proposition 8.9 with the transition mode,* (t,m), *enabled at* M, *then for* $g \in C(s)$

$$mult(g, Pre(s,t;m)) = \begin{cases} mult(g, M(s)) & \textit{if } g \in G_s \\ 0 & \textit{otherwise} \end{cases}$$

Proof: For all $m \in C(t)$, for all $g \notin G_s$, $mult(g, Pre(s,t;m)) = 0$ (by its definition). From the enabling condition on place s it follows that for $g \in G_s$

$$mult(g, Pre(s,t;m)) \leq mult(g, M(s)) \leq mult(g, I(s,t;m))$$
$$\Rightarrow \quad mult(g, Pre(s,t;m)) \leq mult(g, M(s)) \leq mult(g, Pre(s,t;m))$$
$$\Rightarrow \quad \quad mult(g, Pre(s,t;m)) = mult(g, M(s))$$

which proves the lemma. $\square$

From the firing rule, this lemma, and that for all $m \in C(t)$, $Post(s,t;m) = \emptyset$ we have for all $m \in C(t)$

$$M'(s) = M(s) - Pre(s,t;m) + Post(s,t;m)$$

124

$$\Rightarrow M'(s) = M(s) - Pre(s,t;m)$$

$$\Rightarrow mult(g, M'(s)) = \begin{cases} 0 & \text{if } g \in G_s \\ mult(g, M(s)) & \text{otherwise} \end{cases}$$

which proves the proposition. $\square$

(Note that the condition that for $m \in C(t)$ and $g \notin G_s$, $mult(g, Pre(s,t;m)) = 0$ and $mult(g, I(s,t;m)) = \infty$ implies that there is no enabling requirement (from t) on tokens in place s, that are not in G_s.)

8.5.1 Graphical Representation

In the P-Graph, let μG_s be a carrier of the many-sorted algebra and include a variable $Z : \mu G_s$ in the signature. To purge place, s, of a subbag of $M(s)$ (that is a multiset over G_s) by an occurrence of transition, t, we set $A(s,t) = IA(s,t) = Z$.

We now need to slightly modify the mapping from the P-Graph to the P-net (section 7.4) to accommodate the fact that $\lambda Z.Z$ is an identity mapping from μG_s to itself and hence multiplicities of tokens are only defined for $g \in G_s$ and not otherwise. For all modes $m \in C(t)$ and for $g \in C(s) \setminus G_s$: for the Pre and $Post$ maps we use the default that $mult(g, Pre(s,t;m)) = mult(g, Post(s,t;m)) = 0$; and for the inhibitor that $mult(g, I(s,t;m)) = \infty$.

Note that this is rather like the convention adopted in section 7.9, that zero multiplicities are not shown for input and output arcs and that infinite multiplicities for inhibitor arcs are not represented explicitly. (If all multiplicities for the pre or post map are zero, then no arc is drawn, and similarly if all multiplicities for the inhibitor map are infinite, then no inhibitor arc is drawn - it is just the natural extension of this.)

8.5.2 Notation for Subsets of Product Sets

This section develops a notation for annotating reset arcs when $C(s)$ is a product set, G, and we wish to purge $M(s)$ of all tokens in a subset of G. Let $G = G_1 \times \ldots \times G_n$, and represent an element of G by the tuple $(g_1, \ldots, g_n)$, where $g_i \in G_i$ are constants for $i \in In$ where $In = \{1, \ldots, n\}$.

The notation $(g_1, \ldots, g_n)$ may also represent the singleton set $\{(g_1, \ldots, g_n)\}$. The notation $*$ is introduced in position i to indicate a subset of G where all values of G_i are included. Thus $(g_1, \ldots, g_{i-1}, *, g_{i+1}, \ldots, g_n)$ represents the subset $\{g1\} \times \ldots \times \{g_{i-1}\} \times G_i \times \{g_{i+1}\} \times \ldots \times \{g_n\}$. This may be generalised to allow a '$*$' to replace any constant, g_k, in the tuple where $\{g_k\}$ is then replaced by G_k to obtain the subset. If all the constants are replaced by stars then the subset is the original set G.

In general, given a tuple consisting of constants and stars, let $G_s \subseteq G$, and $I_* \subseteq In$ be the set of positions in which stars occur in the tuple. Then the subset of G corresponding to the tuple notation is given by

$$G_s = \{(g_1, \ldots, g_n) \mid g_i \in G_i, i \in I_*\}.$$

For $C(s) = G_1 \times \ldots \times G_n$, it is convenient to use the above notation to indicate the subset G_s. For $i \in In$, $a_i ::= g_i|*$, we annotate a reset arc (s,t) by $\#(a1, \ldots, a_n)$, to indicate that all tokens of the subset G_s (determined by the tuple $(a1, \ldots, a_n)$, as above) are to be purged from place, s, on an occurrence of transition, t (in any mode).

For example, if the reset arc (s,t) is annotated by $\#(g_1, \ldots, g_n)$, then the subset of $C(s)$ is the singleton $G_s = \{(g_1, \ldots, g_n)\}$, and all instances of the token $(g_1, \ldots, g_n)$ in $M(s)$ would be purged on the occurrence of t. Similarly, for the reset arc annotation $\#(g_1, \ldots, g_{i-1}, *, g_{i+1}, \ldots, g_n)$, the subset is $G_s = \{(g_1, \ldots, g_n) \mid g_i \in G_i\}$, so that any member of G_s, would be purged from $M(s)$ on the occurrence of t.

To provide an interpretation of the P-Graph as a P-net, we need to obtain the notation given in the previous section. This is done as follows. Replace $\#(a1, \ldots, a_n)$, on the reset arc (s,t), by a variable, $v : \mu G_s$, where G_s is determined by the tuple notation.

The advantage of the $\#$-tuple notation, on the reset arc, is that it allows the subset, G_s, to be identified on the graph part of the P-Graph without having to refer back to the declaration part.

8.6 Purging Partitions of Markings

Consider a partition of $C(s)$

$$C(s) = \bigcup \{G^1, \ldots, G^k\}$$

and for $j \in I_k = \{1, \ldots, k\}$, let $D^j = \mu G^j$.

Define the colour set for transition, t, as follows.

$$C(t) = (\bigcup_j D^j) \times D'$$

where D' is as before.

Let $D_s = \bigcup_j D^j$, then $C(t) = D_s \times D'$. For $d \in D_s, d' \in D'$, for all $j \in I_k$ and for all $g^j \in G^j$, the pre and inhibitor maps are given by, for all $(d, d') \in C(t)$,

$$mult(g^j, Pre(s,t;(d,d'))) = \begin{cases} mult(g^j, d) & \text{if } d \in D^j \\ 0 & \text{otherwise} \end{cases}$$

$$mult(g^j, I(s,t;(d,d'))) = \begin{cases} mult(g^j, d) & \text{if } d \in D^j \\ \infty & \text{otherwise} \end{cases}$$

Similar propositions to those of the previous section may now be stated.

Proposition 8.10 *Given a P-net with $t \in T$, $s \in S$, $g \in C(s)$, $C(s) = \bigcup\{G^1, \ldots, G^k\}$, for $j \in I_k$, $D^j = \mu G^j$, $C(t) = (\bigcup_j D^j) \times D'$ and $Pre(s,t)$ as above, then for $d^j \in D^j$, $\exists m = (d^j, d') \in C(t)$, such that for any $M(s) \in \mu C(s)$,*

$$mult(g, Pre(s,t;(d^j,d'))) = \begin{cases} mult(g, M(s)) & \text{if } g \in G^j \\ 0 & \text{otherwise} \end{cases}$$

Proof: Essentially the same as that for proposition 8.8. For any $M(s)$, consider the subbag, SUB, where the multiplicities of all elements not in G^j are set to zero and the others are as they were in $M(s)$, i.e.

$$mult(g, SUB) = \begin{cases} mult(g, M(s)) & \text{if } g \in G^j \\ 0 & \text{otherwise} \end{cases}$$

Now for all $j \in I_k$, define $d^j \in \mu G^j$ where for all $g \in G^j$, $mult(g, d^j) = mult(g, SUB)$. Hence for $d' \in D'$,

$$mult(g, Pre(s, t; (d^j, d'))) = \begin{cases} mult(g, d^j) & \text{if } g \in G^j \\ 0 & \text{otherwise} \end{cases}$$

For $g \in G^j$, $mult(g, d^j) = mult(g, SUB) = mult(g, M(s))$ and the proposition follows immediately. $\square$

We can now state that an occurrence of t will purge a member of a partition of $C(s)$.

Proposition 8.11 *Given the P-net of proposition 8.10 with $m \in C(t)$ and $I(s, t; m)$ as defined above and $\forall m \in C(t), Post(s, t; m) = \emptyset$, then for $d^j \in D^j$, when t occurs in mode $m = (d^j, d') \in C(t)$, place s is purged of all tokens in G^j, i.e.*

$$mult(g, M'(s)) = \begin{cases} 0 & \text{if } g \in G^j \\ mult(g, M(s)) & \text{otherwise} \end{cases}$$

Proof: Follows directly from proposition 8.9 on setting $G_s = G^j$ for any $j \in I_k$. $\square$

8.6.1 Graphical Representation

In the P-Graph, let $D_s = \bigcup_j \mu G^j$ be a carrier of the many-sorted algebra and include a variable $Y : D_s$ in the signature. If we wish to purge place s of all elements of any member of a partition of $C(s)$ by an occurrence of transition t, then for $(s, t) \in F$ and $(s, t) \in IF$, set $A(s, t) = IA(s, t) = Y$.

The same default convention, regarding the mapping of the P-Graph to the P-net, discussed in section 8.5.1, is adopted here.

8.6.2 Notation for Partitions of Product Sets

In a similar way to section 8.5.2, the aim here is to develop a notation for annotating reset arcs when $C(s)$ is a product set, G, and we wish to purge $M(s)$ of all tokens in a subset of G, where this subset is a member of a partition of G.

For $G = G_1 \times \ldots \times G_n$, consider a tuple of typed variables $(v_1, \ldots, v_n)$ where for all $i \in In$, $v_i : G_i$. We may use this tuple (in conjunction with # - see below) as notation to indicate a partition of G in which each element of G is a singleton set member of the partition. If $Part(G)$ denotes a partition of G, then

$$Part(G) = \{\{(g_1, \ldots, g_n)\} \mid g_i \in G_i, i \in In\}.$$

The * notation can now be used in the same way as in section 8.5.2. For example, if v_i is replaced by a star we obtain the notation $(v_1, \ldots, v_{i-1}, *, v_{i+1}, \ldots, v_n)$, with the corresponding partition

$$Part(G) = \{\{(g_1, \ldots, g_n) \mid g_i \in G_i\}, \mid g_j \in G_j, j \in In \setminus \{i\}\}.$$

This may be extended to allow stars in any of the tuple positions. In general, the corresponding partition is

$$Part(G) = \{\{(g_1, \ldots, g_n) \mid g_l \in G_l, l \in I_*\}, \mid g_j \in G_j, j \in In \setminus I_*\}$$

with I_* as defined in 8.5.2. If stars are placed in every position the partition reduces to the original set.

For $C(s) = G_1 \times \ldots \times G_n$, we may replace the variable $Y : D_s$ annotating the reset arc (s, t), by the above tuple notation preceeded by a $\#$ to indicate the partition in the graph part. Two examples are

- If the annotation on the reset arc is $\#(v_1, \ldots, v_n)$, then a generic member of the partition is $G^j = \{(g_1, \ldots, g_n)\}$.

- If the annotation on the reset arc is $\#(v_1, \ldots, v_{i-1}, *, v_{i+1}, \ldots, v_n)$, the tuple now defines another partition determined by the notation defined above.

We may of course have a star in any of the positions of the tuple. (When there is no partitioning of $C(s)$, the corresponding notation, $\#(*, *, \ldots, *)$, corresponds to Y, where $Y : \mu C(s)$.)

8.6.3 Example: Aborting a Broadcast

In concurrent systems design, situations often arise in which we wish to send information to a list of destinations; for example broadcast protocols. Quite often there is a mechanism for aborting, part-way through the broadcast. A way of specifying this behaviour is to have a list of possible destinations for each source. The broadcast is received by an arbitrary number of destinations at a time (the destinations being chosen arbitrarily from the list), with the destinations that have been serviced being transferred to a destinations-serviced list. Aborts may occur at any stage. When an abort occurs from a particular source, all the destinations that have been serviced (for that source) are removed from the serviced list and reinstated on the original list.

A P-Graph specification of the management of the list is given in figure 8.4. The list of destinations for each source is stored in place, $p1$, and the serviced destinations in $p2$. Broadcasting occurs on firing $t1$ and aborts on the occurrence of $t2$. When source a aborts, all tokens with a's address in the first element of the source-destination address pairs stored in $p2$ are removed from $p2$ and transferred to $p1$. Thus on the occurrence of $t2$, the notation $\#(s, *)$ on the reset arc can be read as, for a particular value of s, *add1*, remove all tokens in $p2$ with *add1* as the first element of the pair. This then defines the multiset which is passed onto $p1$.

The corresponding P-net has also been included to show how the P-Graph can be interpreted. (Note that ID is an identity function.)

128

P-Graph

Declarations

S: Finite set of source addresses
D: Finite set of destination addresses
Variables: s:S, d:D
(Replace $\#(s,*)$ by $Y{:}\bigcup_{s\in S}\mu\{(s,d)\mid d\in D\})$
$\forall s\in S, \forall d\in D,\ K(p1;s,d) = 1$
$\forall s\in S, \forall d\in D,\ K(p2;s,d) = 1$
$M_0(p1) = S\times D$
$M_0(p2) = \emptyset$

Graph
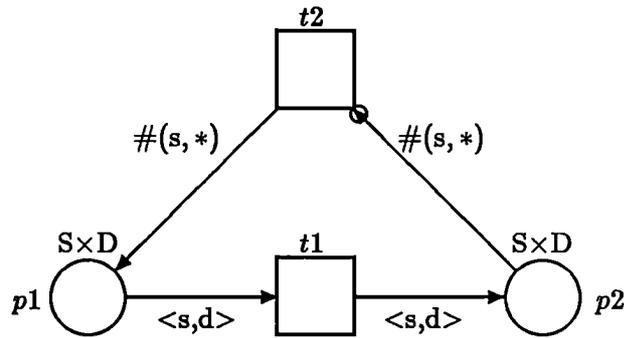
P-Net

$S = \{p1, p2\}$
$T = \{t1, t2\}$
$\mathcal{C} = \{S\times D, \bigcup_{s\in S}\mu\{(s,d)\mid d\in D\}\}$
$C(p1) = C(p2) = S\times D$
$C(t1) = S\times D$
$C(t2) = \bigcup_{s\in S}\mu\{(s,d)\mid d\in D\}$
$Pre(p1, t1) = Post(p2, t1) = \text{ID}{:}S\times D\longrightarrow S\times D$
$Pre(p2, t2) = Post(p1, t2)$
$\forall m2 \in C(t2), \forall (s,d)\in S\times D$

$$mult((s,d), Pre(p2, t2; m2)) = \begin{cases} mult((s,d), m2) & \text{if } m2 \in \mu\{(s,d)\mid d\in D\} \\ 0 & \text{otherwise} \end{cases}$$

$$mult((s,d), I(p2, t2; m2)) = \begin{cases} mult((s,d), m2) & \text{if } m2 \in \mu\{(s,d)\mid d\in D\} \\ \infty & \text{otherwise} \end{cases}$$

$\forall m1 \in C(t1), Pre(p2, t1; m1) = Post(p1, t1; m1) = \emptyset$
$\forall m2 \in C(t2), Pre(p1, t2; m2) = Post(p2, t2; m2) = \emptyset$
$\forall m1 \in C(t1), \forall (s,d)\in S\times D$
$mult((s,d), I(p1, t1; m1)) = \infty$
$mult((s,d), I(p2, t1; m1)) = \infty$
$\forall m2 \in C(t2), \forall (s,d)\in S\times D,\ mult((s,d), I(p1, t2; m1)) = \infty$
The capacities and initial markings are given in the P-Graph.

Figure 8.4: Aborts: Address List Management

P-Graph

Declarations

A: finite non-empty set
B: finite non-empty set
$C = \bigcup_{a \in A} \{a\} \times \mu\{(a, b) \mid b \in B\}$
Variables: x:A, Y:$\bigcup_{a \in A} \mu\{(a, b) \mid b \in B\}$
$\forall a \in A, \forall b \in B, K(p2;a,b) = 1$
$M_0(p1) \subseteq A$
$M_0(p2) = A \times B$

Graph
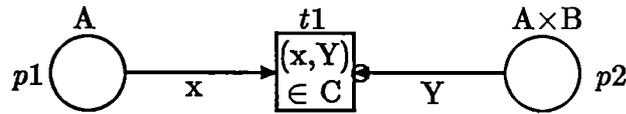
P-Net

$S = \{p1, p2\}$
$T = \{t1\}$
$C = \{A, A \times B, \bigcup_{a \in A} \{a\} \times \mu\{(a, b) \mid b \in B\}\}$
$C(p1) = A$
$C(p2) = A \times B$
$C(t1) = \bigcup_{a \in A} \{a\} \times \mu\{(a, b) \mid b \in B\}$
$Pre(p1, t1) = \pi_1 : \bigcup_{a \in A} \{a\} \times \mu\{(a, b) \mid b \in B\} \to A$
where π_1 is the projection function for the first argument
$\forall m1 = (c, d) \in C(t1), \forall (a,b) \in A \times B$
$$mult((a,b), Pre(p2, t1; (c, d))) = \begin{cases} mult((a,b), d) & \text{if } d \in \mu\{(a, b) \mid b \in B\} \\ 0 & \text{otherwise} \end{cases}$$
$$mult((a,b), I(p2, t1; (c, d))) = \begin{cases} mult((a,b), d) & \text{if } d \in \mu\{(a, b) \mid b \in B\} \\ \infty & \text{otherwise} \end{cases}$$
$\forall m1 \in C(t1), \forall a \in A, mult(a, I(p1, t1; m1)) = \infty$
$\forall m1 \in C(t1), Post(p1, t1; m1) = Post(p2, t1; m1) = \emptyset$
$\forall a \in A, K(p1;a) = \infty$
$\forall a \in A, \forall b \in B, K(p2;a,b) = 1$
Initial markings are given in the P-Graph.

Figure 8.5: Selecting a member of a partition for purging

8.6.4 Purging a Selected Partition Member

This section illustrates a notation for purging all elements of a member of a partition of a place's colour set, that occur in the place's marking, where the member of the partition is selected according to the marking of another place. This situation arises in aborting broadcasts (see chapter 10, figure 10.8).

Declarations

A: finite non-empty set
B: finite non-empty set
Variables: x:A
$\forall a \in A, \forall b \in B, K(p2;a,b) = 1$
$M_0(p1) \subseteq A$
$M_0(p2) = A \times B$

Graph
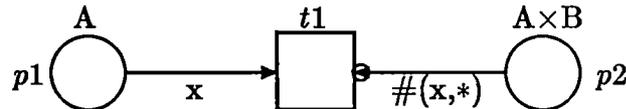
Figure 8.6: A more readable representation for purging a selected member of a partition

An example is shown in figure 8.5. The member of the partition of $A \times B$ that is to be purged is determined by the value of the variable x annotating the arc $(p1, t1)$. Y is a variable which ranges over the sets of multisets over each of the partition members. Without the transition condition, the colour set associated with $t1$ would be $C(t1) = A \times \bigcup_{a \in A} \mu\{(a, b) \mid b \in B\}$. This would correspond to any partition member being purged independently of the value of x. To ensure that the partition selected does depend on the value of x, a transition condition is used which restricts the colour set to

$$C(t1) = \bigcup_{a \in A}\{a\} \times \mu\{(a, b) \mid b \in B\}.$$

Although this representation does allow a direct translation from the P-Graph to the P-net, it is rather difficult to understand. A more easily understood graphical representation is shown in figure 8.6, where we have used the tuple notation developed earlier in this section.

For $a \in A$, and x = a, we can read $\#(x, *)$ (associated with a reset arc) to mean: purge all tokens in place $p2$ with the value 'a' in the first position of the pair, when $t1$ occurs. In the declaration, we only declare the variables that are explicitly represented in the P-Graph; in this case, x. To map back to the P-Graph in figure 8.5, $\#(x, *)$ is replaced by the variable Y and appropriately declared as $Y : \bigcup_{a \in A} \mu\{(a, b) \mid b \in B\}$. The type of this variable is fully determined by the $\#$-tuple notation. The matching of the variable x in both arc inscriptions is used to imply that the transition condition of figure 8.5 applies.

8.7 Purging Subsets of Partitions of Markings

The ideas of the last two sections may be generalised to allow for the purging of a subset of a partition. Let

$$C(s) = \bigcup\{G^1, \ldots, G^k\}$$

as before and I_s be a set of indices that index members of the partition included in the subset, so that $I_s \subseteq I_k(= \{1, 2, \ldots, k\})$. The set of occurrence modes for transition, t, is then defined as

$$C(t) = (\bigcup_{j \in I_s} D^j) \times D'$$

where $D^j = \mu G^j$.

Let $D_s = \bigcup_j D^j$, then $C(t) = D_s \times D'$. For $d \in D_s, d' \in D'$ and for all $j \in I_k$, $g^j \in G^j$ the pre and inhibitor maps are given by for all $(d, d') \in C(t)$,

$$mult(g^j, Pre(s, t; (d, d'))) = \begin{cases} mult(g^j, d) & \text{if } j \in I_s \text{ and } d \in D^j \\ 0 & \text{otherwise} \end{cases}$$

$$mult(g^j, I(s, t; (d, d'))) = \begin{cases} mult(g^j, d) & \text{if } j \in I_s \text{ and } d \in D^j \\ \infty & \text{otherwise} \end{cases}$$

The propositions of section 8.6 apply when $j \in I_s$.

8.7.1 Graphical Representation

In the P-Graph, let $D_s = \bigcup_{j \in I_s} \mu G^j$ be a carrier of the many-sorted algebra and include a variable $Y : D_s$ in the signature. To purge place s of all elements of any member of a subset of a partition of $C(s)$ by an occurrence of transition t, then for $(s, t) \in F$ and $(s, t) \in IF$, set $A(s, t) = IA(s, t) = Y$.

8.7.2 Notation for Subsets of Partitions of Product Sets

We can combine the notation developed in sections 8.5.2 and 8.6.2 to obtain a tuple notation consisting of constants, variables and stars to represent subsets of partitions of product sets. With G, g_i, v_i and $i \in In$ as previously defined we consider two situations.

The tuple $(g_1, \ldots, g_{i-1}, v_i, g_{i+1}, \ldots, g_n)$ represents a subset of the partition where all positions have variables. The subset is

$$\{\{(g_1, \ldots, g_n)\} \mid g_i \in G_i\}.$$

We can generalise this for variables in any number of positions. Let $I_v \subseteq In$ be the set of positions in which variables occur, then the subset is

$$\{\{(g_1, \ldots, g_n)\} \mid g_j \in G_j, j \in I_v\}.$$

When $I_v = In$ we have the notation described in section 8.6.2, where all positions of the tuple are occupied by variables.

132

A complete notation is now obtained by allowing stars to occur in any position. Let $I_* \subseteq In$ be the set of positions in which stars occur. The subset of partitions described above (see section 8.6.2) when using stars and variables is

$$\{\{(g_1,\ldots,g_n) \mid g_l \in G_l, l \in I_*\} \mid g_j \in G_j, j \in I_v\}$$

Of course, I_v and I_* must be disjoint.

For example, the tuple $(g_1,\ldots,g_{i-1},v_i,g_{i+1},\ldots,g_{k-1},*,g_{k+1},\ldots,g_n)$ represents the following subset of a partition defined above (see section 8.6.2) with all variables in the tuple except for a star at position k.

$$\{\{(g_1,\ldots,g_n) \mid g_k \in G_k\} \mid g_i \in G_i\}$$

As before, when $C(s)$ is a product set, we can use this notation, preceeded by a $\#$, to replace the variable $Y : D_s$ annotating a reset arc (s,t) to indicate the subset of the partition on the graph form. An example is

$$A(s,t) = IA(s,t) = \#(3,b,*)$$

where $C(s) = N_6 \times Bool \times N_8$ with $b : Bool$ a Boolean variable and for $n \in N$, $N_n = \{0,1,\ldots,n-1\}$. This can be read as: on an occurrence of t, where b is bound to $true$, remove all tokens of the form $(3,true,-)$ from place s. We could also have an occurrence of t where b is bound to $false$, in which case all tokens of the form $(3,false,-)$ would be removed from place s.

8.8 Discussion

This chapter has investigated the semantics of purging places for high-level nets. The basic idea is very simple, but the theory is quite complicated. If we think at the level of P/T-systems with threshold inhibitors, then we need a transition for every possible marking of the place we wish to purge. In general, for an infinite colour set associated with the place, the number of transitions would be uncountable (the cardinality of the set of multisets over an infinite set is uncountable). If we restrict the colour set to being finite, then the number of transitions (transition modes in the high-level net) becomes countably infinite. This of course has consequences for analysis (which are not investigated here).

Fortunately, in most practical situations, not only do we deal with finite sets, but also with finite resources so that places that we wish to purge will have a finite capacity. In this case, the number of underlying transitions required for purging is finite and we can always translate the P-nets to CP-nets for analysis.

Proposition 8.3 ensures that P-nets that only use inhibitors for resetting will always satisfy condition 1 of theorem 6.4. Hence only condition 2 will need to be checked to see if the transformation from P-nets to CP-nets will preserve concurrency.

An attempt has been made to provide an intuitively appealing syntax for purging a place of all elements of a member of (a subset of) a partition of a place's colour set when it is a product. This is based on a modification of the use of

tuples annotating arcs, similar to that of PrT-nets. Care has also been taken to provide a semantics for this notation in terms of P-nets. This work is not complete. In the example of selecting a partition member to be purged (section 8.6.4), the meaning in terms of a P-net for the particular example is given, but the procedure for mapping from the P-Graph with the # syntax to that without it is only discussed informally.

Part III

Application to Specification
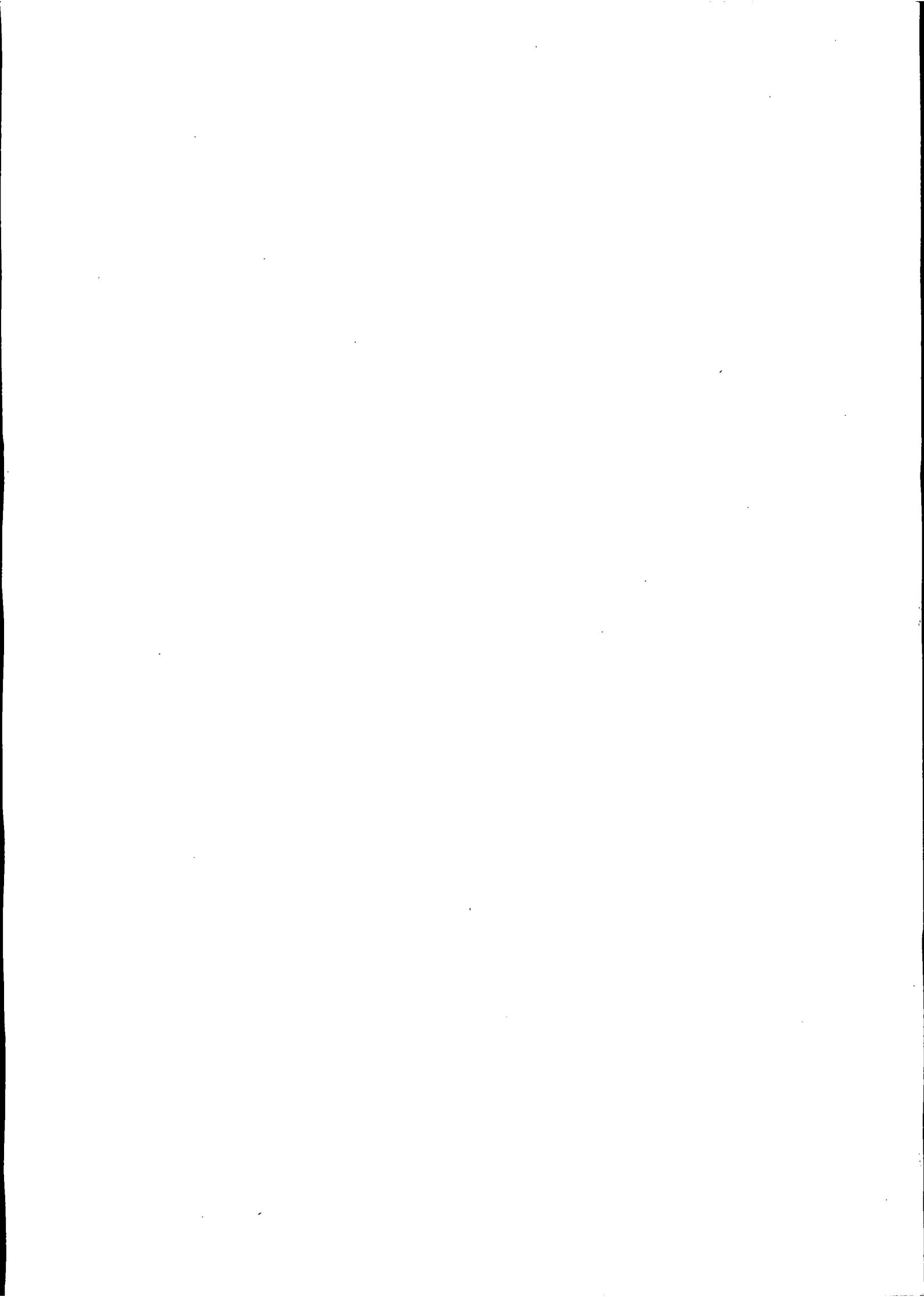
Chapter 9

Communications Examples

9.1 Queues

Queues are important data structures in communications. For example they are often used when defining service specifications. This section illustrates how the P-Graph can be used to model queues, by taking the First-In-First-Out (FIFO) and Last-In-First-Out (LIFO) queues as examples.

We can use a place to store the queue (or a set of queues) and the occurrence of associated transitions to manipulate the queue. The terms annotating the arcs will then determine the service discipline of the queue. For FIFO and LIFO queues we need to record the order of arrival of items joining the queue. Let the set of possible items that can join the queue be A. We can record order by forming the set of strings, A^*, over A, and use this as our colour set for the queueing place. Thus for a place, s, representing a queue or set of queues, let $C(s) = A^*$, so that a token represents a queue.

We shall denote the empty string by ε and the set of strings over A of length no more than n by A^{n*}.

9.1.1 Functions

We define two functions that will be generally useful for the specification of queueing systems.

Concatenation: A binary function on strings which appends one string to the tail of another.

$$. : A^* \times A^* \longrightarrow A^*$$

where for $i = 1, \ldots, j$, $a_i \in A$, let $x = a_1 a_2 \ldots a_j$; and for $i = 1, \ldots, k$, $b_i \in A$, let $y = b_1 b_2 \ldots b_k$; then $x, y \in A^*$, and $x.y = a_1 a_2 \ldots a_j b_1 b_2 \ldots b_k$.

We use the convention that the first letter of the string is on the left and the last on the right.

Word Length: A unary function returning the length of a string.

$$WL : A^* \longrightarrow N$$

136

where for $x = a_1 a_2 \ldots a_n, WL(x) = n$. It will be convenient to use outfix notation $|x|$ for $WL(x)$.

9.1.2 Predicates

A number of predicates on strings can be used in transition conditions. We discuss a few here, although they are not illustrated further in our FIFO and LIFO examples.

We may need predicates for string lengths. For this, the standard binary predicates *less than* $<$, *less than or equal to* $\leq$, *equals* $=$, *not equals* $\neq$, *greater than* $>$ and *greater than or equal to* $\geq$ are useful. For example $|x| < n$, where $x \in A^*, n \in N^+$, to restrict a queue size or $|x| = n$ to select strings of length n.

Other predicates like *is a prefix of* or *is a letter of* will be useful (see [66]) but are not illustrated here.

9.1.3 Examples

FIFO Queues

FIFO queues are important models for communications services and protocols as they are a convenient way of representing sequence preservation. Let us firstly consider an unbounded FIFO queue.

The queue is specified in the P-Graph of figure 9.1 along with its corresponding P-net. The place *FIFO* stores the items of the queue as a string. Transition *AF* (Add to FIFO) adds items of type A to the tail of the queue, whereas *SF* (Serve FIFO) removes the head of the queue. The concatenation operator has been defined above (section 9.1.1).

We now consider a bounded FIFO queue of length n as shown in figure 9.2. The addition of items to the queue is restricted by the typing of q to sequences of length less than n ($|q| < n$). Since $|x| = 1$, this ensures that the queue size never exceeds its bound, n. If $M(FIFO) = s$ and $|s| = n$, then AF is not enabled. In this case, an item must be served (SF must occur) before another item can join the queue.

In an application, A may be the union of a number of product sets, which will allow items to be structured. We may also index queues by setting $C(FIFO) = J \times A^*$ where J is a set of indices, which could also be a product set. This will be useful in the specification of protocol services.

The above representations have the problem that they require interleaving of enqueueing and dequeueing, whereas these actions are inherently concurrent (unless the queue is full or empty).

Where it is important that enqueueing and dequeueing are concurrent, the FIFO queue can be represented as a circular buffer with pointers to the head and tail of the queue. This representation is not as abstract and is at the level of an implementation. The costs are in the complexity of the representation (you need

137

P-Graph

Declarations

A: Set of Queue Items
A^*: Set of strings over A
ε: Empty string
$x : A, q : A^*$
'.' binary string concatenation operator
$M_0(FIFO) = \{\varepsilon\}$
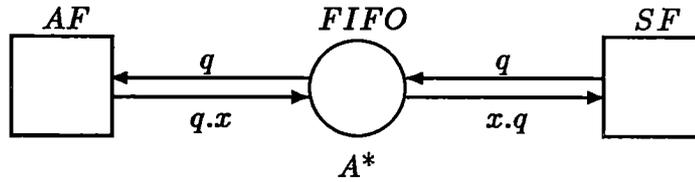
P-Net

$S = \{FIFO\}$
$T = \{AF, SF\}$
$C = \{A^*, A \times A^*\}$
$C(FIFO) = A^*$
$C(AF) = C(SF) = A \times A^*$
$\forall x \in A, \forall q \in A^*, Pre(FIFO, AF; (x, q)) = q$
$\forall x \in A, \forall q \in A^*, Pre(FIFO, SF; (x, q)) = x.q$
$\forall x \in A, \forall q \in A^*, Post(FIFO, AF; (x, q)) = q.x$
$\forall x \in A, \forall q \in A^*, Post(FIFO, SF; (x, q)) = q$
$\forall q \in A^*, K(FIFO; q) = \infty$
$M_0(FIFO) = \{\varepsilon\}$

Figure 9.1: Unbounded FIFO Queue

P-Graph

Declarations

A: Set of Queue Items
A^{n*}: Set of strings over A, length $\leq n$
ε: Empty string
$x : A, q : A^{(n-1)*}$
'.' binary string concatenation operator
$M_0(FIFO) = \{\varepsilon\}$
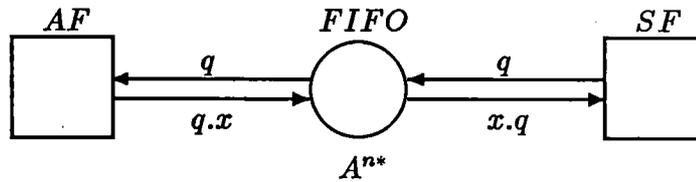
P-Net

$S = \{FIFO\}$
$T = \{AF, SF\}$
$C = \{A^{n*}, A \times A^{(n-1)*}\}$
$C(FIFO) = A^{n*}$
$C(AF) = C(SF) = A \times A^{(n-1)*}$
$\forall x \in A, \forall q \in A^{(n-1)*}, Pre(FIFO, AF; (x,q)) = q$
$\forall x \in A, \forall q \in A^{(n-1)*}, Pre(FIFO, SF; (x,q)) = x.q$
$\forall x \in A, \forall q \in A^{(n-1)*}, Post(FIFO, AF; (x,q)) = q.x$
$\forall x \in A, \forall q \in A^{(n-1)*}, Post(FIFO, SF; (x,q)) = q$
$\forall s \in A^{n*}, K(FIFO; s) = \infty$
$M_0(FIFO) = \{\varepsilon\}$

Figure 9.2: Bounded FIFO Queue

Declarations

A: Set of Queue Items
$N_n = \{0, 1, \ldots, n-1\}, n \in N^+$
$i : N_n, x : A$
$\oplus : N_n \times N_n \to N_n$ modulo n addition
$M_0(FIFO) = \emptyset$
$M_0(PT) = M_0(PH) = \{0\}$
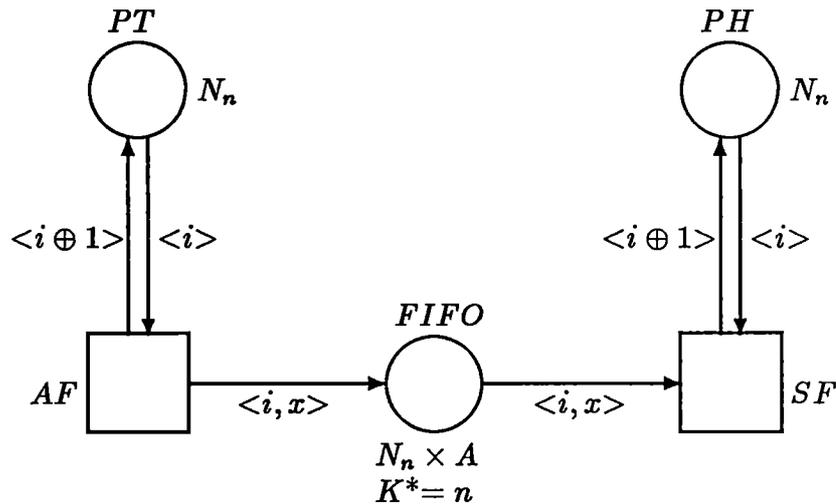
Figure 9.3: Concurrent Bounded FIFO Queue

to keep track of the values of the pointers) and in the increase in the number of states. The situation is depicted in figure 9.3. This example illustrates the use of the total capacity notation, $K^* = n$, to bound the queue.

LIFO Queue

The Last-In-First-Out queue or *stack* is the same as the FIFO queue except that items are added to the head (instead of the tail) of the queue. Thus the pre map for serving is the same as the post map for adding and vice versa. A P-Graph for the bounded LIFO queue is shown in figure 9.4. Transition AL (Add-to-LIFO) adds items to the head of the queue and transition SL (Serve-LIFO) removes items from its head.

With LIFO queues there is contention between adding items to the queue and removing them. Thus it is important that these activities are in conflict, which is the case in figure 9.4. We could build a model of the LIFO that did not involve sequences of items as tokens, but single items as tokens. In this case we would need to have a pointer to the head of the queue, with mutually exclusive access to it for adding and removing. This has the disadvantage of having to store the value of the pointer and leads to a less abstract description.

140

Declarations

A: Set of Queue Items
A^{n*}: Set of strings over A, length $\leq n$
ε: Empty string
$x : A, q : A^{(n-1)*}$
'.' binary string concatenation operator
$M_0(FIFO) = \{\varepsilon\}$
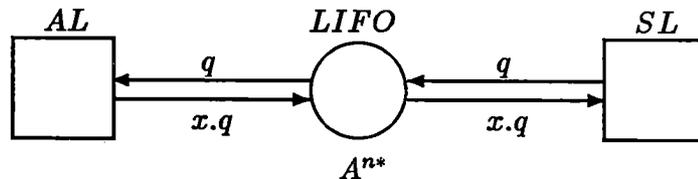
Figure 9.4: Bounded LIFO Queue

9.2 Demon Game

In this section a small example is presented that is being used as a test case for formal methods being developed in ISO and CCITT with application to Open Systems Interconnection protocols and services. The example is called the Demon (Daemon) Game [4].

The following provides a description of the demon game which is slightly more abstract than the narrative description in [4] in that no assumption is made regarding communication. Thus there is no reference to the use of 'signals', as this is considered to be prejudicing an implementation. It is believed that the spirit of the game is still the same!

9.2.1 Narrative Description

Consider a system in which there lurks a demon which generates *bumps*; the number of bumps not being directly observable from outside the system. The aim of the game is to guess when there has been an odd number of bumps generated. The demon informs a player of the outcome of the guess: either *win* or *lose* corresponding to there being an odd or even number of bumps respectively, at the time of the guess. The demon keeps a score which is initially zero. It is incremented by one for a successful guess and decremented by one if unsuccessful. A player can request his score at any time and the result will be returned by the demon.

The game can be played by several players. Before starting a game, a player must log-in. A unique identifier is allocated to a player on logging-in and deallocated on logging-out.

9.2.2 MAN Specification

The Demon Game can be specified using a (strongly-typed) many-sorted algebraic net. It illustrates the use of simple many-sorted unary operators. The game can be specified by 4 places and 5 transitions with their associated inscribed arcs and is given in figure 9.5.

The top two transitions and associated arcs and places specify the behaviour of players logging-in and logging-out. The next two transitions specify how to play the game (guessing the state of the demon's bumps and requesting the cumulative score) and the bottom transition specifies the bumping of the demon.

The convention of double-headed arcs described in section 7.5.4 is used. That is, if the annotation of the arcs associated with the same place and transition are the same $(A(s,t) = A(t,s))$, then both the arcs and the annotations are superimposed, producing a singly annotated arc with an arrowhead at both ends. For example, see $f1 = $ (Scores,Request) and $f2 = $ (Request,Scores) in figure 9.5, where $A(f1) = A(f2) = <i,s>$.

Information about players is represented as a triple comprising: an identifier; the outcome of a guess (including initially the *null* outcome denoting that no guess has yet been made); and a score. This state information is stored as the marking of place *Players*. Unused identifiers are stored in place *IDs*; players' scores in *Scores*; and the state of the demon's bumps in *Bump-state*.

Initially, there are no players (place *Players* is empty); no scores (place *Scores* is empty); all identifiers are available (place *IDs* is marked with the complete set of identifiers *I*); and the demon has not begun to bump. As far as the game is concerned, it is only important to model the state of the bumps as *even* or *odd*; there is no need to count the actual number of bumps. Thus initially there is an even number (zero) of bumps, represented by place *Bumps* being marked with the token *even*.

On logging-in (transition *Login*), a player's state and score is initialised, and his identifier is removed from the unused identifier list. He may now make a guess (transition *Guess*) whereupon his score is updated and he is informed of the outcome. He may also request his score (transition *Request*) or logout (transition *Logout*) with his identifier being returned to the unused list and all information about him being destroyed. The demon bumps whenever it wishes.

9.2.3 Concurrency, Conflict and Interleaving

The bumping is arbitrarily interleaved with players making guesses (a conflict). Similarly, after logging-in, a player may (non-deterministically) make a guess; request his score; or logout (another conflict). This interleaving behaviour is an essential part of the design. For example, it makes no sense to be able to logout and request the score simultaneously. It also makes no sense to guess and bump at the same time or to guess and request the score simultaneously. These situations are naturally in conflict and require interleaving of these events.

142

Declarations

Set of Player Identifiers:I
Set of Game States:G = {win,lose,null}
State of Bumps:B = {even,odd}
Set of Integers:Z
Variables b:B; i:I; g:G; s,r:Z
Functions
Complement $^{-}$:B→B where
$\overline{even}$ = odd and $\overline{odd}$ = even
Score S:B→ {−1, 1} where
S(even) = −1 and S(odd) = 1
Outcome O:B→G where
O(even) = lose and O(odd) = win
M_0(IDs) = I
M_0(Scores) = ∅
M_0(Players) = ∅
M_0(Bumps) = even

Graph
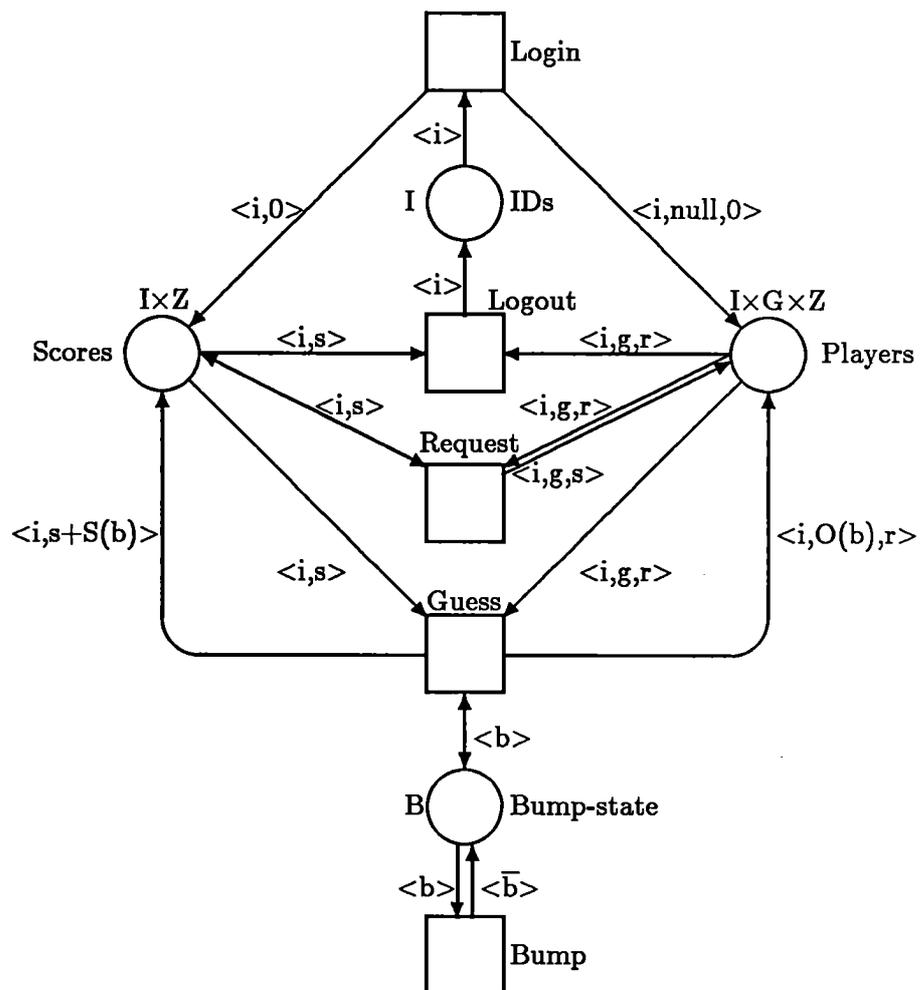
Figure 9.5: MAN Specification of Demon Game

On the other hand, for a particular player, the events of requesting a score or logging in or out, are independent of the demon bumping. Hence transitions *Login* and *Bump*; *Logout* and *Bump*; and *Request* and *Bump* are concurrent.

We would also expect that all players would act independently of one another and this is mostly the case. Any number of players may login, logout or request their scores concurrently but are limited to interleaving when making guesses. Here we have made the assumption that 'read access' to the bump-state is exclusive. This is not essential and it is valuable to delay such decisions to the implementation phase.

This limitation may be overcome by making copies of the Bump-state, and removing all the old ones when the demon bumps (transition *Bump*). Let us assume that there can be n simultaneous accesses to the bump-state, where $n \in N^+$, then setting $A(Bump\text{-}state, Bump) = n <b>$ and $A(Bump, Bump\text{-}state) = n <\bar{b}>$ achieves the desired specification. *Bump* and *Guess* are still in conflict, but *Guess* may occur concurrently with itself limited by n and the number of players logged-on.

These more subtle parts of the design could easily be glossed over with a technique based on interleaving semantics. With an interleaving model, the implementer could be unaware of which parts of the specification were intentionally in conflict and which could be concurrent. Also, the need to specify the number of simultaneous accesses to a resource could be overlooked.
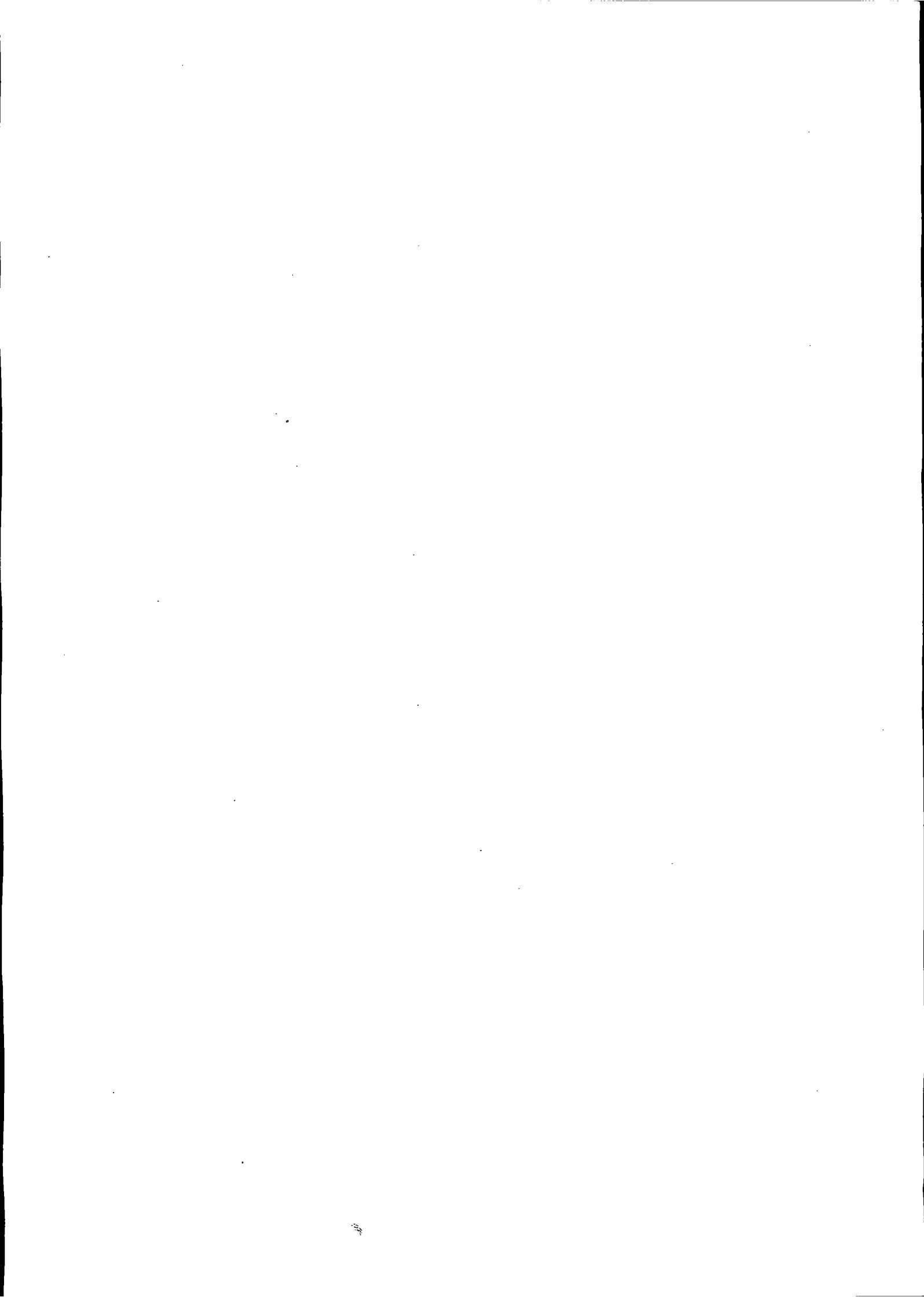
Chapter 10

Cambridge Fast Ring Service Specification

This chapter provides a detailed case study of the use of the P-Graph to specify the service provided by the hardware of the Cambridge Fast Ring (CFR) networking system [78]. The chapter only provides the details of the operation of the CFR that are necessary for the specification. Further details concerning the CFR and relevant protocol architectures can be found in [78,45,44].

The CFR was chosen as the case study for several reasons:

- the service it provides is considered to be reasonably representative of local area and metropolitan area networks;

- it allows the consideration of both point-to-point and broadcast modes;

- it allows for a variety of behaviours including the possibility of messages being lost, out of sequence, and duplicated;

- there was expertise available locally to clarify the intention of the service envisaged for the CFR;

- no previous attempt had been made to provide a formal specification of the service;

- at the time it appeared that it may be useful for the development of the protocols to be used above the basic CFR service;

- the service is simple enough for it to be investigated in some detail without it becoming the major work of the thesis.

An earlier version of this specification has recently been published by the author [26]. This chapter updates that work using P-Graphs instead of Numerical Petri Nets.

```
┌─────────────────────┐  ┌─────────────────────┐
│                     │  │  Logical            │
│                     │  │  Link               │
│     Unison          │  │  Control            │
│     Data Link       │  │  (IEEE 802)         │
│                     │  ├─────────────────────┤
│                     │  │  M-Segment          │
├─────────────────────┴──┴─────────────────────┤
│                                               │
│                   M-Access                    │
│                                               │
├───────────────────────────────────────────────┤
│                                               │
│                   Physical                    │
│                                               │
└───────────────────────────────────────────────┘
```
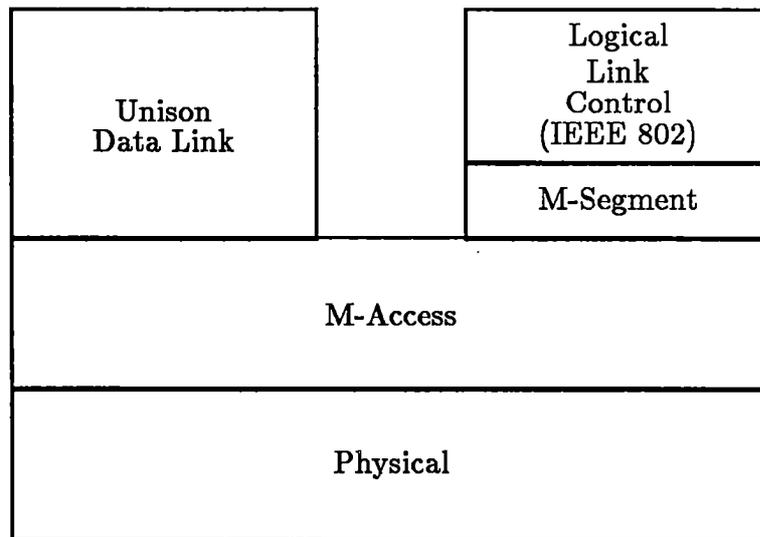
Figure 10.1: Lower Layer Protocol Architectures for the CFR

10.1 Introduction

The Cambridge Fast Ring (CFR) networking system [78] consists of a cluster of CFRs interconnected by bridges. The CFR is a slotted ring designed during the early 1980s to provide a raw 100 MBit/s transmission speed and to substantially increase the bandwidth between point-to-point users. Hardware for the stations, the monitor and bridges for the Cambridge Fast Ring has recently been fabricated in VLSI. The hardware implements the low level protocols between the various distributed components. The task of designing a set of protocols above the basic hardware to provide application services is underway.

An initial draft of the protocol architectures for the CFR has been compiled in [45], where it is shown that different architectures can co-exist above the basic service provided by the CFR hardware. This service is known as the M-Access Service and has been defined in [44]. The protocol architecture is shown in figure 10.1. The lower two layers correspond to the CFR hardware. On the left side is the lowest layer of the Unison architecture that is supported by the CFR, the Unison Data Link Layer. On the right side are the lower layers of an architecture that can support the IEEE 802 and Open Systems Interconnection protocols. The M-Segment layer bridges the gap between the standard Media Access Control (MAC) Service of IEEE 802 and the CFR's M-Access Service. Thus M-Segment and M-Access together provide the MAC service over which the Logical Link Control protocol can be implemented.

The following benefits would accrue from providing an appropriate formal description of the service.

- Ambiguities that arise in narrative descriptions would be removed.

- The formal model could be executed to investigate properties of the service, such as global sequences of service primitives.

146

- The model can provide the basis for the development or synthesis of protocols to be implemented above the M-Access Service, such as the Unison Data Link Protocol [133].

- The model provides the basis for the verification of such protocols once specified.

The chapter is structured as follows. Section 10.2 describes the M-Access Service based on [44] and [78] and discusses some of the assumptions made about the operation of the CFR. Section 10.3 presents the P-Graph specifications and various specification issues are discussed in section 10.4. The final section provides some conclusions.

10.2 CFR M-Access Service

10.2.1 Terminology

In this chapter we shall use the term *packet* to refer to a CFR packet as defined in [78]. The CFR packet includes a Cyclic Redundancy Check (CRC) to detect transmission errors. The term M-Access Service Data Unit (M-SDU) will be used to describe data that is transparently exchanged between users of the M-Access Service.

10.2.2 Features of M-Access

A draft description of the M-Access Service is given in [44]. The service provides an abstract view of the features of a ring cluster: a set of rings interconnected by gateways. The main concern of the M-Access Service is to transfer messages between hosts connected to the CFR. From a user's perspective the ring cluster provides the following facilities:

- Error protected communications paths;

- Fixed slots of 32 octets in which to transmit messages (i.e. user data);

- A routing mechanism by which slots can be routed to their destinations, given that a 16-bit address is provided by the user.

- Two types of communication service:

  1. Point-to-point, where the address indicates the particular destination; and

  2. Broadcast, where a special address (hex FFFF) indicates that the message is to be broadcast to all other stations on the ring cluster.

- Some buffering comprising a single transmit buffer and a single receive buffer per host and a large number of buffers in bridges (gateways) for smoothing traffic.

- The communications path has the following characteristics

  1. M-SDUs may be lost

  2. M-SDUs may be duplicated (this is a rare event, but possible)

  3. M-SDUs may not be sequenced (this can only happen in an interconnected ring cluster where there is the possibility of two (or more) paths from source to destination).

10.2.3 Service Primitives

In the style of Open Systems Interconnection, service primitives define the communication between the users and the provider of a communications service in terms of

- what is to be transferred across the interface - this is defined by a set of parameters associated with the service primitive and the service primitive type.

- the allowable sequences of service primitives both at a local interface and globally.

- the relationships between the service primitives at each of the interfaces.

The service specification is provided at an abstract level in order to avoid over-specification of the interface between the user and provider.

In the M-Access Service, two service primitives are defined:

- M-DATA request

- M-DATA indication

M-DATA request

This primitive is invoked to initiate the sending of data from one service user to another service user (for point-to-point operation) or to all other service users (broadcast). The primitive therefore has 3 essential information types: the source address; the destination address; and the data to be transferred. (These parameters comprise a M-Access Service Data Unit). The primitive and its associated parameters are represented, as usual, by the following notation:

M-DATA request(source-CFR-address, destination-CFR-address, M-data)

In [44], two further parameters are defined: *retry-control* and *transmit-status*. The retry control parameter is a boolean indicating whether or not retries are allowed by the transmitter of a CFR station (as a mechanism for recovery from lost packets). If retries are not allowed, then the service cannot duplicate M-SDUs. We shall model the retry-control parameter at a higher level of abstraction (perhaps more appropriate for a service specification as retry-control does not

148

involve both users (or all users in broadcast mode)), where we shall allow the service provider to choose any number of retries non-deterministically. Hence any number of retries (including zero) would be allowable in a realisation of the service, and user control of this number on a per M-SDU basis would also be a possibility.

We argue here that the *transmit status* parameter should be removed from the M-DATA request primitive, as the return of its value cannot be considered atomic with the transfer of data from user to provider. This is important from the point of view of defining sequences of service primitives. If a value of *transmit-status* needs to be determined before the M-DATA request can occur, then the corresponding M-DATA indication may have occurred before it! The present time-sequence diagrams [44] quite rightly deny this possibility - so we have a problem. This may be solved by creating a separate TRANSMIT-STATUS indication primitive. This primitive will only occur at a local interface (no global significance) and is normally excluded from service definitions, however, it appears useful to include it in a simplified form as discussed below.

M-DATA indication

This primitive is invoked to receive data sent by a sending user. It complements the M-DATA request primitive. The receiving user has completed the receipt of all the data when the primitive occurs.

The primitive has the same set of parameters as the M-DATA request primitive.

M-DATA indication(source-CFR-address, destination-CFR-address, M-data)

These parameters have the same values as those in the corresponding M-DATA request primitive. (Note: we are not considering address mapping required in multi-cluster configurations. They have also not been considered in [44].)

M-TOG indication

The CFR hardware has the capability of telling a user of the M-Access Service that a transmission of a packet has not succeeded (i.e. the number of retries has been exhausted without a positive acknowledgement). This signal is known as 'Thrown-on-Ground' or TOG for short. A TOG will normally indicate that the receiver is busy. The effect of the TOG is to discard the packet. The user then has the option of retrying the same M-SDU, accepting the loss, or trying another M-SDU to a different destination before retrying sometime later. Hence the TOG signal has important consequences for the way in which the user behaves. It appears to be useful at the service level to explicitly define a primitive to express this characteristic of the service, as a form of notification service. A possible parameterless primitive would be: M-TOG indication, indicating that the current packet is considered lost by the service provider.

10.2.4 Sequences of Service Primitives

Point-to-Point

The sequences of primitives are partially determined by the time-sequence diagrams in [44]. For a particular source-destination pair, the M-DATA indication (at the destination) either follows the corresponding M-DATA request (at the source), or it does not occur at all. What is not covered in [44] is a statement as to how much buffering will be allowed, i.e. how many M-DATA requests can occur, before a M-DATA indication must occur in the case where there is no loss? This is obviously implementation-dependent and an implementation-independent specification must allow for the choice to be arbitrary.

Other important points are that duplicates are possible (although rare) and that the medium does not preserve sequence (but only in CFR clusters where there are multiple paths between source and destination).

Now that we have introduced the M-TOG indication, its affect on the allowable sequences of primitives will need to be defined. This will be done in the formal specification.

Broadcast

The sequences of primitives for a successful broadcast are partially given by the time sequence diagrams in [44]. There are a number of questions that need to be discussed here. The set of M-DATA indications that may arise from a broadcast must occur (if at all) after the originating M-DATA request. However, nothing is stated regarding the sequences of occurrences of the resulting M-DATA indications. This will depend on the ring topology, the delays through the receiver, whether or not the receiver is busy and whether or not retries are allowed when broadcasting. Retries (and hence duplicates) can occur if the source station does not receive its own broadcast because it is busy receiving another packet for example. If no retries are allowed, then the medium cannot duplicate packets (and hence M-SDUs). Out of sequence messages are still a possibility. A reasonable abstraction may be to assume that the occurrences of M-DATA indications are not ordered across the different receiving stations. Although it is possible that M-DATA indications will occur in the order that stations are encountered on the ring, this cannot be guaranteed, due to M-SDUs being flow controlled across the M-Access interface.

[44] appears to suggest that if any M-SDU is lost, all are lost. It is possible however, that a number of stations could receive a packet, while others do not. A M-DATA indication need not occur due to the receiver being busy or because of a transmission error (or other reasons). It appears reasonable to abstract from the ring topology and assume that loss by a particular station is independent of its position on the ring, even though loss due to a corrupted packet would imply that all further destinations on the ring concerned would discard the packet (except in the unlikely event of a packet's CRC being made good due to noise). Because packets can be lost due to the receiver being busy, this position-independence assumption for loss is required as the most general case.

The use of M-TOG indication in broadcast mode is rather problematic. The broadcast protocol appears to work in the following way. A broadcast is initiated by a source by setting the destination address to all ones. If other stations on the CFR are willing to accept packets from the source of the broadcast, they will do so if they are not busy. On detecting that it is a broadcast packet, the CRC is not changed. If the CRC is bad, the packet is not received, but continues on its way around the ring. If the CRC is good the packet is received and sent on its way, again with the same CRC (good). If the sender receives back the broadcast packet with a good CRC, it notes that it is a broadcast packet and no retransmissions are initiated. Hence no TOG will occur. Of course, the broadcast may well have failed to be received by many stations that were busy at the time.

It is possible, however, on a multislot ring, that the sender of the broadcast packet is busy receiving another packet when its broadcast packet returns. In this case, no signal is sent from the receiver to indicate that it was a broadcast packet from itself, and it is treated like a normal packet. The good CRC will indicate that the packet has not been received and should be retransmitted. This could cause a string of duplicate broadcast messages to be received until the retry limit is exceeded. At this stage a TOG signal occurs. The user will find it difficult to use the TOG in any sensible way, as its interpretation can be that any number of the destinations have received the packet and any number of duplicates. It appears to be advisable for the TOG signal to be ignored when broadcast mode is used. Hence, at the service level, the M-TOG indication will not occur.

10.2.5   List of Assumptions

This section summarises the assumptions that have been discussed above.

Point-to-point

1. A M-DATA request must have preceded the occurrence of a corresponding M-DATA indication.

2. The parameters associated with the M-DATA indication are identical to those of the corresponding M-DATA request.

3. Duplication is possible, but only if retries are allowed.

4. In ring clusters, sequencing is not maintained in general.

5. Single M-SDU buffering occurs in the transmitter and receiver of a station in the current CFR implementation. There will also be buffering in any bridges.

6. Loss of M-SDUs is possible and handled in two ways:

   - reported to a user in an M-TOG indication if a retry limit is exceeded; but
   - otherwise not reported to the user.

Broadcast

1. A M-DATA request must have preceded the occurrence of a corresponding M-DATA indication.

2. The parameters associated with the broadcast set of M-DATA indications are identical to those of the originating M-DATA request.

3. Loss of M-SDUs is possible. In general it does not depend on the position of the station on the ring. It is not reported to the user.

4. Duplication is probable on multislotted rings, if retries are allowed.

5. Occurrences of M-DATA indications are not ordered across destination stations.

6. In general misordering is possible.

7. Same buffering as for point-to-point.

8. M-TOG indications do not occur.

10.3 Formal Specification

In this section we will specify various characteristics of the M-Access Service with the P-Graph. The following aspects will be investigated for both Point-to-Point and Broadcast operation.

1. An arbitrary cluster of rings, with unlimited storage.

2. A single CFR with a single M-SDU buffer for sending and a single M-SDU buffer for receiving in each of its stations, corresponding to the CFR implementation.

10.3.1 General Comments

The aim of this section is to start with the most general M-Access Service that may be envisaged and then to refine it towards the actual Cambridge Fast Ring Architecture. It will be assumed that there can be an arbitrary number of stations communicating over the service, where the number is limited by the Source Address space.

10.3.2 Abstracting from CFR slots

At the top level of abstraction we shall assume that any station can access the CFR simultaneously with any other station. This implies no contention over slots. (In a more detailed specification the slot contention could be modelled.)
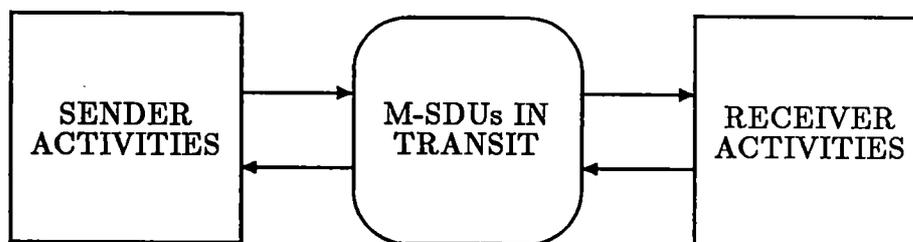
Figure 10.2: Means/Activity Net of CFR M-Access Service

10.3.3 Abstracting from Ring Topology

We shall also abstract from the ring topology by assuming that the service is independent of the positions of the stations around the ring. For ring clusters we shall also assume that the service is independent of the ring to which a station is attached. For example, a station on ring 5 communicating with another station on ring 1 will be treated as identical to stations communicating on the same ring. This is the general case for the CFR and CFR clusters when considering possible global sequences of service primitives as there can be arbitrary delays caused by a receiving station being busy.

10.3.4 Structure

Because CFR stations are built from identical chips, the sending (and receiving) operations in each of the stations are the same. We therefore only need to model a generic sender communicating over the ring with a generic receiver, each parameterised by the station address.

The structure of the CFR M-Access Service is given in figure 10.2. "Sender Activities" and "Receiver Activities" involve the invocation of service primitives associated with sending and receiving data, respectively. The resources are the M-SDUs that are in transit from sender to receiver. It is assumed that M-SDUs are available for the sender and that they are forwarded on to the destination user. Hence the sender produces M-SDUs (via the occurrence of M-DATA requests) but its activity can also be affected by M-SDUs (stimulating the occurrence of an M-TOG indication). The action of receiving an M-SDU (M-DATA indication) requires the presence of an M-SDU in transit and in general removes the M-SDU.

10.3.5 Specification of a CFR Cluster

Implicit Interaction with Users

The Means/Activity Net of figure 10.2 may be refined into a general M-Access Service where the dynamics are specified by the P-Graph of figure 10.3. Implicit interaction with the M-Access Service Users is modelled by the occurrence of service primitives. Explicit interaction with M-Access Service Users is considered in the next section. Service primitives associated with sending are drawn on the

Declarations

Sets: S (Source Addresses), D (Destination Addresses), M (User Messages)
$D' = D \backslash \{b\}$ where b is the Broadcast Address, $b \in D$
Variables: s:S, d:D, d':D', m:M
Initial Marking: M_0(SP-storage) $= \emptyset$

M-TOG indication
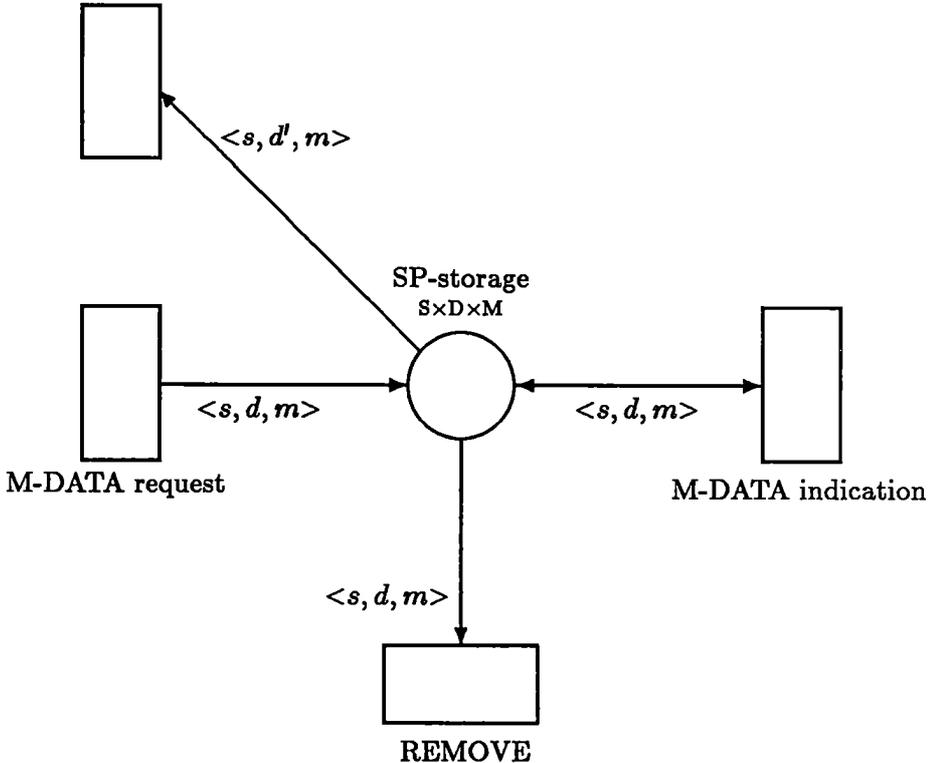
Figure 10.3: Top Level P-Graph of CFR M-Access Service

left side of the diagram, those associated with describing the channel between the sender and receiver are drawn in the centre and those associated with receiving on the right.

The P-Graph comprises four transitions and one place. Three of the transitions model the three service primitives, while the forth models removal of M-SDUs that is not reported to a user. The place 'SP-storage' (Service Provider storage) models an unbounded number of buffers in the service provider and may be regarded as a queue where the item to be served is chosen arbitrarily. 'SP-storage' has a colour set that is the cartesian product of the set of source addresses, S; the set of destination addresses, D; and the set of possible messages, M; and hence may store tokens that are triples. The double-headed arc notation of section 7.5.4 is used.

The following properties of the service provider are modelled.

- Addresses and Messages. S, the set of source addresses, consists of strings of bits of length 16, where a bit $\in \{0, 1\}$, but it excludes the string of 16 zeros, reserved for the monitor address, m, and the string of 16 ones,

reserved for the broadcast address, b. D, the set of destination addresses, comprises all strings of bits of length 16. D' is the set of (real) destination addresses and hence is the same as D without the broadcast address (D' = D\{b}). M, the set of user data messages, are strings of bits of length 256.

- M-SDUs. M-SDUs are modelled as tokens placed in 'SP-storage' (i.e. M-SDU $\in$ S$\times$D$\times$M).

- Service Primitive Parameters. Arc inscriptions consist of triples (either (s, d, m) or (s, d', m)) of variables that are typed appropriately in the declaration of the P-Graph. (We assume that tupling is performed by a primitive operator that does not need to be defined explicitly in the signature.) $s : S$ represents the *source-cfr-address* parameter and similarly d:D and m:M represent the *destination-cfr-address* and *M-data* parameters respectively of the M-DATA primitives. d':D' also represents the *destination-cfr-address* parameter.

- Service Primitive Occurrences. The occurrences of the service primitives M-DATA request, M-DATA indication and M-TOG indication are modelled by the firing of the transitions labelled with the corresponding names. The station at which the service primitive occurs is determined by the address variables in the associated M-SDUs. The M-DATA request and M-TOG indication primitives occur at the *source* address of the associated M-SDU and similarly the M-DATA indication primitive occurs at the *destination* address of its associated M-SDU. The occurrence of the M-TOG indication indicates to the source user that the provider has discarded the M-SDU and believes that it has not been delivered to its destination.

- Arbitrary Number of Buffers within the service provider. In order to allow any amount of storage in the M-Access Service Provider, it is necessary to allow the place 'SP-storage' to be unbounded or to have a finite but indeterminate capacity. We have modelled the unbounded case here as it is easier to represent. This allows a particular implementation to have any number of buffers and still conform to the Service Specification.

- M-SDU Sequencing: The place 'SP-storage' allows arbitrary overtaking of an M-SDU by another M-SDU and hence models the "non-sequence preserving" nature of the service. Note that FIFO order is also a possibility. (If we exclude the possibility of multiple paths between source and destination stations, then FIFO order is preserved. This may be modelled by typing 'SP-storage' with a set of FIFO queues, one for each source-destination pair and changing the arcs and their inscriptions accordingly using the ideas for FIFO queues presented in chapter 9.)

- Arbitrary Loss: Loss not reported to the service user is modelled by the occurrence of the "REMOVE" transition. The REMOVE transition serves two purposes. It indicates genuine loss (as already mentioned) but it is also used to remove an M-SDU from the service provider after it has been delivered to the receiver, as described in the next two items.

- Normal Transfer: M-SDUs are placed in 'SP-storage' on the occurrence of an M-DATA request. The values for the parameters are chosen arbitrarily from the domains of the variables. Any number of M-DATA requests may occur, initiated by any station and destined for any station. So long as there is an M-SDU in 'SP-storage', an M-DATA indication may occur. The M-SDU is retained to allow for possible duplication or for broadcast (see below). Normal transfer is modelled by the occurrence of the M-DATA indication transition followed by the occurrence of the REMOVE transition for the same M-SDU, without any intervening occurrence of another M-DATA indication for the same M-SDU.

- Hate List and Select Register: The possibility exists for a destination not to receive M-SDUs from a source on the *hate list* or when the receiver uses the *select register*. The hate list is a list of sources stored by a destination. The destination will not receive any M-SDUs from the listed sources. The select register allows a receiver to choose from 3 possibilities: receive from nobody; receive from a single source; or receive from everybody. For a source on the hate list of a destination, this corresponds to sequences in which M-DATA requests for the particular source-destination pair are always followed by either a REMOVE event or a M-TOG indication, but not by a M-DATA indication. If the select register is set to receive from nobody for destination b, then the same applies for all source-b pairs. If destination b selects source a, then the same applies for all source-b pairs except (a, b) (unless a has been placed on the hate list!). If receive from everybody is selected, then all sequences are allowed (modulo the hate list).

- Duplication: Duplication is modelled by the occurrence of the M-DATA indication transition 2 or more times for the same M-SDU.

- Broadcast: This is indistinguishable from duplication except that the 'destination' address parameter must be the broadcast address value, b. The individual destination address for each occurrence of the M-DATA indication for the broadcast is not known at this level of abstraction. The next section details how this information may be incorporated by a refinement of the specification in figure 10.3. Duplication of broadcast M-SDUs is allowed. If duplication is not intended (as is the case with the CFR) it may be removed in a further refinement as shown in a further section.

- M-TOG indication. The occurrence of this transition for a particular M-SDU prevents any further occurrences of the M-DATA indication for the same M-SDU (by removing it from the queue), and indicates to the source that the service provider believes (rightly or wrongly) that the M-SDU has been discarded. The main reason for this in the CFR is that the receiving station is busy and has refused to accept the M-SDU (on a number of occasions determined by the retry limit). We have deliberately forbidden the occurrence of an M-TOG indication for a broadcast M-SDU, by using the variable d':D' in the arc inscription, instead of d:D. (This makes use of subtyping in the P-Graph.) An alternative would have been to use d:D and to associate the condition $d \neq B$ with the M-TOG indication transition. This was the approach used in [26].

- Retry control. In this specification retry control is handled implicitly and non-deterministically. The retries are the mechanism for duplication. Duplication can only occur if retries are allowed, but it may not occur even if retries are allowed. As far as the occurrence of service primitives is concerned, the number of retries is not relevant, and is invisible to the users. One important factor to users is the number of duplicates and that they can be limited to zero by retry prevention. The present specification models arbitrary duplication. This is more general than the Cambridge Fast Ring, where the number of duplicates is bounded by the retry limit. A more detailed specification can be given to accommodate this limit by explicitly modelling a retry control parameter which passes the limit to the M-Access Service provider.

- Quality of Service. Quality of service parameters have not been included in [44] and have thus been ignored in this specification. The Retry Control parameter may be considered as an implementation-dependent QOS parameter, as it affects a) the transfer delay, b) the probability of M-SDU loss and c) the probability of duplication. In a service specification it is important to abstract away from implementation decisions. This is why the present service specification does not include the retry control parameter. It is considered inappropriate at the service level of specification.

- Ring Broken. It appears to be useful to include in the definition of the service a *Ring-Broken* primitive. This has not been modelled as again it does not form part of the M-Access Service definition in [44]. Given that the effect of a broken ring is to lose M-SDUs and possibly to allow for duplicates, the present specification does model this behaviour without the introduction of a specific primitive.

Explicit Interaction with Users

The specification of figure 10.4 shows how the service interacts with its users and also specifically indicates the destinations that receive M-SDUs as a result of a broadcast. Five places (and associated arcs) have been added, three for the source user and two for the destination user. It is quite arbitrary whether or not any, all or none of the destinations receive a broadcast M-SDU.

Each station's source has a set of messages stored in place 'Messages' which it wishes to transmit to any one of a set of destinations. The source may also wish to broadcast the message. The broadcast and destination addresses are stored in place 'Destinations'. When a M-DATA request occurs, an M-SDU is formed from the message and the particular source-destination pair, and stored in the M-Access service provider. This M-SDU may now be lost (transition 'REMOVE' occurs); discarded by the service provider while informing the source ('M-TOG indication' occurs); or it may be delivered to an allowed destination ('M-DATA indication' occurs). The M-TOG indication may not occur for a broadcast M-SDU. When it does occur, the discarded M-SDU is saved in the place 'Lost-SDUs'. (This is more general than that defined for the CFR. It would only give an indication to the user that an M-SDU had been lost without returning the

Declarations

Sets: S,D,D',M
Constants: $b \in D$
Variables: s:S, d, e:D, d':D', m:M
Initial Marking
M_0(SP-storage)=M_0(SDUs-received)=M_0(Lost-SDUs)=$\emptyset$
M_0(Messages)$\in \mu$(S×M)
M_0(Destinations)$\subseteq$ S×D
M_0(Acceptable-sources)$\subseteq$ S×D'
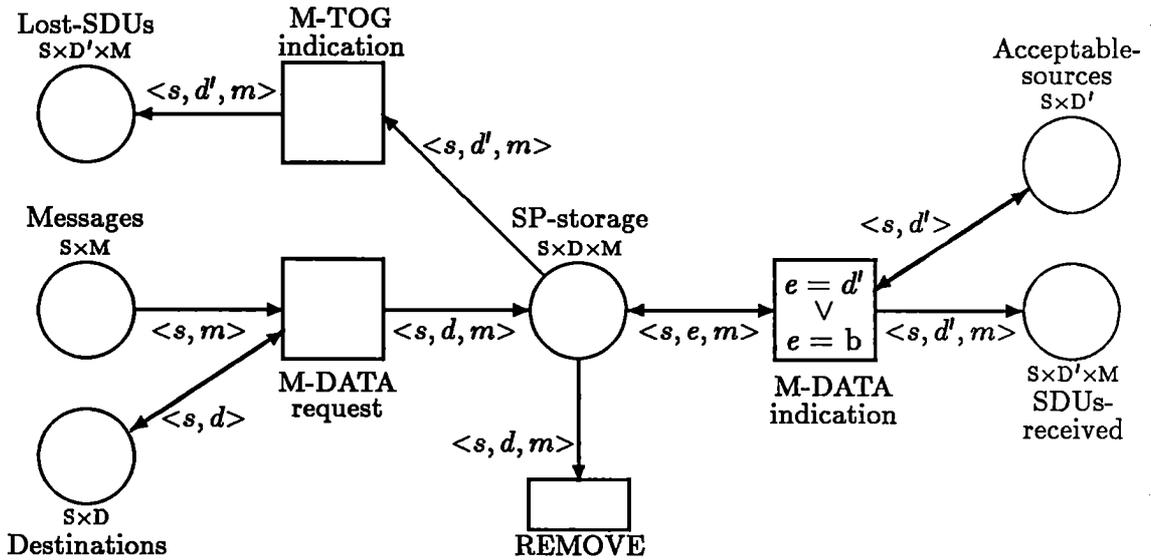
Figure 10.4: CFR M-Access Service: Explicit interaction with users

actual M-SDU.) Any number of M-DATA requests may occur concurrently from any number of stations.

Each destination is prepared to receive messages from a set of sources as determined by its hate list and select register. The acceptable sources are stored in the place 'Acceptable-sources'. If the source address of an M-SDU in 'SP-storage' is on the list of acceptable sources, the M-DATA indication may occur and the M-SDU is passed to the destination and stored in 'SDUs-received'. Duplication is allowed by the M-SDU remaining in 'SP-storage'. If it is a broadcast M-SDU, then the source address must still be acceptable to the destinations that receive it. Two points should be made regarding broadcast.

1. A destination station which receives the broadcast M-SDU is now identified.

2. Arbitrary duplication of broadcast M-SDUs is allowed to each destination that finds the source acceptable. If the specification is to be restricted to disallowing duplicates when broadcasting then a more complicated specification results. The details are presented in the next section.

158

Declarations

Sets: S,D,D',M
Constants: $b \in D$
Variables: $s{:}S$, $d{:}D$, $d'{:}D'$, $m{:}M$
Initial Marking
M_0(M-SDU-reception)=$\emptyset$
M_0(In-transit-M-SDUs)=$\emptyset$
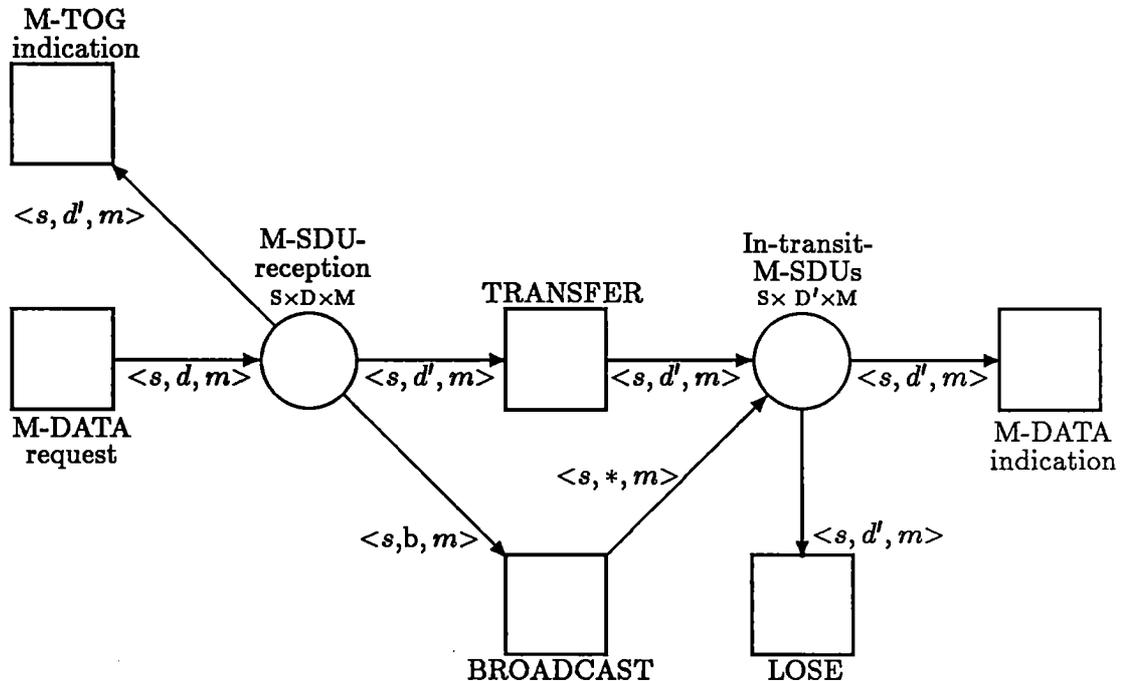
Figure 10.5: M-Access Service: No Duplication

No Duplication when Broadcasting

In this section a specification of the CFR M-Access service is developed where no duplication occurs in broadcast mode. We will not include interaction with the users explicitly, as this can be done in exactly the same manner as in the previous section.

We assume that the broadcast to each receiving station is not ordered and that any number of stations may not receive the broadcast. We firstly consider the simplest situation where there is no duplication for point-to-point. (This may be regarded as close to the initial expectation of the service to be provided by the CFR.)

The specification is given in figure 10.5. It has been necessary to refine the service provider storage into two places: 'M-SDU-reception' which stores M-SDUs of the initial M-DATA request; and 'In-transit-M-SDUs' which stores all possible broadcast and point-to-point M-SDUs. (This has been done to ensure that M-TOG indications may only occur for point-to-point M-SDUs.) Point-to-point M-SDUs are simply transferred from 'M-SDU-reception' to 'In-transit-M-SDUs' by transition 'TRANSFER'. The transition 'BROADCAST' converts a broadcast M-SDU into a set of M-SDUs, one for each possible destination. (This has been

159

Declarations

Sets: S,D,D$'$,M
Constants: b $\in$ D
Variables: s:S, d:D, d':D$'$, m:M
Initial Marking
M_0(M-SDU-reception)=$\emptyset$
M_0(In-transit-M-SDUs)=$\emptyset$
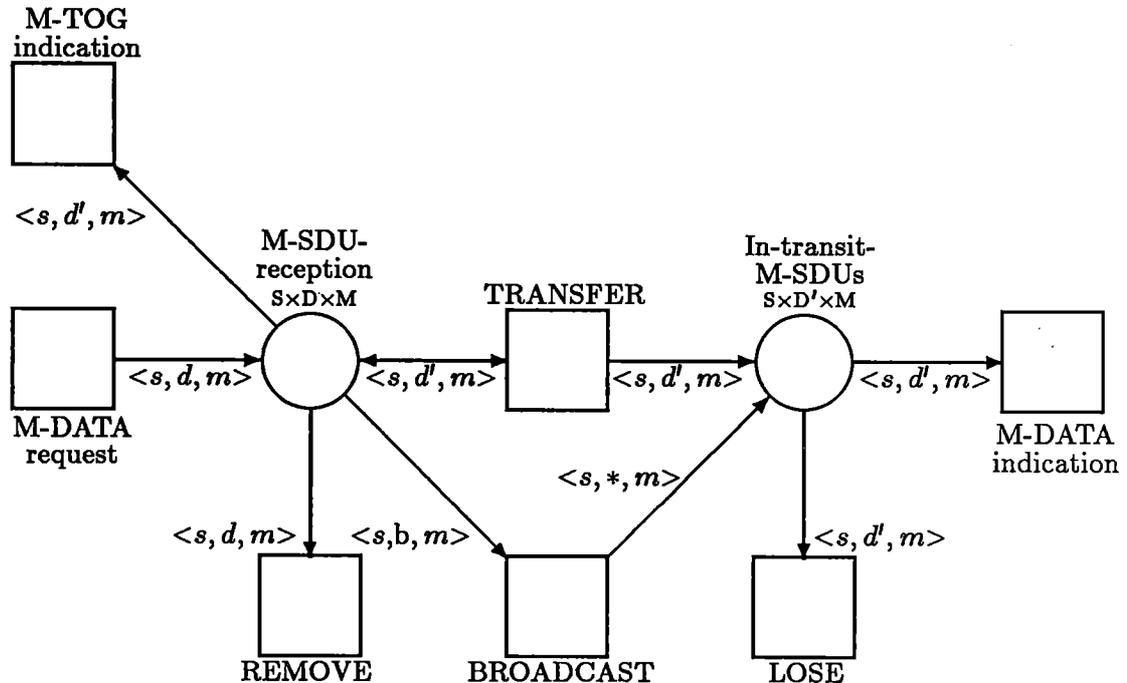
Figure 10.6: M-Access Service: No Duplication for Broadcast

achieved using the '*' notation for sums - see section 7.9.) Any M-SDU may be lost (transition 'LOSE') or successfully delivered to its destination ('M-DATA indication').

In order to allow for duplication of point-to-point M-SDUs, we retain a copy of the M-SDU in 'M-SDU-reception' and allow any number of duplicates by successive firing of 'TRANSFER'. The situation is depicted in figure 10.6. An extra transition is included to allow the M-SDU to be removed from the provider. If (for a particular source-destination pair) the REMOVE transition occurs before the occurrence of TRANSFER or BROADCAST, then it also models loss.

CFR M-TOG indications

The above specifications allow a M-TOG indication to occur any time after a M-DATA request has occurred, so long as the M-SDU remains in 'SP-storage' (figure 10.3) or 'M-SDU-reception' (figures 10.5 and 10.6). This allows any number of M-DATA requests to have occurred at a particular station, before an M-TOG indication occurs which relates to any one of the previous M-DATA requests.

This is more general than the situation that exists in the CFR implementation, where only single buffering is provided in each station for the transmission of M-

SDUs. Thus after a M-DATA request, a M-TOG indication must occur before the next M-DATA request, if it occurs at all. In other words, for a particular CFR station, the M-TOG indication relates to the M-DATA request that immediately preceded it. Thus a strict order is imposed.

This may be specified in the P-Graphs of figures 10.5 and 10.6 by introducing a place (and associated arcs) that restricts the capacity of 'M-SDU-reception' to one M-SDU per source station. This may also be conveniently abbreviated by using the capacity notation developed in section 7.10. This construction is illustrated in the next section.

In the original design of the CFR, the idea of double buffering (i.e. allowing two M-SDUs to be stored in the transmit and receive FIFO buffers) was considered. This would allow two M-DATA requests to have occurred before the M-TOG indication occurred for the first M-DATA request. This can also be modelled by using the bounded FIFO queue discussed in chapter 9 with its length restricted to two. A queue would be needed for each source-destination pair.

10.3.6 Single CFR Specification

Each station in the CFR has two buffers: one for sending M-SDUs and the other for receiving M-SDUs. Each buffer has the capacity for just one M-SDU.

The more general case of unlimited storage was specified in the previous section. It is applicable to CFR clusters where bridges can have large numbers of buffers. We now turn our attention to a single CFR.

We refine the P-Graph of figure 10.6 to the specific case of single buffering for the CFR. We shall only consider the case of implicit interaction with the users. Explicit interaction can be added trivially, in a similar way to that shown in figure 10.4.

We shall consider the following characteristics of a single CFR:

- Arbitrary number of stations

- Point-to-point and broadcast modes

- Single transmit buffer and single receive buffer for each station

- Sequence of M-SDUs preserved per source-destination flow

- Single broadcast by each station (only one broadcast per station is allowed at any one time due to the single transmit buffer)

- Arbitrary loss of M-SDUs

- Three modes of duplication:

  1. Arbitrary duplication in both point-to-point and broadcast mode;
  2. No duplication in broadcast mode, but arbitrary duplication for point-to-point operation; and

3. No duplication

The duplication case 2 is close to the operation of the CFR, although duplication for point-to-point is very rare and limited. A limit to the amount of duplication can be incorporated into the specification in a straightforward way if desired. (It requires an extra place to store the duplication limit for each station.)

We shall consider the three modes of duplication in separate specifications. As usual, the left side of each diagram represents the transmitter and the right side the receiver. The transitions in the centre represent various ways in which the CFR can operate. We represent a set of transmit buffers, one for each station, by the single place 'Transmit-buffers' and we record the stations that have empty buffers in place 'Empty-transmit-buffers'. (This is the same as the control place for determining the capacity of M-SDU-reception mentioned above.) A similar situation exists for the receive buffers. We also include explicitly which stations are acceptable sources of M-SDUs for each of the destinations, by storing them in place 'Acceptable-sources' as we did in figure 10.4.

Arbitrary Duplication

The single CFR M-Access service with arbitrary duplication in both broadcast and point-to-point modes is specified in figure 10.7. The initial state of the service is specified by the initial marking of the net. Each station connected to the CFR will have an empty buffer for transmitting and one for receiving. The presence of an empty transmit buffer is represented by storing the station's source address in place 'Empty-transmit-buffers' and the presence of an empty receive buffer is similarly represented by storing the station's address in place 'Empty-receive-buffers'. The monitor is always attached to a ring, but cannot transmit normal packets [78]. It can, however, receive normal packets. This is why it is excluded from the set of source addresses, but included in the set of destination addresses. Since the monitor is always attached to an operational ring, its address must be included in the initial marking of place 'Empty-receive-buffers'. The addresses of the source stations acceptable to each destination are stored in place 'Acceptable-sources' as source-destination pairs. Initially all the transmit and receive buffers are empty and hence places 'Transmit-buffers' and 'Receive-buffers' are empty.

With this initial state, any number of stations may request the sending of an M-SDU. This is achieved by firing transition 'M-DATA request'. A token representing an M-SDU, is placed in 'Transmit-buffers' and the token representing that the buffer was empty for that station is removed from 'Empty-transmit-buffers'. If the M-SDU is not broadcast, then one of three events may occur:

1. The M-SDU is successfully transferred to the chosen destination. This may only occur if the source is acceptable to the destination. This is achieved by firing transition 'TRANSFER'. A copy of the M-SDU is maintained in the transmit buffer while it is transferred to the destination's receive buffer which is removed from the list of empty buffers. The M-SDU may then be removed from the transmit-buffer which would then be marked free by the

Declarations

Sets: S,D,D′,M
Constants: b,m ∈ D
Variables: s:S, d:D, d':D′, m:M
Initial Marking
M_0(Transmit-buffers)=M_0(Receive-buffers)=∅
M_0(Empty-transmit-buffers)⊆ S
M_0(Empty-receive-buffers)=M_0(Empty-transmit-buffers)∪{m}
M_0(Acceptable-sources)⊆ S×D′
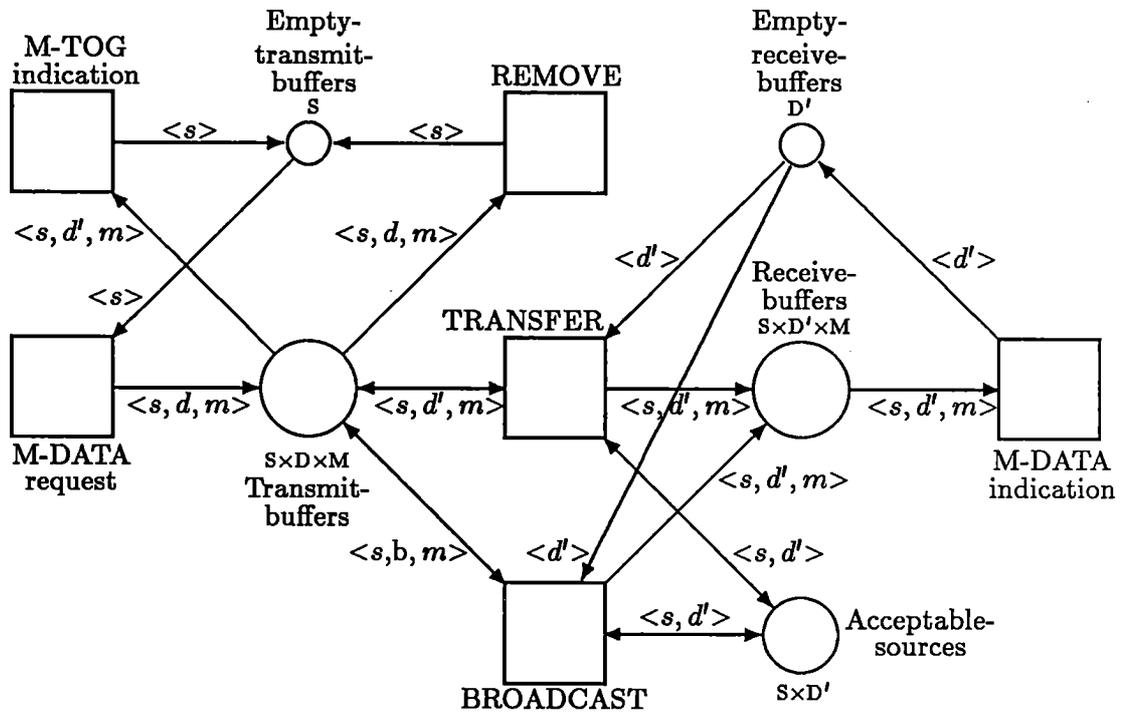
Figure 10.7: Single CFR M-Access Service: Duplication

occurrence of transition 'REMOVE'. Concurrently, an M-DATA indication may occur at the destination, with the M-SDU being removed from the receive-buffer which is marked free. This may be considered as the normal operation of the service. Duplication may occur by firing 'TRANSFER' twice (or more) before the occurrence of the 'REMOVE' transition.

2. The M-SDU is refused by the destination and this is reported to the source user. This is achieved by firing 'M-TOG indication', which removes the M-SDU from the transmit buffer and marks it free.

3. The M-SDU is lost. The CFR transmitter hardware falsely believes that the M-SDU has been accepted by the destination, due to a CRC error in the return path. This is represented by the firing of the 'REMOVE' transition. The M-SDU is discarded and the transmit buffer marked free.

For broadcast M-SDUs, there are two possibilities.

1. The M-SDU is lost by firing transition 'REMOVE'.

2. The M-SDU is broadcast one at a time to any of the allowable destinations by repetitively firing transition 'BROADCAST'. When this transition occurs, a copy of the M-SDU is retained in the transmit buffer, the M-SDU is transferred to an accepting destination and its buffer is removed from the empty list. An M-DATA indication may then occur with the consequent release of the receive buffer. This then allows duplication of the broadcast M-SDU, as the 'BROADCAST' may occur again for the same destination. It may also occur again for any other destination. The broadcast ends with the occurrence of the 'REMOVE' transition, which empties the transmit buffer.

The specification of figure 10.7 could be made more compact by folding transitions 'TRANSFER' and 'BROADCAST' using the Transition Condition '$e = d' \vee e = b$' and changing the inscription of the arc from place 'Transmit-buffers' to $<s, e, m>$ for the new transition. Exactly the same procedure has been followed in figure 10.4 (see transition 'M-DATA-indication'). This has not been done so that point-to-point and broadcast modes are clearly separated as this helps with the development of the specifications in the next two subsections.

No Duplication in Broadcast mode

In order to avoid duplication in broadcast mode we must keep a record of the stations to which we have broadcast. In a single CFR this is relatively easy as no simultaneous transmissions by a particular station are allowed due to single buffering. For each station, only a single point-to-point or broadcast transmission is possible and this must have completed (successfully or not) before the next transmission can occur. This allows us to use the list of allowed source-destination pairs stored in 'Acceptable-sources' to determine which station has received a broadcast M-SDU.

Declarations

Sets: S,D,D',M
Constants: b ∈ D
Variables: s:S, d:D, d':D', m:M
Initial Marking
M_0(Transmit-buffers) = M_0(Receive-buffers) = M_0(Broadcast-destinations) = ∅
M_0(Acceptable-sources) ⊆ S×D'
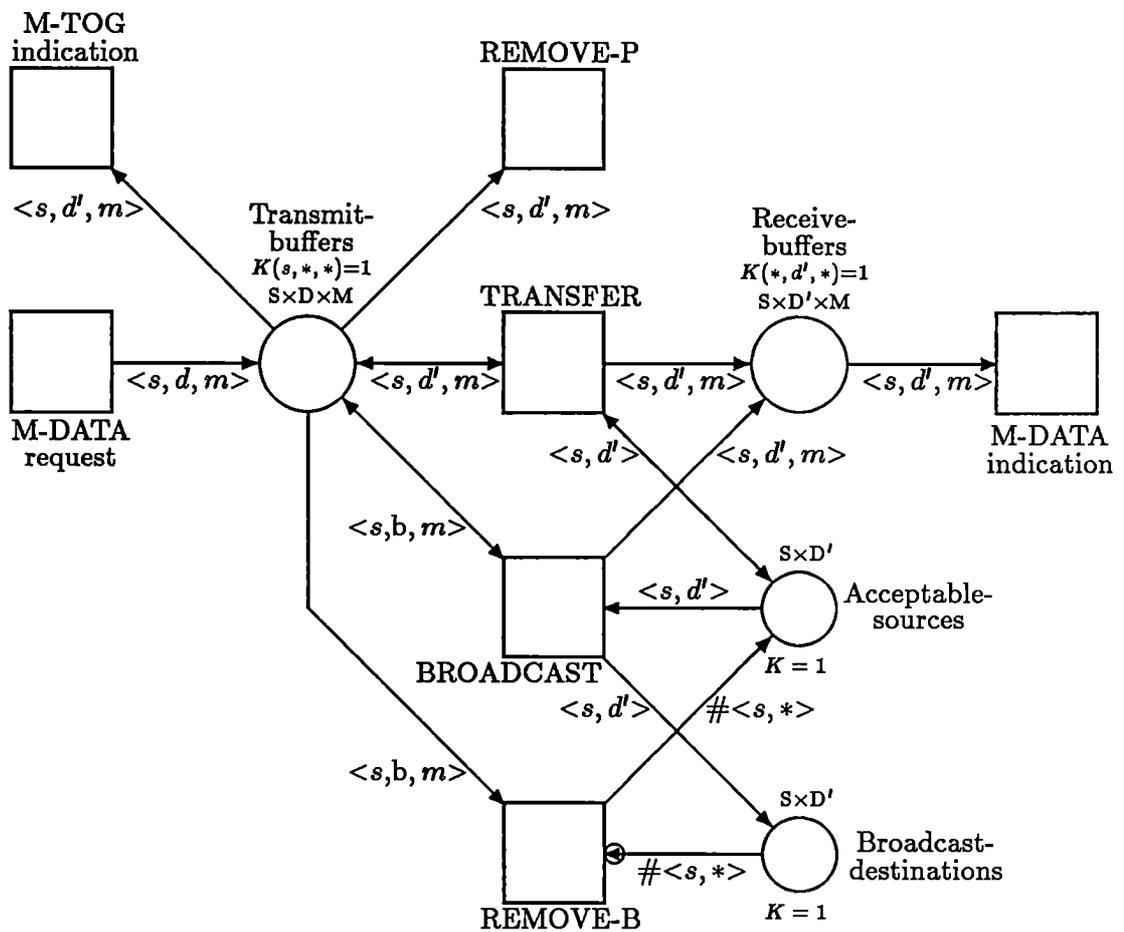
Figure 10.8: Single CFR M-Access Service: No Duplication for Broadcast M-SDUs

The specification is shown in figure 10.8. It is the same as figure 10.7, except that

- The places 'Empty-transmit-buffers' and 'Empty-receive-buffers' and their associated arcs and initial markings have been removed and replaced by the extended capacity notation defined in section 7.11.

- The place, 'Broadcast-destinations', (with initial null marking), the transition, 'REMOVE-B', and associated arcs and inscriptions have been added.

- The 'REMOVE' transition has been renamed 'REMOVE-P'. 'REMOVE-P' may only remove point-to-point M-SDUs as the tuple annotating the arc now contains the variable d':D', instead of d:D. 'REMOVE-B' may only remove broadcast M-SDUs.

- The *return* arc from transition 'BROADCAST' to 'Acceptable-sources' has been deleted.

- Both 'Acceptable-sources' and 'Broadcast-destinations' have been annotated with a capacity '$K = 1$'.

The specification is the same as figure 10.7 for point-to-point operation. As before, broadcasting may occur when a broadcast M-SDU is in a transmit buffer and there is a destination (with a free buffer) that will accept M-SDUs from the source of the broadcast. When 'BROADCAST' fires, the destination is removed from the set of accepting destinations stored in 'Acceptable-sources', and is written to a set of destinations that have received a broadcast M-SDU. The set is stored in place 'Broadcast-destinations'. The broadcast will continue until either the set of accepting destinations is exhausted (there will no longer be a source-destination pair in 'Acceptable-sources' with the broadcast source address - hence 'BROADCAST' will not be enabled (for this source address) and the only remaining possibility for the broadcast M-SDU is that it is removed from the transmit buffer by firing 'REMOVE-B') or the M-SDU is removed by firing 'REMOVE-B'.

'REMOVE-B' is enabled by a broadcast M-SDU being in a transmit buffer. When it fires, the following actions occur atomically:

1. A particular source's broadcast M-SDU is removed from the transmit buffer.

2. All destinations that have successfully received the source's broadcast are purged from 'Broadcast-destinations' and returned to 'Acceptable-sources'.

Any number of stations can be active at the same time and they operate independently except for contention (conflict) for destination receive-buffers.

To be able to purge 'Broadcast-destinations' of the required destinations for the associated source, we have used the notation for purging a member of a partition developed in section 8.6. The addition of the capacity restriction ($K = 1$)

166

to places 'Acceptable-sources' and 'Broadcast-destinations' better reflects the intent of the specification and guarantees that the P-net to CP-net transformations can be applied. This specification is also interesting in that it obeys the restrictions that are necessary for the P-net to CP-net transformation to preserve concurrency (theorem 6.4). This is because BROADCAST and REMOVE-B are in conflict for a particular source of M-SDUs and REMOVE-B has no self concurrency.

No Duplication

The single CFR M-Access service with no duplication can be derived from figure 10.8 by deleting the (return) arc from 'TRANSFER' to 'Transmit-buffers'. When 'TRANSFER' fires, the M-SDU is removed from the transmit buffer and hence no duplication can occur. It would also be useful to rename the REMOVE-P transition to LOSE-P as it would only model loss.

We have made the assumption that as far as users of the M-Access Service are concerned, the operation of delivery of an M-SDU to a receiving station and the freeing of the transmit buffer can be considered atomic for point-to-point operation.

10.3.7 Modelling Slot Contention

The above specifications of the M-Access service for a single CFR are more general than that provided by the CFR hardware. The specifications allow there to be simultaneous transmissions (point-to-point or broadcast) by all stations. The CFR's hardware only permits there to be n simultaneous transmissions, where n is the number of slots on the ring. In most practical CFR installations, the number of stations will be greater than the number of slots.

It is a relatively simple matter to model slot contention in a single CFR. All that is needed is a common side place for the transitions that free transmit buffers and another for transitions that write to receive buffers. Both side places would be initially marked by the number of ring slots. For example, in figure 10.8, one side place would have input and output arcs to transitions M-TOG indication, REMOVE-P and REMOVE-B, while the other side place would have input and output arcs to transitions TRANSFER and BROADCAST. As far as the users are concerned, only n transmit (receive) buffers can become available at any one time.

In many ways this is an implementation issue that it is preferable to suppress in a service specification.

10.3.8 Notification Service

The above specifications have included the M-TOG indication primitive as a form of notification service. Of course it is possible not to provide this service by not informing users when the transmit hardware believes that an M-SDU

has been lost. This can easily be modelled by deleting the 'M-TOG indication' transition and its associated arc from the above specifications.

10.4 Discussion

There are a number of general specification issues that are worth discussing in the context of the above specifications. These include finite delay and progress properties, fairness, and conformance to service specifications.

10.4.1 Finite Delay

The specifications presented above say nothing about the time it takes before a transition fires after it is enabled, the enabling time. This is because nets have abstracted away from time. Hence the enabling time could be anything from zero to infinity. An important property that we would like to preserve in our models is that given a M-DATA request at some point in time either an associated M-DATA indication or M-TOG indication or a LOSS event occurs some bounded time later.

We define a LOSS event as either the occurrence of a LOSE transition, or the occurrence of a REMOVE transition that has not been preceeded by one of a TRANSFER, BROADCAST or M-DATA indication transition for the same M-SDU.

For this to be the case in the net, we must ensure that a transition cannot be enabled indefinitely without firing. This is known as the finite delay property. Another way of looking at this is to consider only those sequences generated by the net where the stop state corresponds to all storage places (e.g. buffers for M-SDUs) being empty. For example in figures 10.3 and 10.4, the stop state is defined by $M_0(\text{SP-storage}) = \emptyset$. Hence for a particular M-SDU, the singleton sequence M-DATA request is excluded. It must be followed (at some stage) by one of the three other possible events mentioned above.

10.4.2 Progress Properties

Another desirable property of the service is that infinite sequences of events must include an M-DATA indication. On the other hand we are quite happy for infinite sequences to exclude the occurrence of either an M-TOG indication or a LOSS event or both. Thus we do not wish the service to be fair to events we would not consider useful.

We would like to guarantee some form of progress property. For example that there exists in every possible sequence, the subsequence "M-DATA request(u,s), M-DATA indication(u,d)" where u=(s,d,m) is a M-SDU comprising the source address, s, the destination address, d, and the M-data parameter, m; and the second parameter defines the station address at which the primitive occurs.

It appears that the CFR does not support such a progress property. For example, it is possible that every station switches its select register to "receive from nobody". In this case, it is not possible for a M-DATA indication to occur.

We may define a quasi-progress property as follows. 'No infinite sequence will contain an infinite subsequence of LOSS events'. This rules out the possibility of loss of M-SDUs occurring infinitely often.

This is probably true in a single CFR, as loss depends on the probability of a transmission error which is much less than one and hence the probability of an infinite repetition of loss events is zero. However, in ring clusters, M-SDUs may be lost for a number of other reasons. Consider the case when a station on one ring wishes to send M-SDUs to a station on another ring. The receiver may not accept an M-SDU for a number of reasons. For example if the source is not selected, the M-SDU will be lost as no signal is passed back to the source for a M-TOG indication to occur. Thus an infinite loss sequence is possible.

The above specifications allow the infinite loss case to occur. This appears to be an accurate description in the case of CFR clusters. In the case of a single CFR it may provide too general a model. To overcome this we could do one of two things:

- constrain the model to exclude the offending infinite sequences. This may be done by introducing an extra place to limit the number of LOSS events to some finite number. Unfortunately this will increase the state space.

- eliminate the offending sequences when analysing the model.

10.4.3 Fairness

In the above section we have mentioned that we are quite happy for the service not to be fair to 'LOSS' events and 'M-TOG indications'. We would be delighted if these events never occurred. Another form of fairness that would probably be desirable is that the service should be fair to each of the stations. By this we mean that we want to disallow the behaviour where a set of stations can be locked out of communication with another station indefinitely by yet another station constantly gaining access to it.

The CFR allows a receiver to select a set of stations (the 'hate list') from which it will not accept M-SDUs, so in general it is not fair. However, given that a source station is not on the hate list, we would like to guarantee that eventually it will succeed. The problem is identical to that described in the previous section. We wish to guarantee that the subsequence 'M-DATA request((s,d,m),s), M-DATA indication((s,d,m),d)' occurs in a infinite sequence containing an infinite subsequence of 'M-DATA request((s,d,m),s)'s. Although allowing for this possibility, the specification does not guarantee this behaviour.

10.4.4 Conformance to Service Specifications

When defining the service it is important to be able to state which sequences of service primitives are essential and which others are optional. More generally one needs to specify that one or more of a set of sequences is mandatory. Thus in order to conform to the M-Access Service, it is necessary that, there exists a sequence in which "M-DATA request((s,d,m),s), M-DATA indication((s,d,m),d)" is a subsequence. It is now debatable whether or not this should be universally quantified over all source-destination pairs. This is probably too strong, as there will be some destinations that do not want every other source to be able to send them data (c.f. the "hate list" in the CFR). However, it does seem reasonable to quantify over source addresses. Thus at the very least, each source must be able to send one M-SDU to one other station. On the other hand, it is obviously not mandatory for the service to include sequences that contain LOSS events. It is also necessary that the language of service primitives of the realisation of the service is a sublanguage of that defined in the service specification.

We could therefore consider figures of merit of conformance to a service specification. For example factors in a figure of merit would be the number of sequences that contained LOSS events, and the proportion of LOSS events in the sequence.

10.5 Summary and Conclusions

The service provided by clusters of Cambridge Fast Rings, known as the M-Access Service, has been specified. The specification has been divided into a set of 'senders' (one for each station) and 'receivers' (one for each station and the monitor), communicating via a queue in the service provider.

An attempt has been made to clarify the present M-Access service definition and care has been taken to itemise the modelling assumptions.

The specification is presented at various levels of detail. In its most general form, the M-Access Service provider can re-order, duplicate or lose M-SDUs that can be transmitted either to a single destination or broadcast to all stations. This allows a very simple model of the behaviour using a CP-Graph (figure 10.3) consisting of just one place (representing a queue of arbitrary size and service discipline) and four transitions (3 representing service primitive occurrences and the forth representing removal of M-SDUs). This specification does not indicate which destinations receive broadcast M-SDUs, only that some broadcast M-SDUs may have been received. In this sense it is incomplete.

At the next level of detail, interaction with users is made explicit (figure 10.4). In particular, the list of each destination's acceptable sources (realised in the 'hate list' and 'select register' of the CFR) is specified and this allows the destinations to which broadcasts are received to be defined. This further detail comes at the expense of 5 extra places and associated arcs.

If the broadcast service is restricted to being duplicate free, then this can be modelled with the addition of one place and 2 transitions and associated arcs (figure 10.5). The addition of a further transition allows the complete service to

170

be duplicate free (figure 10.6). This illustrates the utility of the '*' notation to describe sums of tokens.

A further refinement is presented where the service provided by just a single Cambridge Fast Ring is modelled (figures 10.7 and 10.8). In this specification, the sequence of M-SDUs is preserved and single buffers are modelled for transmitting and receiving M-SDUs (for each station). Duplication and loss are still possible. The list of acceptable sources is included in the specification. The service provider is conveniently modularised into service primitive actions and those associated with its internal operation on M-SDUs: transference (originals or duplicates); broadcast and removal. Further refinements placing restrictions on the amount of duplication are also presented. This has illustrated the use of the capacity notation and that for purging developed in previous chapters.

The specifications are concise (half to one A4 sheet) and allow flow of data to be visualised by executing the net. Sequences of service primitives may also be generated. This allows considerable confidence to be gained in the veracity of the specification.

The specification is also general. It appears highly probable that the particular model developed here could be slightly modified to represent an electronic mail service or connectionless network service for example. It also provides an adequate model for many local area networks of varying topologies (rings, buses, broadcast star networks). The more abstract specification encompassing a number of networks and services would be an interesting application for a P-Graph Schema.

High-level nets allow very general specifications to be modelled quite simply. As these are restricted the models become more complex. The greater the degree of non-determinism and concurrency the simpler the net representation. (This is illustrated in the discussion of modelling CFR slots.) This facilitates stepwise refinement from general specifications to more specific situations.

Limitations of the approach have been indicated. These concern the need to exclude unwanted infinite sequences and involve notions of fairness. This issue is a subject of research within the net community and elsewhere.

The specifications presented here could provide a formal basis for the development or verification of protocols being designed to operate over CFR systems. It is stressed that conformance to a service specification needs to be specified in order to allow systems to be verified and the concept of a figure of merit for the conformance of protocols to service specifications has been canvassed.

This chapter has illustrated that service specifications of networks can be developed with some degree of elegance and visual appeal using P-Graphs.

Chapter 11

Conclusions

11.1 Contribution of Dissertation

There are several significant research results presented in this thesis, covering the development of a specification technique and its application in the protocol domain.

Although the concepts of place capacity and inhibitor arcs are well known for Petri nets (Place/Transition systems), there had previously been little attempt to incorporate them formally into the high-level net framework. The transformation from P-nets to CP-nets is new, based on the idea of complementary places for P/T-systems. A new complementary place invariant for CP-nets is established, generalising a previously well known result for P/T-systems. The transformation allows P-nets to be used for specification in the knowledge that the existing analysis techniques and tools (and those being developed) for CP-nets will be applicable in probably all practical situations.

Conditions under which the P-net to CP-net transformation preserves true concurrency are established. No counterpart of this result previously existed for P/T-systems with inhibitors. The result established here applies to P/T-systems (with inhibitors) as a special case.

Because of its utility in visualising flow of control and data, the P-Graph was developed for describing systems. The P-Graph takes ideas from algebraic specification, that of a many-sorted signature with variables, and includes it in a Petri net structure. The signature allows terms to be generated and these are used to annotate arcs and transitions of an inhibitor net. The signature allows variables to be typed in a manner that suits applications that are naturally many-sorted. Difficulties with Algebraic Nets and Predicate/Transition nets are highlighted. The problems stem from the single-sorted nature of these nets, that partial functions are employed and that variables are typed by a single carrier. This can lead to nets with legal inscriptions that are undefined at a concrete level (causing problems of interpretation when executing the net), and also to behaviour that is not intended by the specifier. These problems are avoided in the P-Graph by the use of a many-sorted signature, appropriately typing variables and the use

of total functions. The P-Graph may be used as a basis for classifying high-level nets, as many different classes result from restricting its structure.

Although the P-Graph was initially developed independently, the approach to its formulation is similar to that taken by Vautherin [143], but it has a number of significant differences. Vautherin stays at the syntactic level (defining a schema), does not type variables in his definition, only allows 'equations' to be associated with transitions, does not include place capacities or inhibitors, and does not allow for natural number terms to be used as multiplicities when defining multisets of terms. I have also defined a P-Graph Schema that has these differences. The differences are important from the point of view of specification as they provide constructs for more concise formal descriptions. An interpretation of the P-Graph and P-Graph schema in terms of P-nets is also provided.

A theory of atomically purging places of all (or subbags of) their tokens has been developed for P-nets. P-nets that only use inhibitors for purging, satisfy one of the two conditions required to ensure that the transformation to CP-nets preserves true concurrency. Some convenient notation for purging is developed for the P-Graph. This involves the notion of a reset arc and the development of a tuple notation for arc inscriptions. Notations for place capacity have also been defined.

Two new applications of the technique to the specification of systems are presented: the Demon Game; and the M-Access Service of the Cambridge Fast Ring. The latter is a detailed case study which illustrates the utility of a number of the features developed for the P-Graph for the specification of communication systems. These include the purging of a member of a partition of a marking and the extended capacity notation.

11.2 Future Work

11.2.1 Analysis

No attempt has been made in this dissertation to present a set of analysis methods for P-nets. Rather, the transformation from P-nets to CP-nets, was developed to allow the analysis methods developed for CP-nets to be applied. These analysis techniques include reachability, invariants and reductions [37], model checking and also analysis via the *skeleton* P/T-system as discussed in [143]. It may also be the case that some of these techniques may be able to be applied directly to the P-net. Direct analysis of subclasses of the P-Graph is possible using invariants as shown in [121] and applied in [66].

11.2.2 Extended Capacity Notation

When developing the extended capacity notation care was taken to give an interpretation in terms of another P-Graph. This meant that the basic P-net definition of the enabling condition was retained.

A different and more general approach has been suggested by Jensen [90], where another *capacity* colour set is associated with each place. Hence we need a *capacity colour function*. If $C_K(s)$ is the capacity colour set associated with place $s \in S$, then we form the set $CAP = \{(s,c) \mid c \in C_K(s), s \in S\}$ and define the capacity to be $K \in \mu_\infty^+ CAP$. We introduce a new function $\kappa : \mu PLACE \rightarrow \mu CAP$ and replace the capacity enabling condition by

$$\kappa(M + Post'(T_\mu)) \leq K$$

The usefulness of this generalisation and its effect on transformations to CP-nets need to be studied.

11.2.3 P-net to CP-net Transformations

The transformation from P-nets to CP-nets employs the idea of complementary places. It may be that there are other more useful transformations. For example, transformations using partial complementary places need to be studied. This is indicated by the MPrT-net and P-Graph of safe train operation presented in chapter 7. The properties preserved by these transformations also need to be investigated.

11.2.4 Applications

A number of examples have been included in this dissertation for illustrative purposes, but they have all been rather small. The aim of the work is that P-nets will be able to be applied to large industrial applications and particularly within the telecommunication and information services sector. This will require ways of structuring and refining specifications. At a fundamental level, the development of refinement morphisms for P/T-systems [107] shows considerable promise. These ideas will need to be incorporated into high-level nets. Of more immediate use at the practical level, the development of hierarchical CP-nets [80] are of interest. This area has not been addressed in this dissertation and it will require a large amount of effort in the future.

11.2.5 Syntax

This dissertation has concentrated on developing the semantics of a high-level net suitable for protocol specification. Some syntactic issues have been explored (the P-Graph; its extended capacity notation; the reset arc and its associated #-tuple notation), but other issues such as typographical conventions for names, markings and capacities and the formal definition of syntax have not been addressed. This would be essential for the development of computer-aided tools to support P-nets and P-Graphs.

References

[1] Design/simulator specification papers. Meta Software Corporation, 1988. Proprietary papers on CPN Palette.

[2] *International Workshop on Timed Petri Nets*, IEEE Computer Society, Torino, Italy, July 1985.

[3] Proceedings of the IFIP WG6.1 international workshops on protocol specification, testing and verification. North Holland. Volumes II to X, 1982 to 1990.

[4] ISO/IEC JTC 1/SC 21/WG1. Proposed draft technical report on the guidelines for the application of Estelle, LOTOS and SDL. ISO/JTC 1/SC 21 S45, December 1988. Also to be published as a CCITT manual by ITU.

[5] I. Araki and T. Kasami. Some decision problems related to the reachability problem for Petri nets. *Theoretical Computer Science*, 3:85 – 104, 1977.

[6] Australia. Reply to questionnaire on Question 2/X. CCITT Study Group X, Question 2/X, Com X - No. 16, October 1985.

[7] J.M. Ayache, J.P. Courtiat, and M. Diaz. Rebus: a fault tolerant distributed system for industrial real time control. *IEEE Trans on Computers*, 31(7):637 – 647, July 1982. Special Issue on Fault-Tolerant Computing.

[8] T. J. Batten. An extension to invariants analysis techniques applied to Petri net models of protocols. *Australian Telecommunication Research*, 21(2):37 – 46, 1987.

[9] E. Battiston, F. De Cindio, and G. Mauri. OBJSA nets: a class of high-level nets having objects as domains. In G. Rozenberg, editor, *Advances in Petri Nets 1988, Lecture Notes in Computer Science 340*, pages 20 – 43, Springer-Verlag, Berlin, 1988.

[10] B. Baumgarten, P. Ochsenschläger, and R. Prinoth. Building blocks for distributed system design. In M. Diaz, editor, *Protocol Specification, Testing and Verification, V*, pages 19–38, Elsevier Science Publishers B.V., 1986.

[11] M. Y. Bearman. 1985. personal communication on testing of OSI Class 2 Transport Protocol using NPN generated test sequences.

[12] M. Y. Bearman. *Formal Specification of the Open Systems Interconnection Transport Protocol Class 2 Using NPNs.* Technical Report 25, CSIRONET, 1986.

[13] M. Y. Bearman, M. C. Wilbur-Ham, and J. Billington. Analysis of Open Systems Interconnection Transport Protocol standard. *Electronics Letters,* 21(15):659–661, July 1985.

[14] M. Y. Bearman, M. C. Wilbur-Ham, and J. Billington. *A Formal Specification of the OSI Class 0 Transport Protocol using NPNs.* Research Laboratories Report 7736, Telecom Australia, Clayton North, 3168, Victoria, Australia, October 1984.

[15] M. Y. Bearman, M. C. Wilbur-Ham, and J. Billington. Some results of verifying the OSI class 0 Transport Protocol. In *ICCC,* pages 597–602, Sydney, November 1984.

[16] M.Y. Bearman, K.R. Parker, and R.A. Berger. *A Formal Specification of the OSI Class 3 Transport Protocol using NPNs.* Technical Report TR-ED-88-02, CSIRO Division of Information Technology, 1988.

[17] G. Berthelot and R. Terrat. Petri nets theory for the correctness of protocols. *IEEE Transactions on Communications,* COM-30(12):2497–2505, Dec 1982.

[18] B. Berthomieu. Methods for carrying proofs on Petri nets using their structural properties. April 1983. to be published in IEEE Transactions on Software Engineering.

[19] B. Berthomieu, N. Choquet, C. Colin, B. Loyer, J.M. Martin, and A. Mauboussin. Abstract Data Nets: combining Petri nets and abstract data types for high level specifications of distributed systems. In *Proceedings of the Seventh European Workshop on Application and Theory of Petri Nets,* pages 25 – 48, Oxford, England, 30 June - 2 July 1986.

[20] E. Best and C. Fernandez. *Notations and Terminology on Petri Net Theory.* Arbeitspapiere 195, GMD, January 1986.

[21] J. Billington. Abstract specification of the ISO Transport Service Definition using labelled Numerical Petri Nets. In Harry Rudin and Colin H. West, editors, *Protocol Specification, Testing and Verification, III,* pages 173–185, Elsevier Science Publishers B.V., Amsterdam, 1983.

[22] J. Billington. *Extending Coloured Petri Nets.* Technical Report 148, University of Cambridge Computer Laboratory, New Museums Site, Pembroke Street, Cambridge, England, October 1988.

[23] J. Billington. Extensions to Coloured Petri Nets. In *Proceedings of the Third International Workshop on Petri Nets and Performance Models,* pages 61–70, IEEE CS Press, Kyoto, Japan, 11-13 December 1989.

[24] J. Billington. First year progress report on research into protocol engineering. October 1987. University of Cambridge Computer Laboratory internal report.

[25] J. Billington. *Formal Approaches to Protocol Engineering: A State of the Art Report.* Switched Networks Research Branch Paper 185, Telecom Australia Research Laboratories, Clayton, Vic., 3168, Australia, November 1989.

[26] J. Billington. A High-Level Petri net specification of the Cambridge Fast Ring M-Access service. In *Proceedings of the BCS-FACS Workshop on Specification and Verification of Concurrent Systems*, University of Stirling, Stirling, Scotland, July 1988. Earlier version published as University of Cambridge Computer Laboratory Technical Report No. 121, December 1987.

[27] J. Billington. Many-sorted high-level nets. In *Proceedings of the Third International Workshop on Petri Nets and Performance Models*, pages 166–179, IEEE CS Press, Kyoto, Japan, 11-13 December 1989.

[28] J. Billington. Protocol engineering and nets. In *Proceedings of the Eighth European Workshop on Application and Theory of Petri Nets*, pages 137 – 156, Zaragoza, June 1987.

[29] J. Billington. Specification of the Transport Service using Numerical Petri Nets. In C. Sunshine, editor, *Protocol Specification, Testing and Verification*, pages 77–100, North Holland, Amsterdam, 1982.

[30] J. Billington and D. Kinny. Computer aided protocol engineering. In *Proceedings of the Conference on New Business Applications of Information Technology*, pages 69–73, Melbourne, 26-27 April 1989.

[31] J. Billington, G. Wheeler, and M.C. Wilbur-Ham. PROTEAN: a high-level Petri net tool for the specification and verification of communication protocols. *IEEE Transactions on Software Engineering, Special Issue on Tools for Computer Communication Systems*, SE-14(3):301–316, March 1988.

[32] J. Billington, M. C. Wilbur-Ham, and M. Y. Bearman. Automated protocol verification. In M. Diaz, editor, *Protocol Specification, Testing and Verification, V*, pages 59–70, Elsevier Science Publishers B.V., 1986.

[33] T. P. Blumer and D. P. Sidhu. Mechanical verification and automatic implementation of communication protocols. *IEEE Transactions on Software Engineering*, SE-12(8):827–843, August 1986.

[34] G.V. Bochmann and C.A. Sunshine. Formal methods in communication protocol design. *IEEE Trans. Comm.*, COM-28(4):624 – 631, April 1980.

[35] W. Brauer, editor. *Net Theory and Applications*, Springer-Verlag, Berlin, 1980. LNCS Vol. 84.

[36] W. Brauer, W. Reisig, and G. Rozenberg, editors. *Petri Nets: Applications and Relationships to other Models of Concurrency.* Volume 255 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1987. Advances in Petri Nets 1986, Part 2: Proceedings of an Advanced Course, Bad Honnef, September, 1986.

[37] W. Brauer, W. Reisig, and G. Rozenberg, editors. *Petri Nets: Central Models and Their Properties.* Volume 254 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1987. Advances in Petri Nets 1986, Part 1: Proceedings of an Advanced Course, Bad Honnef, September, 1986.

[38] G. Bruno and G. Marchetto. Process translatable Petri nets for the rapid prototyping of control systems. *IEEE Transactions on Software Engineering*, SE-12(2):346–357, February 1986.

[39] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the Fifth IEEE Symposium on Logic in Computer Science*, Philadelphia, PA, USA, 4-7, June 1990.

[40] H. J. Burkhardt and H. Eckert. Transport service, formal specification; network service, formal specification; transport protocol class 2, formal specification. 1984. early drafts.

[41] H. J. Burkhardt, H. Eckert, and R. Prinoth. Modelling of OSI-communication services and protocols using Predicate/Transition Nets. In Y. Yemini, R. Strom, and S. Yemini, editors, *Protocol Specification, Testing and Verification, IV*, pages 165 – 192, Elsevier Science Publishers B.V., Amsterdam, 1985.

[42] CCITT. *Blue Book.* Volume III, VI, VII, VIII, ITU, Geneva, 1989.

[43] CCITT. Recommendation Z.100 Functional Specification and Description Language. In *Blue Book*, ITU, Geneva, 1989.

[44] A. M. Chambers. CFR M-Access service definition. November 1986. Draft Unison Project Document, Ref: UA008.

[45] A. M. Chambers and D. L. Tennenhouse. Communications architectures for the Cambridge Fast Ring. October 1986. Draft Unison Project Document, Ref: UA004.

[46] G. Chiola, G. Bruna, and T. Demaria. Introducing a color formalism into Generalised Stochastic Petri Nets. In *Proceedings of the Ninth European Workshop on Applications and Theory of Petri Nets*, pages 202 – 215, Venice, Italy, 22-24 June 1988.

[47] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Trans on Programming Languages and Systems*, 8(2):244–263, July 1986.

[48] E.M. Clarke and O. Grümberg. Research on automatic verification of finite-state concurrent systems. *Ann. Rev. Comput. Sci.*, 2:269 – 290, 1987. Also in *Temporal Logic in Computing: A Tutorial*, Department of Computer Science, University of Manchester, April 1987.

[49] E.M. Clarke, O. Grümberg, and M.C.Browne. Reasoning about networks with many identical finite-state processes. In *Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing*, pages 240–248, August 1986.

[50] E.M. Clarke, D.E. Long, and K.L. McMillan. Compositional model checking. In *Proceedings of the Fourth IEEE Symposium on Logic in Computer Science*, Asilomar, CA, USA, 4-8, June 1989.

[51] B. Cohen, W.T. Harwood, and M.I. Jackson. *The Specification of Complex Systems*. Addison-Wesley Publishing Company, Workingham, England, 1986.

[52] J.M. Colom, M. Silva, and J.L. Villarroel. On software implementation of Petri nets and Coloured Petri Nets using high-level concurrent languages. In *Seventh European Workshop on Application and Theory of Petri Nets*, Oxford, England, July 1986.

[53] The SPECS Consortium and J. Bruijning. Evaluation and integration of specification languages. *Computer Networks and ISDN Systems*, 13(2):75 – 89, 1987.

[54] J. P. Courtiat, J. M. Ayache, and B. Algayres. Petri nets are good for protocols. In *ACM SIGCOMM '84 Symposium, Communications Architectures and Protocols*, pages 66–74, Montreal, Canada, June 1984.

[55] B. Cousin, J.M. Couvreur, C. Dutheillet, and P. Estraillier. Validation of a protocol managing a multi-token ring architecture. In *Proceedings of the IX European Workshop on Applications and Theory of Petri Nets*, Venice, Italy, 22-24 June 1988. Fourth paper in volume II.

[56] J.M. Couvreur and J. Martinez. Linear invariants in commutative high level nets. In *Proceedings of the Tenth International Conference on the Application and Theory of Petri Nets*, pages 96 – 114, Bonn, West Germany, 28-30 June 1989.

[57] J. Dähler. PACE: Prototyping, Analysis and Code-generation Environment. 1989. Handout at Tools Exhibition at the 10th International Conference on Application and Theory of Petri Nets, Bonn, 28-30 June 1989.

[58] R. Devillers. The Semantics of Capacities in P/T nets: A first look. In *Proceedings of the 6th Petri Net Workshop on Applications and Theory of Petri Nets*, pages 171 – 190, Espoo, Finland, June 1985.

[59] M. Diaz. Modeling and analysis of communication and co-operation protocols using Petri net based models. *Computer Networks*, 6:419–441, 1982.

[60] M. Diaz. Petri net based models in the specification and verification of protocols. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, pages 135 – 170, Springer-Verlag, Berlin, 1987. Lecture Notes in Computer Science, Vol. 255.

[61] S. Drees, D. Gomm, H. Plünnecke, W. Reisig, and R. Walter. Bibliography of net theory. In G. Rozenberg, editor, *Advances in Petri Nets 1987*, pages 309 – 451, Springer-Verlag, Berlin, April 1987. Latest update is 'Bibliography of Petri Nets 1988', Arbeitspapiere der GMD, No. 315, June 1988, which contains over 2,500 entries.

[62] C. Dutheillet and S. Haddad. Regular Stochastic Petri Nets. In *Proceedings of the Tenth International Conference on the Application and Theory of Petri Nets*, Bonn, West Germany, 28-30 June 1989. Third paper of Supplement to the Proceedings.

[63] H. Eckert and R. Prinoth. A computation-systems based method for automated proving of protocols against services. In Harry Rudin and Colin H. West, editors, *Protocol Specification, Testing and Verification, III*, pages 213–228, Elsevier Science Publishers B.V., Amsterdam, 1983.

[64] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1, Equations and Initial Semantics*. Volume 6 of *EATCS Monographs on Theoretical Computer Science*, Springer-Verlag, Berlin, 1985.

[65] F. Feldbrugge. Petri net tools. In G. Rozenberg, editor, *Advances in Petri Nets 1985*, pages 203 – 223, Springer-Verlag, Berlin, February 1986. Lecture Notes in Computer Science, Vol. 222.

[66] G. Findlow and J. Billington. High-level Nets for Dynamic Dining Philosophers Systems. In *Proceedings of the International Workshop on Semantics for Concurrency, Leicester, UK, 23-25 July 1990*, Springer-Verlag, London, 1990.

[67] G. Florin and S. Natkin. Evaluation based upon stochastic Petri nets of the maximum throughput of a full duplex protocol. In C. Girault and W. Reisig, editors, *Application and Theory of Petri Nets, Informatik-Fachberichte, 52*, pages 280–288, Springer-Verlag, Berlin, 1985. Selected papers from the First and Second European Workshops on Application and Theory of Petri Nets.

[68] R.J. Fone. *An analysis of a Numerical Petri Net Description of ISDN Call Control Procedures*. Research Laboratories Report 7915, Telecom Australia, August 1988.

[69] R.J. Fone. *A Numerical Petri Net Description of ISDN Call Control Procedures*. Research Laboratories Report 7914, Telecom Australia, August 1988.

[70] H. J. Genrich. Predicate/Transition Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and their Properties. Advances in Petri Nets 1986, Part 1: Proceedings of an Advanced Course, Bad Honnef, September 1986*, pages 207 – 247, Springer-Verlag, Berlin, February 1987. Lecture Notes in Computer Science, Volume 254.

[71] H. J. Genrich and K. Lautenbach. System modelling with high-level Petri nets. *Theoretical Computer Science*, 13:109–136, 1981.

[72] H.J. Genrich. Equivalence transformations of PrT-Nets. In *Proceedings of the IX European Workshop on Applications and Theory of Petri Nets*, pages 229 – 248, Venice, Italy, 22-24 June 1988.

[73] C. Girault, C. Chatelain, and S. Haddad. Specification and properties of a cache coherence protocol model. In G. Rozenberg, editor, *Advances in Petri Nets 1987*, pages 1 – 20, Springer-Verlag, Berlin, April 1987. Lecture Notes in Computer Science, Vol. 266.

[74] P. E. Green. Protocol conversion. *IEEE Transactions on Communications*, COM-34(3):257–268, March 1986.

[75] P.E. Green, editor. *IEEE Trans. Com., Special Issue on Computer Network Architectures and Protocols*, pages 624 – 677. Volume COM-28, IEEE, April 1980.

[76] S. Haddad and J.M. Couvreur. Towards a general and powerful computation of flows for parametrized Coloured nets. In *Proceedings of the IX European Workshop on Applications and Theory of Petri Nets*, Venice, Italy, 22-24 June 1988. First paper in volume II.

[77] G. J. Holzmann. Tracing protocols. *Bell Systems Technical Journal*, 64(10):2413–2433, December 1985.

[78] A. Hopper and R.M. Needham. The Cambridge Fast Ring Networking System. *IEEE Transactions on Computers*, 37(10):1214 – 1223, October 1988. Earlier version published as University of Cambridge Computer Laboratory Technical Report, No. 90, June 1986.

[79] P. Huber, A.M. Jensen, L.O. Jepsen, and K. Jensen. *Towards Reachability Trees for High-level Petri Nets*. Technical Report DAIMI PB - 174, Datalogisk Afdeling, Aarhus University, May 1985. (see also Theoretical Computer Science, Vol. 45.).

[80] P. Huber, K. Jensen, and R.M. Shapiro. Hierarchies in Coloured Petri Nets. In *Proceedings of the 10th International Conference on Application and Theory of Petri Nets*, pages 192 – 209, Bonn, West Germany, June 1989.

[81] G.C. Illing. Automatic Petri-net based Protocol Implementation. In *IREECON International*, pages 358 – 361, Melbourne, September 1989.

[82] ISO. Information Processing Systems - Open Systems Interconnection - ESTELLE - A Formal Description Technique based on Extended State Transition Model, IS 9074. ISO, 1989.

[83] ISO. Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique based on the Temporal Ordering of Observational Behaviour, IS 8807. ISO, 1988. The final text of DIS 8807 is published as ISO/IEC JTC 1/SC21 N2987, August 1988.

[84] W.D. Itzfeldt. The ESPRIT Project 125 GRASPIN. 1989. Handout at Tools Exhibition at the 10th International Conference on Application and Theory of Petri Nets, Bonn, West Germany 28-30 June 1989.

[85] L. A. Jackson. Special issue on systems design engineering. *British Telecom Technology Journal*, 4(3), July 1986.

[86] C. Jard, J-F. Monin, and R. Groz. Experience in implementing X.250 in VEDA. In M. Diaz, editor, *Protocol Specification, Testing and Verification, V*, pages 315–331, Elsevier Science Publishers B.V., 1986.

[87] K. Jensen. Coloured Petri Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties. Advances in Petri Nets 1986, Part 1: Proceedings of an Advanced Course, Bad Honnef, September 1986*, pages 248 – 299, Springer-Verlag, Berlin, February 1987. Lecture Notes in Computer Science, Vol. 254.

[88] K. Jensen. Coloured Petri Nets and the invariant-method. *Theoretical Computer Science*, 14:317–336, 1981.

[89] K. Jensen. Computer tools for construction, modification and analysis of Petri nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency. Advances in Petri Nets 1986, Part 2: Proceedings of an Advanced Course, Bad Honnef, September 1986*, pages 4 – 19, Springer-Verlag, Berlin, February 1987. Lecture Notes in Computer Science, Vol. 255.

[90] K. Jensen. Private communication. July 1988.

[91] W. Jürgensen and S.T. Vuong. Formal specification and validation of ISO Transport Protocol components, using Petri nets. In *ACM SIGCOMM '84 Symposium: Communications Architectures and Protocols*, pages 75–82, Montréal, Canada, June 1984.

[92] B. Krämer. SEGRAS - A Formal and Semigraphical Language combining Petri Nets and Abstract Data Types for the Specification of Distributed Systems. In *Proceedings of the Ninth International Conference on Software Engineering*, pages 116 – 125, CS Press, Los Alamitos, California, March 1987.

[93] B. Krämer. Stepwise construction of non-sequential software systems using a net-based specification language. In G. Rozenberg, editor, *Advances in Petri Nets 1984*, pages 307 – 330, Springer-Verlag, Berlin, 1985. Lecture Notes in Computer Science, Vol. 188.

[94] R.Y.L. Lai. *Formal Specification and Verification of ISO FTAM Protocol.* PhD thesis, Mathematical and Information Sciences, LaTrobe University, Melbourne, Australia, 1989.

[95] S. S. Lam. Protocol conversion – correctness problems. In *ACM SIGCOMM'86 Symposium: Communications Architectures and Protocols*, pages 19–29, Stowe, Vermont, USA, August 1986.

[96] K. Lautenbach and H. Schmid. Use of Petri nets for proving correctness of concurrent process systems. In *Proceedings of the IFIP Congress, Amsterdam, Netherlands*, pages 187–191, North Holland, Amsterdam, 1974.

[97] C. Lin and D.C. Marinescu. Stochastic High-Level Petri Nets and Applications. *IEEE Transactions on Computers*, 37(7):815 – 825, July 1988.

[98] J. Loeckx. Algorithmic specifications: a constructive specification method for abstract data types. *ACM Transactions on Programming Languages and Systems*, 9(4):646 – 685, October 1987.

[99] M.A. Marsan, G. Balbo, and G. Conti. A class of generalised Stochastic Petri Nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(2):93–122, May 1984.

[100] M.A. Marsan, G. Chiola, and A. Fumagalli. An accurate performance model of CSMA/CD bus LAN. In G. Rozenberg, editor, *Advances in Petri Nets 1987, Lecture Notes in Computer Science Vol. 266*, Springer-Verlag, April 1987.

[101] M.A. Marsan, G. Chiola, and A. Fumagalli. Improving the efficiency of the Analysis of DSPN models. In *Proceedings of the ninth European Workshop on Application and Theory of Petri Nets*, pages 183 – 201, Venice, Italy, 22-24 June 1988.

[102] M.A. Marsan and V. Signore. Timed Petri Net Performance Models of Fiber Optic LAN Architectures. In *Proceedings of the second International Workshop on Petri nets and Performance Models*, pages 66 – 74, IEEE Computer Society Press, Madison, Wisconsin, 24-26 August 1987.

[103] M.C.Browne. An improved algorithm for the automatic verification of finite state systems using temporal logic. In *Proceedings of the 1986 Conference on Logic in Computer Science*, pages 260 – 267, Cambridge, Massachusetts, June 1986.

[104] G. Memmi. Analysing nets by the invariant method. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties*, pages 300 – 336, Springer-Verlag, Berlin, 1987. Lecture Notes in Computer Science, Vol. 254.

[105] P.M. Merlin. *A study of the recoverability of computing systems.* PhD thesis, Information and Computer Science Department, UC Irvine, 1974. Tech Report, No. 58.

[106] P.M. Merlin and D.J. Farber. Recoverability of communication protocols - implication of a theoretical study. *IEEE Transactions on Communications*, 24(9):1036 – 1043, 1976.

[107] J. Meseguer and U. Montanari. Petri nets are monoids: a new algebraic foundation for net theory. In *Proceedings of the Third Annual Symposium on Logic in Computer Science*, pages 155 – 164, IEEE Comp. Soc. Press, Washington, DC, July 1988.

[108] M.K. Molloy. *On the Integration of Delay and Throughput Measures in Distributed Processing Models*. PhD thesis, UCLA, 1981.

[109] E.T. Morgan and R.R. Razouk. Interactive state-space analysis of concurrent systems. *IEEE Transactions on Software Engineering*, SE-13(10):1080 – 1091, October 1987.

[110] R. Nelson, L. Haibt, and P. Sheridan. Casting Petri nets into programs. *IEEE Transactions on Software Engineering*, SE-9(5):590–602, September 1983.

[111] H. Oberquelle. Nets as a tool in teaching and terminology work. In W. Brauer, editor, *Net Theory and Applications*, pages 481–506, Springer-Verlag, Berlin, 1980. LNCS Vol. 84.

[112] K. Okumura. A formal protocol conversion method. In *ACM SIG-COMM'86 Symposium: Communications Architectures and Protocols*, pages 30–37, Stowe, Vermont, USA, August 1986.

[113] C. J. O'Neill and J. Billington. Proposed abstract specification of the network service definition. *SDL Newsletter*, (6):139–158, January 1984.

[114] C. Paule and H. Eckert. *The NEt Simulation SYstem NESSY, Summary and Example*. Arbeitspapiere 182, GMD, November 1985.

[115] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall, Englewood Cliffs, N.J., 1981.

[116] T. F. Piatkowski. Protocol engineering. In *ICC*, pages 1328–1332, June 1983.

[117] C. Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*. PhD thesis, MIT, 1974. Project MAC report number MAC-TR-120.

[118] R.R. Razouk. *A Guided Tour of P-NUT*. Technical Report 86-25, Information and Computer Science Department, University of California, Irvine, January 1987. Release 2.2.

[119] R.R. Razouk and C.V. Phelps. Performance analysis using Timed Petri Nets. In Y. Yemini, R. Strom, and S. Yemini, editors, *Protocol Specification, Testing and Verification, IV*, pages 561 – 576, Elsevier Science Publishers B.V., Amsterdam, 1985.

[120] W. Reisig. *Petri Nets, An Introduction.* Volume 4 of *EATCS Monographs on Theoretical Computer Science*, Springer-Verlag, Berlin, 1985.

[121] W. Reisig. *Petri Nets and Algebraic Specifications.* SFB-Bericht 342/1/90 B, Technische Universität München, Institut für Informatik, München, West Germany, March 1990.

[122] W. Reisig. Petri nets in software engineering. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, pages 63 – 96, Springer-Verlag, Berlin, 1987. Lecture Notes in Computer Science, Vol. 255.

[123] W. Reisig and J. Vautherin. An algebraic approach to high level Petri nets. In *Proceedings of the Eighth European Workshop on Application and Theory of Petri Nets*, pages 51–72, Zaragoza, Spain, 24-26 June 1987.

[124] G. Richter. Clocks and their use for time modelling. In A. Sernandas et. al., editor, *Information Systems:Theoretical and Formal Aspects*, North Holland, Amsterdam, 1985.

[125] G. Rozenberg, editor. *Advances in Petri Nets 1987.* Volume 266 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1987.

[126] G. Rozenberg, editor. *Advances in Petri Nets 1988.* Volume 340 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1988.

[127] P.H. Starke. CPNA: Coloured Petri Net Analyser. 1989. Handout at Tools Exhibition at the 10th International Conference on Application and Theory of Petri Nets, Bonn, 28-30 June 1989.

[128] C.A. Sunshine, editor. *IEEE Trans. Com., Special Section on Protocol Specification, Testing, and Verification*, pages 2485 – 2518. Volume COM-30, IEEE, December 1982.

[129] F.J.W. Symons. Development and Application of Petri Net based techniques in Australia. In K. Voss, H.J. Genrich, and G. Rozenberg, editors, *Concurrency and Nets*, pages 497 – 509, Springer-Verlag, Berlin Heidelberg, 1987.

[130] F.J.W. Symons. Introduction to Numerical Petri Nets, a general graphical model of concurrent processing systems. *Australian Telecommunication Research*, 14(1):28 – 33, 1980.

[131] F.J.W. Symons. *Modelling and Analysis of Communication Protocols using Numerical Petri Nets.* PhD thesis, University of Essex, 1978. Dept. of Elec. Eng. Sci. Telecommunication Systems Group Report No. 152, May 1978.

[132] D. Taubner. On the implementation of Petri nets. In G. Rozenberg, editor, *Advances in Petri Nets 1988*, pages 418 – 439, Springer-Verlag, Berlin, July 1988. Lecture Notes in Computer Science, Vol. 340.

[133] D. L. Tennenhouse. The Unison Data Link protocol specification. September 1986. Draft Unison Project Document, Ref: UC022.

[134] H. Tuominen. Temporal logic as a query language for Petri net reachability graphs. In *Seventh European Workshop on Application and Theory of Petri Nets*, pages 327 – 342, Oxford, England, 30 June - 2 July 1986.

[135] K.J. Turner, editor. *Formal Description Techniques*, North Holland, Amsterdam, 1989.

[136] R. Valk. Self-modifying nets, a natural extension of Petri nets. In *Automata, Languages and Programming, Udine (Lecture Notes in Computer Science 62)*, pages 464–476, Springer-Verlag, 1978.

[137] A. Valmari. Compositional state space generation. In *Proceedings of the Eleventh International Conference on the Application and Theory of Petri Nets*, Paris, France, 27-29 June 1990.

[138] A. Valmari. Eliminating redundant interleavings during concurrent program verification. In *Proceedings of the European Conference on Parallel Architectures and Languages Europe*, Eindhoven, The Netherlands, 12-16 June 1989.

[139] A. Valmari. Error detection by reduced reachability graph generation. In *Proceedings of the Ninth European Workshop on the Application and Theory of Petri Nets*, pages 95 – 112, Venice, Italy, 22-24 June 1988.

[140] A. Valmari. State space generation with induction. In *Proceedings of the second Scandinavian Conference on Artificial Intelligence (SCAI'89)*, Tampere, Finland, 13-15 June 1989.

[141] A. Valmari. Stubborn sets for reduced state space generation. In *Proceedings of the Tenth International Conference on the Application and Theory of Petri Nets*, Bonn, West Germany, 28-30 June 1989. First paper of Supplement to the Proceedings.

[142] J. Vautherin. Calculation of semi-flows for Pr/T-systems. In M.K. Molloy and M.K. Vernon, editors, *Proceedings of the International Workshop on Petri Nets and Performance Models*, pages 174–183, IEEE Computer Society Press, Washington D.C., August 1987.

[143] J. Vautherin. Parallel systems specifications with Coloured Petri Nets and algebraic specifications. In G. Rozenberg, editor, *Advances in Petri Nets 1987*, pages 293 – 308, Springer-Verlag, Berlin, April 1987. Lecture Notes in Computer Science, Vol. 266.

[144] K. Voss, H.J. Genrich, and G. Rozenberg, editors. *Concurrency and Nets. Advances in Petri Nets*, Springer-Verlag, Berlin Heidelberg, 1987.

[145] G. R. Wheeler. *Numerical Petri Nets - A Definition*. Research Laboratories Report 7780, Telecom Australia, May 1985.

[146] G. R. Wheeler, T. J. Batten, J. Billington, and M. C. Wilbur-Ham. A methodology for protocol engineering. In *ICCC86*, pages 525 – 530, Munich, September 1986.

[147] G. R. Wheeler, M. C. Wilbur-Ham, J. Billington, and J. A. Gilmour. Protocol analysis using Numerical Petri Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1985*, pages 435–452, Springer-Verlag, Berlin, 1986. Lecture Notes in Computer Science, Vol. 222.

[148] M. C. Wilbur-Ham. April 1987. Private Communication.

[149] M. C. Wilbur-Ham and J. Billington. A Protocol Emulation and Analysis Tool. In *IREECON International*, pages 30–32, Sydney, September 1983.

[150] G. Winskel. Petri Nets, Algebras, Morphisms, and Compositionality. *Information and Computation*, 72(3):197 – 238, March 1987. Earlier version published in University of Cambridge Computer Laboratory Technical Report 79, September 1985.

[151] H. Zimmermann. OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. *IEEE Trans. Comm.*, COM-28(4):425 – 432, April 1980.

Appendix A

Sets, Multisets and Vectors

A.1   Sets

We make use of the following sets:

- $N = \{0, 1, \ldots\}$ the natural numbers.

- $N_\infty = N \cup \{\infty\}$

- $N^+ = N \setminus \{0\}$, the positive integers

- $N_\infty^+ = N^+ \cup \{\infty\}$

- $Z = \{\ldots, -1, 0, 1, \ldots\}$, the integers

A.2   Multisets

We define a multiset, B, (also known as a bag) over a *basis* set, A, to be the function

$$B : A \longrightarrow N$$

which associates a multiplicity, possibly zero, with each of the basis elements. (We require multisets to have finite support.) There are times when we shall consider a set as a special case of a multiset, where the multiplicities of each of the basis elements is unity.

The set of multisets over A is denoted by μA (i.e. $\mu A = [A \longrightarrow N]$). For a multiset $B \in \mu A$, to avoid confusion, we sometimes use the notation $mult(a, B) = B(a)$ where $a \in A$, for the multiplicity of a in B.

We may extend the definition to include the value ∞, and denote the set of multisets over A, that allows infinite multiplicities, by $\mu_\infty A = [A \longrightarrow N_\infty]$ and that which disallows multiplicities of zero by $\mu_\infty^+ A = [A \longrightarrow N_\infty^+]$.

A.2.1 Vector or Sum representation

We may represent a multiset as a symbolic sum of basis elements scaled by their multiplicity.

$$B = \sum_{a \in A} B(a)a$$

A.2.2 Membership

Given a multiset, $B \in \mu_\infty A$, we say that $a \in A$ is a member of B, denoted $a \in B$, if $B(a) > 0$, and conversely if $B(a) = 0$, then $a \notin B$.

The empty multiset, $\emptyset$, has no members: $\forall a \in A, \emptyset(a) = 0$.

A.2.3 Cardinality

We define *multiset cardinality* in the following way. The cardinality $|B|$ of a multiset B, is the sum of the multiplicities of each of the members of the multiset.

$$|B| = \sum_{a \in A} B(a)$$

A.2.4 Equality and Comparison

Two multisets, $B1, B2 \in \mu A$, are equal, $B1 = B2$, *iff* $\forall a \in A$, $B1(a) = B2(a)$, and $B1$ is less than or equal to (or contained in) $B2$, $B1 \leq B2$ *iff* $\forall a \in A$, $B1(a) \leq B2(a)$. Other comparison operators may be defined analogously.

A.2.5 Operations

We define four binary operations on multisets, $B1, B2 \in \mu A$, known as union, intersection, addition and subtraction, as follows:

$B = B1 \cup B2$ *iff* $\forall a \in A$ $B(a) = max(B1(a), B2(a))$
$B = B1 \cap B2$ *iff* $\forall a \in A$ $B(a) = min(B1(a), B2(a))$
$B = B1 + B2$ *iff* $\forall a \in A$ $B(a) = B1(a) + B2(a)$
$B = B1 - B2$ *iff* $\forall a \in A$ $(B1(a) \geq B2(a)) \wedge (B(a) = B1(a) - B2(a))$

We also define scalar multiplication of a multiset, $B1 \in \mu A$, by a natural number, $n \in N$, to be

$$B = nB1 \text{ } iff \text{ } \forall a \in A, B(a) = n \times B1(a)$$

A.2.6 Adding ∞ and Subtracting from ∞

For all $n \in N$, $n + \infty = \infty + n = \infty$.
For all $n \in N$, $\infty - n = \infty$.

A.2.7 Multiplication by ∞

For all $n \in N_{\infty}^{+}$, $\infty \times n = n \times \infty = \infty$ but $\infty \times 0 = 0 \times \infty = 0$.

A.3 Vectors

There are times when we wish to subtract one multiset from another when the above restriction on multiset subtraction does not apply. We then need to consider multisets as vectors. We define a vector, V, over a (basis) set, A, to be the function

$$V : A \longrightarrow Z$$

which associates a negative, zero or positive multiplicity, with each of the basis elements. The set of vectors over A is denoted by νA (i.e. $\nu A = [A \longrightarrow Z]$). For a vector, $V \in \nu A$, to avoid confusion, we sometimes use the notation $mult(a, V) = V(a)$ where $a \in A$, for the multiplicity of a in V.

Subtraction is a closed operation for vectors defined component-wise as follows. For $V1, V2 \in \nu A$

$$V = V1 - V2 \; \textit{iff} \; \forall a \in A, \; V(a) = V1(a) - V2(a)$$

.

We can also define scalar multiplication of a vector, $V1 \in \nu A$, by an integer, $z \in Z$, to be

$$V = zV1 \; \textit{iff} \; \forall a \in A, V(a) = z \times V1(a)$$

A.3.1 Equality and Comparison

Two vectors, $V1, V2 \in \nu A$, are equal, $V1 = V2$, *iff* $\forall a \in A$, $V1(a) = V2(a)$, and $V1$ is less than or equal to $V2$, $V1 \leq V2$, *iff* $\forall a \in A$, $V1(a) \leq V2(a)$. Similar definitions apply for $\geq, <$ and $>$.