CHU SPACES: A MODEL OF CONCURRENCY

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Vineet Gupta
August 1994

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Vaughan R. Pratt
(Principal Advisor)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

John C. Mitchell

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

R. J. van Glabbeek

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies

Acknowledgements

A significant portion of this thesis represents joint work with my advisor Vaughan Pratt, however, his ideas and philosophy pervade each and every section of it. He has been instrumental in shaping my ideas of concurrent systems and my view of research in general, and his support and intellectual guidance have made this dissertation possible.

I would also like to thank the members of my reading committee, Dr. R. J. van Glabbeek, Prof. John Mitchell and Prof. Michael Genesereth for taking the time to carefully read the initial drafts and providing extensive comments. Many stimulating discussions with Carolyn Brown, Ross Casley, Rob van Glabbeek, Radha Jagadeesan, Gordon Plotkin, Vijay Saraswat and Ramesh Viswanathan have in numerous ways contributed to the ideas of this thesis — I am indebted to each one of them.

I gratefully acknowledge the generous financial support provided by Accel Partners in the form of a Center for Telecommunications fellowship, and would like to thank Dr. Dixon Doll and Prof. Dale Harris in this context. Substantial support was also provided by several NSF grants and a gift from Mitsubishi Corporation.

In a very significant way, this thesis is a product of the confidence that my wife Mudita has always expressed in me — a special "thank you" to her. Another special thank you goes to my sister Seema, for all her encouragement. And finally, I cannot even begin to express my gratitude to my parents, this thesis is affectionately dedicated to them.

Contents

List of Figures

Abstract

A Chu space is a binary relation between two sets. In this thesis we show that Chu spaces form a non-interleaving model of concurrency which extends event structures while endowing them with an algebraic structure whose natural logic is linear logic.

We provide several equivalent definitions of Chu spaces, including two pictorial representations. Chu spaces represent processes as automata or schedules, and Chu duality gives a simple way of converting between schedules and automata. We show that Chu spaces can represent various concurrency concepts like conflict, temporal precedence and internal and external choice, and they distinguish between causing and enabling events.

We present a process algebra for Chu spaces including the standard combinators like parallel composition, sequential composition, choice, interaction, restriction, and show that the various operational identities between these hold for Chu spaces. The solution of recursive domain equations is possible for most of these operations, giving us an expressive specification and programming language. We define a history preserving equivalence between Chu spaces, and show that it preserves the causal structure of a process.

Chapter 1

Introduction

This dissertation presents a model of concurrent behavior called Chu spaces. We propose this model as a simple yet effective means of understanding the mathematics behind concurrency.

The goal of any model of concurrency is to formalize the notion of concurrent behavior. Whenever a programmer writes a program, he or she has some intuitive idea of how the system will behave. For example, if one writes `if` (a `or` b) `then` `A`, then one has some notion of how the system would evaluate a `or` b. This may or may not correspond to how the system actually does the evaluation. The purpose of a mathematical model is to make precise the intuitions, so that there is no gap between the user's perspective and the actual implementation. This gap is even more apparent in concurrency, making mathematical models absolutely necessary.

The second goal is to make available the tools that mathematicians have developed over the years to designers and users of concurrent systems. This is clear by analogy with other areas of computer science, for example computer graphics, which has clearly benefitted from the mathematical models of geometry — vector spaces, metric spaces, topology etc. These have not only led to a formalization of graphics, but have also led to a variety of mathematical tools for graphics. We seek a similar model for concurrent behavior, which would help develop tools useful for designing and verifying concurrent systems — traditional tools for debugging and simulation are tailored to deterministic sequential programs and are very inadequate for concurrent programs.

Many models of concurrent behavior already exist — Petri nets, automata, CCS, event structures etc. So why do we need a new model? Our experience with these models has shown us that none of them are as well matched to concurrent behavior as say vector spaces are to graphics. We will examine a number of these models in detail in the next chapter to show this. In this dissertation we propose a model which seems almost trivial due to its mathematical simplicity, yet is rich enough to subsume many of the models described above.

Central to the notion of all computer systems is the idea of a *state*, which is a snapshot of a system at any time. The other basic idea is that systems can move from one state to another by doing certain actions. We take just these two ideas, and without imposing upon them any additional mathematical structure, use them to get a model of concurrency. So a system is modeled as a pair of sets — the set of states it can be in, and the set of events (occurrences of transitions) that can happen during its execution. The state of a system carries the history of the system, namely the set of events that have occurred so far. This is exactly the definition of a Chu space—a set of events, and a set of states each of which contains the history by being represented as a set of events.

In the next chapter we will examine some models of concurrency. We will see the ideas which they contributed to Chu spaces, and also see how they fall short of our intuitions for concurrent behavior. In chapters 3 and 4 we will define Chu spaces and present some of the mathematics behind them. We will also give a summary of the applications of Chu spaces to mathematics, the details of these are given elsewhere.

In Chapter 5 we will show how Chu spaces describe the behavior of a process. We will also show how various features of concurrent behavior can be represented in a Chu space. Chapter 6 will describe the algebra of Chu spaces, giving the various operators on them, and interpreting them as constructs for concurrency. We prove a number of identities for Chu spaces, showing that these operators match our intuitions for these operators.

Chapter 7 will compare Chu spaces with the models described in chapter 2. We will show that event structures and Petri nets are generalized by Chu spaces, and will give a non-interleaving Chu space semantics for CCS and CSP. Finally we will present some directions for future work in this area.

This work originated from Professor Vaughan Pratt's study of non-interleaving models of concurrency in the last decade. In 1991 he refined his model of partially ordered multisets to get a new model, event spaces [Pra92c, Pra92a], which are partial orders with all nonempty joins and the empty meet. This further prompted him to define a more general model, partial distributive lattices, which are partial orders for which meet and join are partial functions, that is they may not be defined for all subsets of the set of elements. These partial functions, $\wedge, \vee : 2^A \to A$ had to obey certain axioms which forced them to behave like meet and join for lattices, namely they had to satisfy the idempotence, associativity, absorption and distributivity laws.

In our work on partial distributive lattices, we discovered that they did not satisfy the involution of duality ($A^{\perp\perp} \cong A$), and lacked some other properties. We found that we had to keep more information than just equations of the form $\wedge X = a$ and $\vee Y = b$. We discovered that the most succinct method of keeping this information was by embedding the set A in an underlying distributive lattice, giving the new definition of partial distributive lattices described in chapter 3. Shortly thereafter, Pratt made the connection with Chu's construction, giving the name Chu spaces to partial distributive lattices.

In this thesis, the work in chapters 3 – 5 was mostly done jointly with Prof. Pratt, my advisor. In particular, the logical representation and the gate representation of Chu spaces was his idea, and he was also responsible for the ideas behind figure 3.5. Parts of chapters 6 and 7 were also done jointly, especially the definitions of the combinators for Chu spaces and the embedding into event structures. Prof. Pratt wrote the macros for drawing lattices and circuits, which I have used extensively, as well as the macros for writing the symbol ⅋ (other authors have used all sorts of weird combinations for writing it in LaTeX!). The work in chapter 6 has also benefited from discussions with Dr. R. J. van Glabbeek and Prof. Gordon D. Plotkin. Some of the results in the thesis have been published in [Gup93, GP93].

Chapter 2

Models of Concurrency

Before we get into Chu space theory and applications we will briefly review some models of concurrency which contributed ideas to our model. These include Petri nets, traces, transition graphs and CCS, event structures, geometric automata and pomsets.

All models of concurrency have a concept of a state, which is a description of a system at a particular moment. They also have a way of moving from one state to another by means of certain transitions. Various models represent these transitions by *actions* or *events*. Petri nets, transition graphs and traces are action based models because any transition can occur repeatedly. Event structures, geometric automata and pomsets are event based models, where each event can occur at most once during the execution of a process.

In order to represent multiple occurrences of an action, event based models use labeling functions, so each event can be labeled with an action. Event based models keep around the entire history of a process, so there are no loops, i.e. after moving out from a state, a system can never return to that state. The main disadvantage of this is that processes which can be represented by small structures in an action based model may need infinite structures in event based models. On the other hand representing each event explicitly allows easy formulation of these models as categories, which automatically equips them with a tractable algebraic theory. Keeping track of each occurrence of an action makes the information content of each state very clear, making them close to domain theory and providing them with a schedule-automaton duality.

Another way to classify these models is as interleaving versus non-interleaving models. Interleaving models have a set of actions which are *atomic* or indivisible, and the parallel execution of two atomic actions is regarded as equivalent to executing them in any order. Thus they satisfy the law : $a||b = ab \sqcup ba$[1]. CCS transition graphs, traces and geometric automata are examples of interleaving models. Non-interleaving models do not project events onto a linear timescale. In such a model, $a||b$ means that there is no information about the order relation between a and b. This is regarded as different from $ab \sqcup ba$, which represents mutual exclusion between a and b. Petri nets, event structures and pomsets are non-interleaving models.

The main justification for interleaving models is that for some kinds of observations, the two processes $a||b$ and $ab \sqcup ba$ are observationally the same. Besides, this assumption leads to a more tractable algebraic theory. On the other hand, the interleaving assumption forces some actions to be regarded as atomic, imposing a fixed granularity on the actions. This may be impractical, as one may wish to view actions on different scales depending on the context. For example, if a user wants to reason about a piece of software, the lowest granularity available may be a lot bigger than atomic actions, as the software writer may not give details below a certain level for proprietary reasons or for modularity — future releases could implement the same actions in a different way. Thus non-interleaving models allow *action refinement*, the splitting up of an action into several smaller actions.

Another philosophical problem with the interleaving assumption is that it imposes on the process a global clock by which all actions are timed, but in distributed systems there is no such clock as delays caused by the network and relativity may allow different observers to see different temporal orders for the actions. Thus it does not make sense to say that they occur in some interleaved order.

[1] $a \sqcup b$ is a choice between a and b

2.1 Action based models

2.1.1 Petri nets

Petri nets are one of the oldest models for modeling concurrent behavior [Pet62b, Pet62a, Rei85]. A net is a directed bipartite graph — (S, T, F), where S and T are the vertices, $S \cap T = \emptyset$ and the set of edges is $F \subseteq (S \times T) \cup (T \times S)$. While there are various flavors of Petri nets, all of them have some notion of *marking*, where tokens are placed on various S vertices, and a transition $t \in T$ fires by taking some tokens from the sources of its incoming arcs and placing some tokens on the targets of its outgoing arcs. The state of a Petri net is given by the marking at any point in its execution.

One of the most appealing features of Petri nets is that they have a simple pictorial representation, as a graph with two kinds of nodes, S-nodes are drawn as circles and T-nodes as squares. Now the tokens can be represented by a number on the S-nodes, and T-nodes can have labels for transitions. This representation makes it easy to understand the structure of a net, and we will continue the tradition by giving a pictorial representation of Chu spaces.

The two common kinds of Petri nets are condition-event nets (C/E nets) and place transition nets (P/T nets). In C/E nets, the vertices in S are conditions and a condition is *true* iff there is a token on it (there can be at most one token per condition). The nodes in S with arcs leading to a transition t are its preconditions, and those which have an arc from t into them are the postconditions. t is *enabled* iff all its preconditions are true and all its postconditions are false. Its firing results in all its preconditions becoming false and all the postconditions becoming true.

A special case of C/E nets is an occurrence net — an acyclic C/E net where each S-node has at most one incoming arc and at most one outgoing arc. Notice that while in a C/E system a transition may be fired many times, in an occurrence net it may be fired exactly once, as is characteristic of event based formalisms. An occurrence net may be formed by unrolling a condition event system, and the acyclic nature allows composition of occurrence nets more easily than for C/E systems.

In an P/T net, each place $s \in S$ has a capacity $K(s)$, and each arc $f \in F$ has a weight $W(f)$. A marking is a function $M : S \to \mathbb{N} \cup \{\omega\}$ giving the number of tokens at each place, and $\forall s. M(s) \leq K(s)$. A transition t is enabled if the source s of each incoming arc f has $M(s) \geq W(f)$, and for each outgoing arc f from t to $s \in S$, $W(f) + M(s) \leq K(s)$. A transition fires by removing the $W(f)$ tokens from the source of each incoming arc f, and place $W(g)$ more tokens on the target of each outgoing arc g, this results in a new marking.

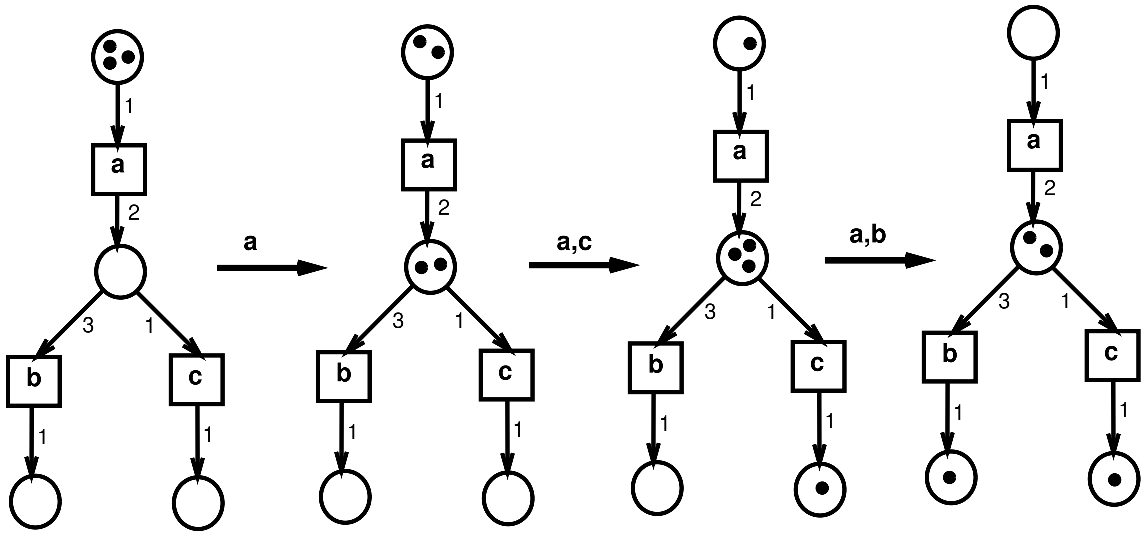


Figure 2.1: A Petri net, and a few steps of its execution.

Note that in Petri nets where two or more transitions are enabled, they may both fire simultaneously, so this is a non-interleaving model. A Petri net with its markings is a simple convenient way to represent which transitions can fire from which state. In Chu spaces we give this relation explicitly, thus allowing Chu spaces to mimic any Petri net.

The main problem with Petri nets is that they do not possess a nice algebra of operations — there are no good methods of combining Petri nets using the standard concurrency operators to get larger Petri nets — they lack *modularity*. Several approaches have been explored to get an algebra. Safe Petri nets are P/T nets where the number of tokens at any place never exceeds one in any reachable marking, even though the capacity of each place may be infinite. Algebras of unlabeled safe Petri nets have been proposed in [LR75] and [Bes86]. In [Bes86], a Petri net semantics has been defined for the language COSY, which has choice, sequential and parallel composition and iteration. However the constructions are specific to nets generated in this way. General definitions are given in [GV87], but the constructions are somewhat complicated. Also none of the authors give a method for doing action refinement, which is one important reason for preferring a non-interleaving semantics over an interleaving one.

Recently, in a series of papers Brown and Gurr [BG90, BGdP91] have come up with a categorical formulation of Petri nets, which provides an algebra for Petri nets. The category is based on Chu's construction, and its objects are Petri nets. The morphisms of their category are simulations, and the categorical product turns out to be synchronous parallel composition. However they do not show how to implement the other operations like sequential composition or choice. Also, they do not make much use of duality. Other categorical formulations of Petri nets have been developed by Winskel [Win88] and Meseguer and Montanari [MM90].

2.1.2 Calculus of Communicating Systems and transition graphs

.

CCS was developed as a result of Robin Milner's efforts towards developing a theory of communicating concurrent processes [Mil80, Mil89]. In this theory processes are defined by algebraic equations, and evolve into other processes by doing some actions. The theory is described in detail in the above references, and we will talk about it in later chapters; here we discuss the prominent features of CCS.

Transition graphs. A model of a CCS process is a labeled directed graph. Its nodes or states are labeled by process terms, and a transition is made from one node to another along an edge by executing the action labeling the edge.

Observations and bisimulation. To an outside observer, the only way of obtaining information is by observing the sequence of actions executed by a process. Since only a sequence of actions can be observed, this is an interleaving model. Two processes are equivalent if they have the same observations, formally captured by the notion of *bisimulation* [Par81]:

Let $P \xrightarrow{\alpha} P'$ represent the evolution of process P to the process P' by doing the action α. Then two processes P and Q are said to be strongly bisimilar if there is a binary relation $\mathcal{R}$ on processes such that for $(P, Q) \in \mathcal{R}$, and for every action α,

1. $P \xrightarrow{\alpha} P' \Longrightarrow \exists Q'.[Q \xrightarrow{\alpha} Q' \text{ and } (P', Q') \in \mathcal{R}]$.
2. $Q \xrightarrow{\alpha} Q' \Longrightarrow \exists P'.[P \xrightarrow{\alpha} P' \text{ and } (P', Q') \in \mathcal{R}]$.

This notion gives a finite method to test whether any arbitrarily long observations of one process can also be made of another. It can also be used on transition graphs; then the relation is between states. We will define a bisimulation between Chu spaces in chapter 6.

Early and late branching. The bisimulation notion above makes an important distinction between processes — the processes $a(b \sqcup c)$ and $ab \sqcup ac$ are distinguished, because the first can make a transition on a and evolve to $b \sqcup c$, while the second one, after the a can evolve to b or to c, neither of which is bisimilar to $b \sqcup c$.

The distinction is made because the first process made the choice of which branch to take after doing the a, while the second one made the choice at the beginning. Now in some cases it is possible for the second process to deadlock in a situation when the first one does not, because it can make a more informed decision. While automata respect this distinction (the automata for the two processes would be different), languages (sets of strings) do not, and the languages for the two processes are the same, $\{ab, ac\}$. The model we present for concurrent processes will make this distinction, as it has important consequences for deadlock detection.

Communication. Two processes which are executing in parallel can communicate with each other by means of certain special actions. These actions are called silent actions because they are transparent to the environment. The notion of strong bisimulation above is extended to ignore these actions, and the resulting definition is called weak bisimulation.

Communicating Sequential Processes [Hoa85] is another language whose models are similar to those of CCS. We will talk about both of these in chapter 7 in more detail.

2.1.3 Traces

Traces or languages have been a popular way of representing behavior of processes. The behavior of a process is characterized entirely by the set of its observations or traces. This approach has been very successful in studying sequential behavior, where the behavior of an automaton (e.g. a Turing machine) is identified with the set of strings it accepts. It has been extended to concurrent processes by regarding the parallel execution of two processes as the shuffle of their languages. Traces can be combined with one another using the various operations on strings, giving a nice algebraic theory of processes.

Traces are a very simple idea, and have received a lot of attention, but they do ignore a lot of the structure of a process. They do not distinguish between early and late branching, and form an interleaving model.

Since traces identify a lot of processes, several variants of traces, called decorated traces, have been proposed to distinguish processes. In complete trace semantics, a process is described by a pair of sets—a set of traces and a set of completed traces. This helps in distinguishing between $ab + a$ and ab, since a is a complete trace of the first but not of the second. Another approach followed by Brookes, Hoare and Roscoe [BHR84] is to include failure pairs $\langle \sigma, X \rangle$, where σ is a trace, and X is a set of actions, and the pair $\langle \sigma, X \rangle$ means that the process executed σ, and after that when given a choice only from among the actions in X, could not do anything more. This allows a distinction between $a(b \sqcup c)$ and $ab \sqcup ac$, which have the same traces and complete traces, but the second process has the failure pair $\langle a, \{b\} \rangle$ where the first does not. There are various other decorated trace models, a detailed account is given in Rob van Glabbeek's PhD thesis [vG90].

Our model Chu spaces will make all the distinctions that can be made by the different trace semantics, and actually makes a few more. However it is possible to define equivalence relations on Chu spaces, such that only processes distinguished by some trace semantics are distinguished—we will show some of these in chapter 7.

2.1.4 Higher dimensional automata

This model was proposed by Pratt in [Pra91] as a generalization of automata to allow them to express non-interleaving concurrency. Standard finite automata are drawn as points representing states and directed arcs representing transitions. So all elements are 0 and 1-dimensional objects. Pratt generalized this to allow elements of any finite dimension, where an n-dimensional object stands for a transition representing the concurrent occurrence of n actions. Mathematical foundations of this model are developed in detail in [GJ92].

Computation may be viewed as a path through such an automaton. Concurrent execution of a and b, $a||b$, is represented as a square whose surface is filled in, and mutual exclusion $ab \sqcup ba$ as a square whose interior is hollow, so one has to follow the edges, doing one of a and b at a time. Note that after doing both a and b, the process "forgets" which it did first, and this is good as keeping such useless information is unnecessary.

Communication can be modeled abstractly as eroding some of the interior surface. When two processes synchronize, they must both be at some fixed stage in their execution *simultaneously*, that is the execution trajectory must pass through a point. Monotonicity of computation now means that certain parts of the square are illegal. Asynchronous communication is similarly modeled as eroding the area where the message was received before transmission. More communication erodes more area, in the extreme case leading to a single path of execution, when both processes are in lock-step. Pratt calls this "the jaws of communication"—they squeeze out the freedom of action of a process.

In fact it is possible to generalize the concept of a computation from a path to a set of paths. As a first approximation we can take it to be a homotopy class, i.e.

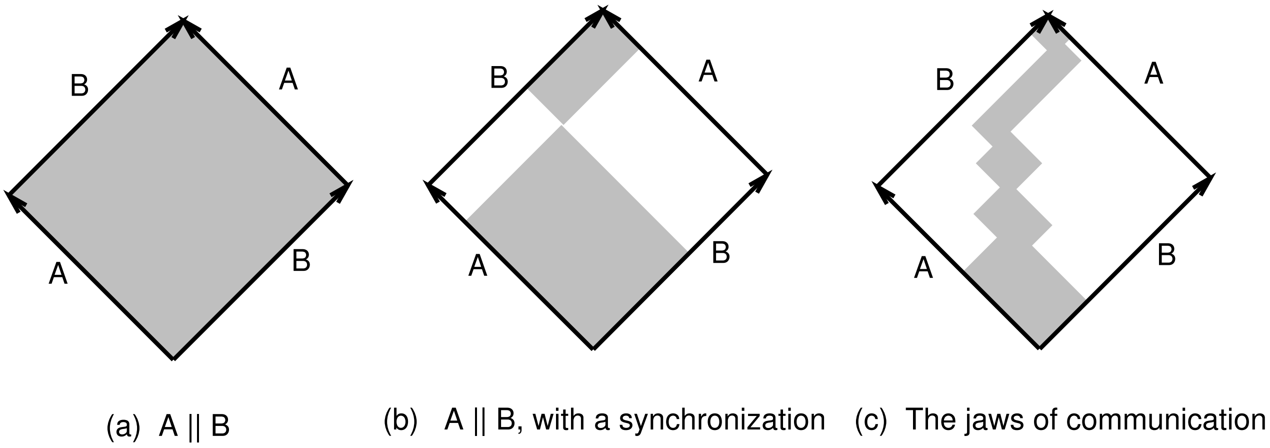


Figure 2.2: Higher Dimensional Automata. Shaded area represents a surface.

a set of paths where each path can be deformed into another without jumping over holes. This is a deterministic computation, since no choices around holes needed to be made. The ignorance of the exact path reflects the observer's ignorance of the exact position where each process is in a computation, since such accuracy may not be available or necessary. As a further generalization, we could agree to gloss over some holes, reflecting the fact that all choices are not relevant for future computation. This is a nondeterministic computation.

Higher dimensional automata are rather difficult to specify, the specifications are quite long. They are very biased towards the automaton side of computation, forgetting about schedules, and at the moment there seems to be no easy way of dualizing an automaton to get its schedule, the way we can do with Chu spaces. However they are able to control information in a better way, allowing forgetting of useless information, and we would like to extend Chu spaces in this direction.

2.2 Event based models

2.2.1 Partially ordered multisets

Partial orders are the simplest non-interleaving model of processes—instead of ordering events linearly as in a trace, they are ordered partially. Partially ordered multisets (pomsets) are posets with a labeling function which labels every event with an action. Thus pomsets generalize strings, which are labeled traces.

Pomsets as a model for concurrent processes have been studied in [Gra81, Pra82, Pra86, Gis88]. A process is modeled as a set of pomsets, and a run executes one of the pomsets in this set. Pomsets can be combined with each other with the operations described in [Pra86], while processes can be combined using the familiar operations on sets. Alternatively, first order logic or temporal logic formulas may be used to specify pomsets, and the algebraic and logical specifications may be freely mixed.

Pomsets have been generalized to prossets [GP87], metric process models[Cre91] and measured sets [Cas91]. Our model does not yet subsume these extensions, though it does subsume pomsets as originally defined. In addition, it gives a more uniform treatment of nondeterminism, so choices can be made at any time in the execution, rather than at the beginning as required by pomsets (the pomset to be executed is chosen at the start of a run). In addition, pomsets emphasize the schedule view of processes, whereas Chu spaces can treat processes as either schedules or automata.

2.2.2 Event Structures

Event structures were developed in [NPW81] as an attempt to bridge Petri net theory and domain theory. In a Petri net, the state is given by a marking, but the same marking can be reached after several different sets of transitions. Thus the information theoretic content of a marking is rather obscure. Event structures remedy this by making the state of a system be exactly the set of actions that have occurred so far.

However in a Petri net multiple occurrences of the same action can occur, so if a state just recorded which actions had occurred and some actions occurred many times, this information would be forgotten. But since the purpose of a state is to keep as much information as possible, the concept of *event* or occurrence of an action was introduced. An event is an action which occurs at most once in an execution.

So a prime event structure is formally defined as a triple $(E, \#, \leq)$, where E is the set of events, $\# \subseteq E \times E$ is a conflict relation, and $\leq \subseteq E \times E$ is a partial order. If two events are in conflict, both of them cannot happen in a single execution. Also, for the event e to happen, all the events before it in the partial order must have happened.

Note that there is no condition which forces two events to occur in some order if they are both enabled, so this is a non-interleaving model.

Prime event structures correspond exactly with occurrence nets described above [NPW81]. They have been generalized considerably—both the conflict relation and the partial order can be generalized, and we shall discuss these in chapter 7.

The state of a process modeled as an event structure is just a history, or the set of events that have occurred so far. The states can be arranged into a partial order based on set inclusion, and this structure is the configuration structure or domain corresponding to an event structure. A configuration structure can be independently characterized as a prime algebraic coherent partial order [NPW81], and have been further explored in [Dro89].

Chu spaces are very closely related to event structures, and can in fact be understood as generalized configuration structures, in which all the constraints placed by event structures are removed. Thus event structures will be a subclass of Chu spaces. The major difference is in the maps between event structures and the maps between Chu spaces. While event structure maps preserve consistency of events (the complement of conflict), Chu maps will preserve conflict.

The category of event structures and their morphisms enables the definition of an algebra of event structures, with the basic operations partial synchronous product and choice. Since configuration structures are domains, recursive domain equations can be solved for them, so they can be used to give a semantics for CCS and CSP[Win86].

A recent extension of event structures is event automata [PP92], which generalize configuration structures by relaxing some of the conditions. Thus any event structure is representable as an event automaton. However this generalization loses the duality between event structures and configuration structures — there is no good characterization for the schedule corresponding to an event automaton. This is fixed by Chu spaces, which subsume event automata, while providing a schedule view of a process.

2.2.3 Geometric Automata

Geometric automata were constructed by J. Gunawardena [Gun91, Gun92] as a generalization of event structures. They are based on a syntactic approach to causality, rather than the model-theoretic approaches followed by the two previous models. A geometric automaton consists of a set of events, each of which is associated with a condition, a boolean formula on the events[2]. Executing an event means changing its value from 0 to 1, and events are executed one at a time when their conditions are satisfied.

Geometric automata provide a very declarative style of programming—for each event we state when it can happen. This is something we carry over, with some generalization, to Chu spaces, where it will also be possible to write out a Chu space as a boolean formula. However the meaning of the formulas is different for Chu spaces and geometric automata — in a geometric automaton, the condition of an event must be true only at the time it is executed, it can become false later, while in a Chu space the condition is always true. Thus there are some geometric automata whose behavior cannot be modeled by Chu spaces in a nice fashion, and vice versa. We will talk about this in chapter 7.

Geometric automata are an interleaving model of concurrency, since one event is executed at a time. This is necessary because if the conditions of two events are satisfied simultaneously, doing one may invalidate the condition of the other.

These are only some of the many existing models of concurrent behavior. We will keep referring to them in the following chapters, to show their relation to Chu spaces.

[2]There are some syntactic conditions on infinitary formulas, arising from the non-existence of a free Boolean algebra on an infinite number of generators.

Chapter 3

Definition of a Chu space

In this chapter, we formally define Chu spaces. Whereas Chu spaces were originally defined as the result of Chu's construction on the category of sets and functions, we will give elementary definitions and return to the categorical definition at the end of the chapter.

3.1 Relations and Chu spaces

Definition 1 A Chu space is a binary relation between two sets A and X. We write it as a triple (A, X, R), where $R : A \times X \to 2$ gives the binary relation as a characteristic function of a subset of $A \times X$. 2 is the set $\{0, 1\}$.

We do not impose any cardinality restrictions on A and X. Thus all arguments given below will work for all cardinalities. We can think of A as the set of events and X as the set of states of the process represented by the Chu space. Then $R(a, x)$ tells us whether event a has occurred in state x.

The obvious way to write out a Chu space explicitly is as a binary matrix of dimension $|A| \times |X|$, with each entry giving the value of the function R on its pair of coordinates. We will actually write A on the top, and X on the side to correspond with some intuitions about time and information. The following are examples of some Chu spaces, referred to again in Figures 3.2 and 3.3. We will use $a, b, c, d, \ldots$ to denote

$$\begin{array}{cc} & abcd \\ x & 1101 \\ y & 1011 \end{array} \qquad \begin{array}{cc} & abcd \\ u & 0001 \\ v & 0011 \\ w & 0101 \\ x & 0111 \end{array} \qquad \begin{array}{cc} & abcd \\ u & 0001 \\ v & 0001 \\ w & 0011 \\ x & 0101 \\ y & 0111 \end{array} \qquad \begin{array}{cc} & abc \\ u & 001 \\ v & 100 \end{array} \qquad \begin{array}{cc} & abc \\ u & 100 \\ v & 011 \\ w & 110 \end{array}$$

$$(a) \qquad\qquad (b) \qquad\qquad (c) \qquad\qquad (d) \qquad\qquad (e)$$

Figure 3.1: Five Chu spaces.

elements of A, and $u, v, w, x, \ldots$ for elements of X. We also use $\mathcal{P}$, $\mathcal{Q}\ldots$ to represent Chu spaces.

While we have not yet assigned any formal meanings to the elements of X, viewing them in different ways yields some useful definitions. Firstly, it is possible to view each element of X as a function from A to 2. Then each row gives the values that the function takes on different elements of A. Now in the extensional view of functions, two functions on the same domain are considered equal if they have the same value everywhere in the domain. So we define an *extensional* Chu space to be one in which if any two functions in X are equal extensionally, then they are the same element. In other words, a Chu space is extensional iff it has no repeated rows. In figure 3.1, (a), (b), (d) and (e) are extensional, while (c) is not. Operationally this means that two states in which exactly the same events have occurred are considered indistinguishable.

Dually we can define Chu spaces in which no columns are repeated. Such Chu spaces are called T_0. This comes from the analogy with topological spaces [LS91] — we take A to be the set of points of a space, and regard elements of X as subsets of A, and the relation R is considered to be the membership relation. Thus the functions in the above functional view of X are the characteristic functions of the subsets, which we call *open sets* of the space. Note that seen this way, a Chu space is a generalization of a topological space, namely one in which there are no conditions on the set of open sets. Now a T_0 space is one in which given any two points, there is an open set containing one but not the other. Thus in a T_0 Chu space, for any two columns, there is a row in which the columns differ, so no two columns are identical. In the process view, this means that two events which always occur simultaneously are identified.

Any Chu space can be made extensional and T_0 by identifying any repeated rows or columns — this process is called standardization. In the rest of this thesis, whenever we need a Chu space with any of these properties, we will assume that it is standardized.

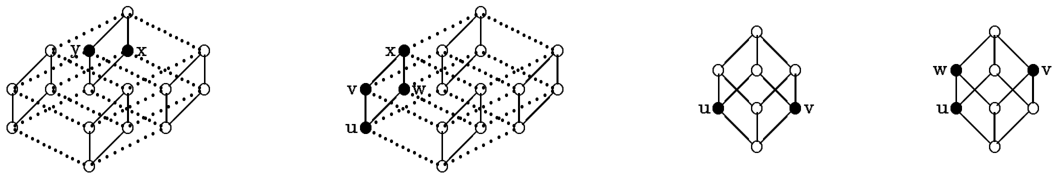
Two Chu spaces (A, X, R) and (B, Y, S) are said to be isomorphic if A is isomorphic to B, X is isomorphic to Y and R and S have the same value on corresponding pairs. Formally, an isomorphism between (A, X, R) and (B, Y, S) is a pair of functions $f : A \to B$ and $g : Y \to X$ such that both f and g are bijective and for all $a \in A, y \in Y$, $R(a, g(y)) = S(f(a), y)$. The reason for g going from Y to X will become clear in the next chapter.

3.2 Pictorial representation of Chu spaces

In the topological representation of an extensional Chu space discussed above, we saw that a Chu space could be completely specified by a set A and a subset of its power set $X \subseteq 2^A$. But a power set is a complete atomic boolean algebra, so for small A, we can draw a picture of 2^A as a Hasse diagram, and can represent the vertices of 2^A as blobs or holes according to whether they are present or absent in X respectively. We call this representation the *partial Boolean Algebra (pBA) representation.*

Note that if A is infinite, 2^A is a complete atomic Boolean Algebra, called a CABA. Completeness means that arbitrary meets and joins must be present, and being atomic means that every maximal chain has a second least element. All finite Boolean Algebras are complete and atomic, but this is not true for the infinite ones, for example the BA consisting of all the finite and cofinite subsets of the natural numbers is not complete.

This representation is called a partial Boolean Algebra because it may be presented as a Boolean Algebra, in which the operations of $\vee, \wedge$ and $\neg$ are partial operations, along with some equations. The underlying CABA 2^A provides a compact way of representing this information. It is also useful for determining homomorphisms, which we will talk about in the next chapter.



3.1(a) 3.1(b) 3.1(d) 3.1(e)

Figure 3.2: The four extensional Chu spaces of Figure 3.1, drawn as partial Boolean Algebras.

In the above figure we have drawn the four extensional Chu spaces in Figure 3.1 in the partial boolean algebra format. The notion of a T_0 Chu space can now be elegantly stated in terms of a lattice theoretic property:

Proposition 3.1 *An extensional Chu space is T_0 iff the elements of X (the dots) generate 2^A using $\cup, \cap$ and $\neg$, the usual operations of set union, intersection and complement. Empty unions and intersections are included.*

Proof: ($\Rightarrow$). We have to show that the atoms can be generated, since all other vertices are formed from these by the $\vee$ operation. Each atom corresponds to an element of the set A, namely the singleton set containing it.

Now given an $a \in A$, form the set $q_a = \bigcap\{x \mid a \in x \in X\} \cap \bigcap\{A - x \mid a \notin x \in X\}$. Then $a \in q_a$, since it is in each set x or $A - x$ in the RHS. Also, if $b \neq a$, then there is a set in X that separates them, by the T_0 property. This set ensures that $b \notin q_a$. Thus the atoms can be generated.

($\Leftarrow$). Given $a \neq b$, the singleton set containing a can be generated. So there must be an element in X separating a and b. ∎

Thus given a partial Boolean Algebra, it is easy to check if it is T_0, by making sure that the CABA is generated. This means that the CABA underlying a T_0 Chu space is the smallest such, and if we regard the underlying CABA as merely providing the information about meets and joins and complements, then this is the most economical way of doing it.

However, as we have seen above, even very small Chu spaces like figure 3.1 above can have very large partial Boolean Algebra representations. This leads us to a similar but more economical representation, the partial distributive lattice.

3.3 Partial Distributive Lattices

In the above section we saw that a Chu space could be written as a set X embedded in a CABA 2^A, such that X generated 2^A. But this means that for even very small Chu spaces, in which X may have only n elements, the CABA representation can be of size 2^{n+1}. An obvious question is if we can present the Chu space with less structure than the CABA 2^A, but containing the same information.

The following proposition [Joh82, p.250] enables us to answer this question in the affirmative. It states that up to isomorphism, every distributive lattice can be uniquely extended to a CABA, the least or canonical such CABA. So instead of drawing the whole CABA generated by 2^A, we need draw only the distributive lattice generated by X, the set of elements of 2^A generated by X using the operations union and intersection (which are meet and join in the lattice. In some cases, this permits up to an exponential savings in size.

For the infinite case, an additional requirement on the distributive lattice is that it be *profinite*. While this term may be applied to various algebras, and in fact CABA's can be described as profinite Boolean Algebras, for distributive lattices there is a similar simple characterization. A distributive lattice is called profinite if it is complete, and its completely join prime elements generate the entire lattice by joins. A completely join prime element $x \in X$ is one such that if $x \leq \bigvee Y, Y \subseteq X$, then $x \leq y$ for some $y \in Y$. (We allow infinite joins here.) For a distributive lattice this condition is equivalent to the condition $\bigvee\{y < x \mid y \in L\} < x$. Note that in a CABA the completely join prime elements are exactly the atoms, so a CABA could also be defined as a complete Boolean Algebra which is generated by its completely join prime elements.

We state and prove the proposition in elementary terms for greater accessibility.

Proposition 3.2 *If L is a profinite distributive lattice, then there is a CABA B and a complete lattice homomorphism $i : L \to B$ such that for any other CABA B' and a complete lattice homomorphism $f : L \to B'$, there is a unique CABA homomorphism $\hat{f} : B \to B'$ such that $f = \hat{f} \circ i$.*

Proof: The idea is to identify a subset of the elements of L and take these to be the atoms of B. Then L will embed in B. We will show that this embedding is unique up to isomorphism.

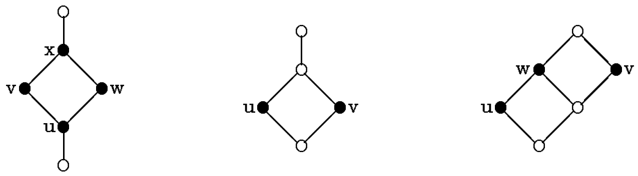
Let A be the set of join prime elements of L. Now we will form the CABA 2^A, and L is embedded in this CABA via $i : L \to 2^A$ defined by $i(y) = \{x \in A \mid x \leq y\}$. This function preserves all meets and joins of L (the proof is similar to that of [MMT87, thm 2.55, pg 83]), and is injective, as A generates L by definition.

Now given $f : L \to B'$, a complete lattice homomorphism, we can extend it to all of B by defining it on the atoms of B. If $a \in A$, define $\hat{f}(a) = f(a) \wedge \neg \bigvee\{f(x) \mid x < a, x \in A\}$. This is required as the atom corresponding to a is exactly $a \wedge \neg \bigvee\{x \in A \mid x < a\}$. Now this is extended to a CABA homomorphism $\hat{f} : B \to B'$. $f = \hat{f} \circ i$ since this is true for the primes $((a \wedge \neg b) \vee b = a \vee b)$. The uniqueness of $\hat{f}$ follows from the canonical definition. ∎

This proposition enables us to work with a smaller pictorial representation on the Chu Space than the partial Boolean algebra. It also makes the structure of the Chu space a lot clearer, and from now on, we will draw only these pictures, called *partial distributive lattices, or pdlats.* We will write a pdlat as a pair (P, L), where L is the lattice generated by X, and P is the set X along with the poset structure it inherits from L, as it is a subset of L. We need this poset structure for duality reasons, and these will be made clear in the next chapter. The pdlats corresponding to the Chu spaces shown above are in figure 3.3.

Notice that these pictures look quite like automata, as the dots are states and edges are events. We go into this in more detail in Chapter 5.

Proposition 3.2 gives us a simple algorithm to recover a Chu space from its pdlat representation. The elements of X are the black vertices of the pdlat, while the elements of A are the join-primes of the underlying lattice. The relation R is given by the $\leq$ relation, $R(a, x) = 1$ iff $a \leq x$ in the lattice.



3.1(b) 3.1(d) 3.1(e)

Figure 3.3: The 3 extensional T_0 Chu spaces of Figure 3.1 drawn as pdlats

Of course we would like to continue the process even further. One possibility is if we just take the poset generated by A. However this does not work, as the following pair of Chu spaces would then be represented by the same structure, even though they are not isomorphic. In fact, pdlats seem to be the best we can do here as there is no extension theorem like theorem 3.2 for other structures.

Figure 3.4: Incompleteness of Poset Structure

3.4 The logic representation

While the pictorial representations of Chu spaces above are useful for understanding what a Chu space looks like, they are still not very compact. An alternative way to specify a Chu space is via a formula, of infinitary propositional Boolean logic. This representation makes it possible to see the properties of the Chu space a lot more easily, so specification of the Chu space is simpler.

We assign a boolean propositional formula to each *extensional* Chu space as follows. The variables of the formula will be the elements of the set A. For each $x \in X$ we form a clause $\bigwedge\{a|a \in A, R(a,x) = 1\} \wedge \bigwedge\{\neg a|a \in A, R(a,x) = 0\}$. The disjunction of all such clauses then gives the propositional boolean formula. We call this the *complete disjunctive normal form* (abbr. CDNF) of a propositional formula. Given any formula and its set of variables, this form is unique up to commutativity, associativity and idempotence of $\wedge$ and $\vee$. Note that if A and X are infinite, the CDNF would need infinitary boolean connectives.

For the 3 extensional T_0 Chu spaces presented above, the boolean propositional formulae are given below. We use the short forms $\bar{a}$ for the negation of a and ab for $a \wedge b$.

$$\begin{array}{ll} 3.1(b) & \bar{a}\bar{b}\bar{c}d \vee \bar{a}\bar{b}cd \vee \bar{a}b\bar{c}d \vee \bar{a}bcd \\ 3.1(d) & \bar{a}\bar{b}c \vee a\bar{b}\bar{c} \\ 3.1(e) & a\bar{b}\bar{c} \vee \bar{a}bc \vee ab\bar{c} \end{array}$$

Of course the CDNF form is not an efficient way to write out the propositional formula, for example for the Chu space 3.1(a), $\bar{a}d$ is just as good as the long expression. In fact this shorter representation makes explicit that the b and c are not constrained in this formula. Any formula logically equivalent to the CDNF formula is an equally valid representation.

The columns of the matrix are exactly the satisfying assignments of the propositional formula. Each column makes the clause corresponding to its state true, so the whole formula becomes true. Moreover, any other assignment differs from each column on at least one variable, it makes each clause false, and thus the whole formula becomes false. So given a propositional formula, we can construct its Chu space as the set of all satisfying assignments.

The logical representation gives another way of drawing Chu spaces, as combinational circuits or gates. The inputs of a circuit representing (A, X, R) consist of the elements of A, and it has one output which calculates the boolean expression representing the Chu space. We will see this in more detail in chapter 5, and applications of the gate view to visualizing the Chu algebra will be discussed in chapter 6.

Theory of a Chu space. The set of all infinitary Boolean formulas that are consequences of the logical formula of a Chu space is called the *theory* of a Chu space. We will use the theory in the next chapter to define Chu morphisms as theory preserving renamings, and later use formulas in the theory to represent concurrent behavior.

3.5 Alternative presentations: duality

In the definition of a Chu space, A and X were symmetric. However in our presentation of a Chu space as a partial Boolean Algebra, we assumed that $X \subseteq 2^A$. Clearly, there is nothing special about this, and regarding A as a subset of 2^X is an equally valid way of drawing the Chu space as a partial Boolean algebra, with the consequent restriction to a partial distributive lattice. This is the essence of duality for Chu spaces, and we will talk about it in the next chapter. In order to distinguish it from the representation given earlier, we will call these the dual pBA or the dual pdlat representations. Both the pdlat representation and the dual pdlat representation will be important in representing a concurrent process as a Chu space.

In the same way, we can write logical expressions with variables corresponding to the elements of X. As above, we will call this the dual logical representation.

3.6 More examples

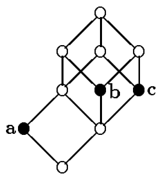
Having defined Chu spaces in various ways, we will now give pictures of all extensional T_0 Chu spaces in which $|A| \leq 3$, up to isomorphism. There are 78 such Chu spaces, 2 with $|A| = 0$, 4 with $|A| = 1$, 8 with $|A| = 2$ and 64 with $|A| = 3$. The corresponding numbers for $|A| = 4, 5$ are 3828 and 37320288, but we won't draw those! (Note that 3828 and 37320288 are *not* powers of 2!)

We show the pdlat form of a Chu space, then give its matrix and a simple logical representation. All Chu spaces with $|A| \leq 1$ are shown, but for 2 and 3, we use some symmetries to display only a few, and the others can be obtained by simple manipulation of the ones shown. Note that the four pdlats with $|A| = 1$ differ from each other only in whether the top or bottom elements are holes or dots. In the dual pdlats, this means presence or absence of spikes, while in the matrices it means presence or absence of the constant rows. In the logical form, it is equivalent to a formula being implied by Γ, the logical representation of the Chu space.

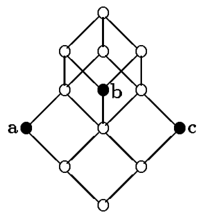
Pdlat repn	Dual pdlat repn	Logical repn	Matrix repn
Bottom is a hole	No spike at top	$\Gamma \rightarrow \bigvee A$	no row of 0's
Top is a hole	No spike at bottom	$\Gamma \rightarrow \neg \bigwedge A$	no row of 1's

Each of the above conditions can hold independently, giving rise to the 4 cases. In the examples here, we will present only the Chu space in which both of them hold, cutting down the number of spaces with $|A| = 2, 3$ to 2 and 16 respectively. Of the 16 Chu spaces with $|A| = 3$, 12 occur in pairs, where the matrix of one member of the pair is the bitwise complement of the other (for the others, the complement is an isomorphic matrix). Complementing the matrices turns the pdlats upside down. We will draw only one member of each pair, further reducing the 16 to 10.

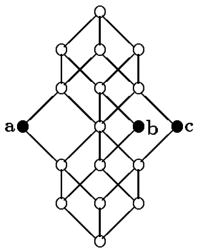
So the 18 Chu spaces are given in Figure 3.5. The dual pdlat at the top, and below it is a propositional formula representing the space. On some formulas a subscript indicates the number of variables in that formula, when written out in its CDNF form. below this to the left is the matrix, and to the right is the pdlat representation of the Chu space.



$a \wedge b \leftrightarrow a \wedge c$

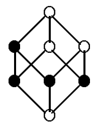


$b \rightarrow a \vee c$

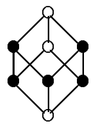


1_3

	abc
w	001
x	010
y	011
z	100



	abc
v	001
w	010
x	011
y	100
z	110



	abc
u	001
v	010
w	011
x	100
y	101
z	110

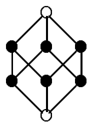


Figure 3.5 continued.

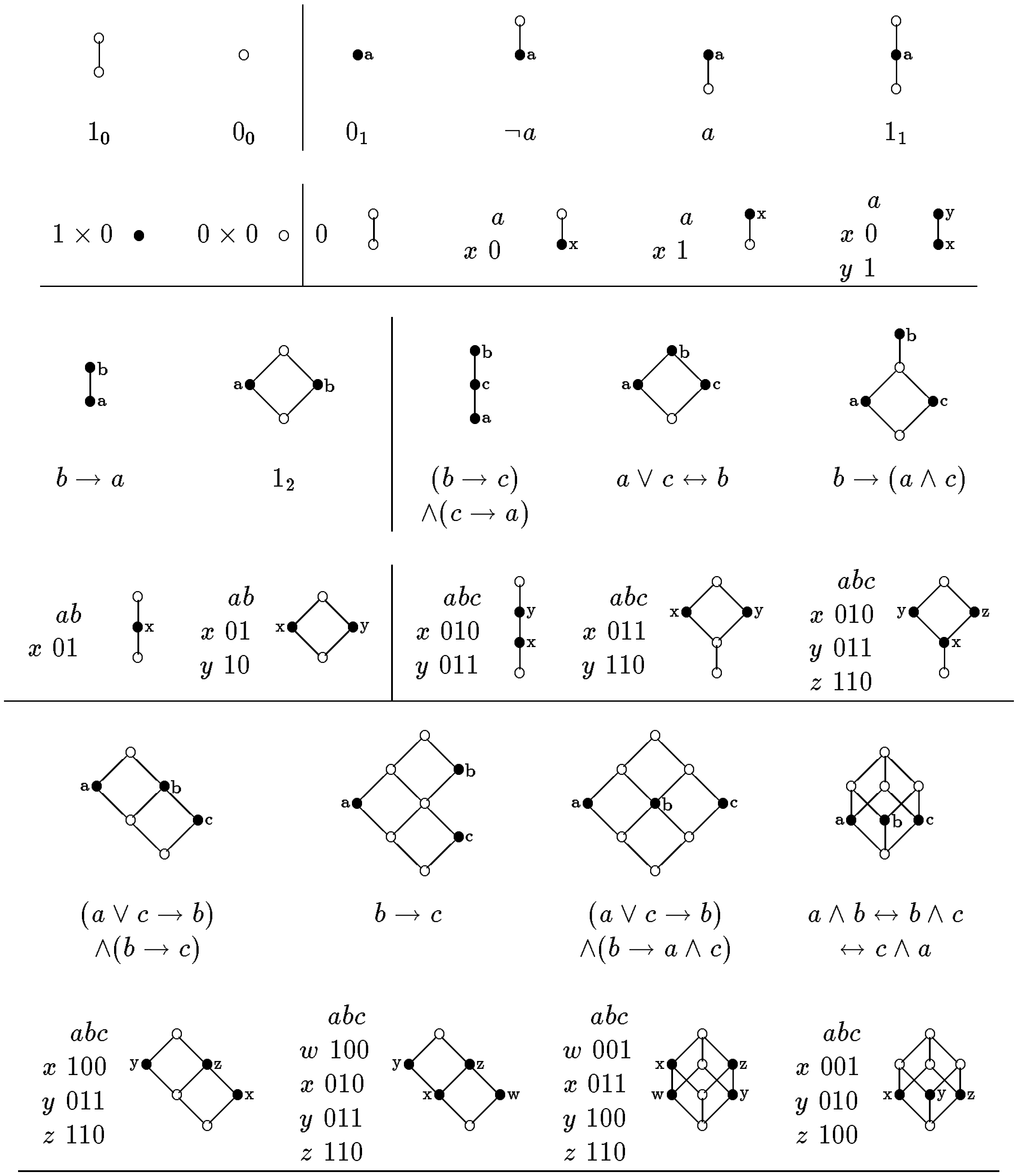


Figure 3.5: The Chu spaces with $|A| \leq 3$.

3.7 Chu's Construction: The categorical definition

Chu spaces first arose as an instance of a general construction, popularly called Chu's construction, described by Michael Barr and Po-Hsiang Chu in [Bar79]. In fact the name Chu spaces was suggested to Pratt by Michael Barr in an email message. Chu's construction is used to make $*$-autonomous categories from an autonomous categories. An autonomous category is a closed symmetric monoidal category, while a $*$-autonomous category is an autonomous category with a dualizing object. We will give a very brief outline of the construction here, and refer the reader to [Bar79] for details.

We start with a finitely complete closed symmetric monoidal category $\mathcal{V}$ and an object of $\mathcal{V}$, denoted $\perp$. The $*$-autonomous category constructed from these is denoted $\mathcal{V}_\perp$. Its objects are triplets (V, V', v) where V, V' are objects of $\mathcal{V}$, and $v : V \otimes V' \to \perp$. Its morphisms are pairs of morphisms from $\mathcal{V}$, $(f, f') : (V, V', v) \to (W, W', w)$ where $f : V \to W$ and $f' : W' \to V'$, such that $v \circ (1, f') = (f, 1) \circ w$.

Now Barr shows that the category $\mathcal{V}_\perp$ is a $*$-autonomous category, which embeds $\mathcal{V}$. The dual of any object (V, V', v) is $(V', V, v \circ s)$, where s is the symmetry isomorphism $s(A, B) : A \otimes B \to B \otimes A$, and the dualizing object is $(\perp, \perp^\perp, l)$, where l is the evaluation map. Since the category of sets and functions satisfies all the necessary properties for Chu's construction, we can apply Chu's construction to it with the two element set $\{0, 1\}$ as $\perp$, and the resulting category is the category of Chu spaces.

In [LS91], Lafont and Streicher studied the same category category, using K-valued matrices instead of 2-valued ones, so the relation R goes from $A \times X$ to K, for some integer K. They called this category $\mathbf{Game}_K$[1], and use it to construct a model of linear logic. They call these games as they view the elements of A and X as strategies for two players, and R is then a payoff function, thus giving a Chu space an interpretation as a von Neumann-Morgenstern game. This model is also similar to the Dialectica Categories of de Paiva [dP89], see [LS91]. The morphisms of the Dialectica categories are a bit different, there $R(a, g(y)) \leq S(f(a), y)$, where it is

[1]Thus Chu spaces are $\mathbf{Game}_2$

assumed that $0 < 1$. The duality of Chu spaces does not hold for general objects, only for decidable ones, and the tensor product is quite different: $(A, X, R) \otimes (B, Y, S) = (A \times B, Y^A \times X^B, T)$ where $T((a, b), (f, g)) = R(a, g(b)) \wedge S(b, f(a))$ for all $a \in A, b \in B, f : A \to Y, g : B \to X$.

Chapter 4

The Category of Chu Spaces

In this chapter we build up a category of Chu spaces. The objects were defined in the last chapter, and we had mentioned the morphisms also in the categorical definition. Here we will give simple characterizations of the morphisms for the different presentations, and present the duality of categories. Applications to algebra will also be mentioned.

4.1 Maps between Chu Spaces

Given two Chu spaces (A, X, R) and (B, Y, S), a Chu transform between them consists of a pair of functions $f : A \to B$ and $g : Y \to X$, such that the following equation holds for any $a \in A$ and any $y \in Y$: $S(f(a), y) = R(a, g(y))$. We will call this condition the adjointness condition for Chu maps, and f and g will be called the left and right adjoint respectively. Thus in a Chu space, the set A transforms covariantly, while X transforms contravariantly.

Just as we wrote out a Chu space as a matrix, we will write out a Chu map as another matrix. A Chu map between (A, X, R) and (B, Y, S) will be a $|Y| \times |A|$ matrix, each of whose $|A|$ columns are columns from (B, Y, S), the column corresponding to $a \in A$ being the column representing $f(a)$ in the matrix for (B, Y, S). Dually, each row is a row from (A, X, R), and the row corresponding to $y \in Y$ is the row representing $g(y)$ in the matrix for (A, X, R). An alternative way of specifying the matrix is to say that the (y,a)-th entry is $S(f(a),y)$, which is equal to $R(a,g(y))$.

Looking at the $|Y| \times |A|$ matrix as a Chu space (A,Y,T), Pratt points out that this Chu space is a factoring of (f,g) as $(f,g) = (f,Id) \circ (Id,g)$, where $(Id,g) : (A,X,R) \to (A,Y,T)$ and $(f,Id) : (A,Y,T) \to (B,Y,S)$. This and other factorings are discussed in [Pra93].

We will just give one example of a Chu map here, more examples will come up later. We will draw the space (A,X,R) on the top left, (B,Y,S) on the bottom right, and the matrix for the morphism on the bottom left.

$$\begin{array}{rc|cl}
 & abcde & & \\
u & 00001 & & \\
v & 00011 & & \\
w & 00101 & & \\
x & 01111 & & \\ \hline
 & 00011 & 0011 & u' \\
 & 00101 & 0101 & v' \\
 & & a'b'c'd' &
\end{array}$$

Figure 4.1: A Chu map in matrix form.

This is the matrix corresponding to the pair (f,g), where $f(a) = a', f(b) = a', f(c) = b', f(d) = c', f(e) = d'$, and $g(u') = v, g(v') = w$. In this case the first Chu space was extensional, so in fact f is uniquely determined by the matrix (there is a unique occurrence of any row on the top left matrix), and similarly as the second Chu space was T_0, g is also uniquely determined. Note that either of f or g uniquely determine the map matrix, so we observe that if $(f,g) : \mathcal{P} \to \mathcal{Q}$ is a Chu map, then f uniquely determines g if $\mathcal{P}$ is extensional, and g uniquely determines f if $\mathcal{Q}$ is T_0. [Bar91].

Maps between pdlats. Just as each T_0 extensional Chu space corresponds to a pdlat, we can define a pdlat map between two pdlats which corresponds to the Chu map between the corresponding Chu spaces.

Definition 1 Given two pdlats (P,L) and (P',L'), $f^\# : L \to L'$ is a pdlat homomorphism if $f^\#$ is a complete lattice homomorphism and $f^\#(P) \subseteq P'$.

The two definitions are equivalent, as is shown by the following proposition:

Proposition 4.1 *Let (A, X, R) and (B, Y, S) be two T_0 extensional Chu spaces, and let (P, L) and (P', L') be their corresponding* dual *pdlat representations. Then the set of Chu maps from (A, X, R) to (B, Y, S) is in 1-1 correspondence with the set of pdlat maps from (P, L) to (P', L').*

Proof: Note that in the dual pdlat representation, L is a sublattice of 2^X and L' is a sublattice of 2^Y. Now if (f, g) is a Chu map, then $g : Y \to X$. Define $2^g : 2^X \to 2^Y$ by if $a \in 2^X$, then $2^g(a) = \{y \mid g(y) \in a\}$ (Note here that a is a subset of X). This is a CABA homomorphism. Now the adjointness condition stated above means that $g(y) \in a \leftrightarrow y \in f(a)$, where we regard $A \subseteq 2^X$ and $B \subseteq 2^Y$. Thus $2^g(a) = \{y \mid y \in f(a)\} = f(a)$. Now define $f^\# = 2^g \restriction_L$. Then $f^\#(P) \in P'$, as $f^\#(a) = f(a) \in B$ for all $a \in A$. Now since L' is generated by B, the range of $f^\#$ is in L', so it is a pdlat map. (It is a complete lattice homomorphism as it is the restriction of a CABA map.)

Conversely, every pdlat map corresponds to a Chu map, by reversing all the implications above. Thus we have the desired 1-1 correspondence. ∎

This proposition shows that the definition of a Chu map is a natural one. This is further substantiated in the next few sections.

Transformations between logical representations. Another equally simple way of representing Chu maps is as transformations between their logical representations. Given the logical CDNF representations Γ and Δ of two Chu spaces, we define a logical transformation between them to be a map $f : var(\Gamma) \to var(\Delta)$[1] such that $\Delta \to f(\Gamma)$ is a logical tautology (here $f(\Gamma)$ is the formula obtained by substituting each variable a of Γ by its image $f(a)$.).

This representation means that every formula in the theory of the source (the theory consists of all consequences of Γ) is mapped to a formula in the theory of the target (all consequences of Δ). So it is a logical homomorphism — if the formulas in the theory are regarded as the constraints on the Chu space, then a homomorphism should preserve all of them, which is exactly what it does.

Proposition 4.2 *Chu maps are in 1-1 correspondence with logical transformations.*

[1] $var(\Gamma)$ is the set of variables in the formula Γ.

Proof: Let (A, X, R) and (B, Y, S) be two T_0 extensional Chu spaces, and let Γ and Δ be their logical representations. If $(f, g) : (A, X, R) \rightarrow (B, Y, S)$ is a Chu map then we define a logical transformation f via its first component. Now consider any satisfying assignment $y \in Y$ for Δ. Then $g(y) \in X$, so it forms a satisfying assignment for Γ. Thus y is also a satisfying assignment for $f(\Gamma)$, because of the adjointness condition. So $\Delta \rightarrow f(\Gamma)$.

Conversely, given a logical transformation f, it determines a unique map $g : Y \rightarrow X$, since every satisfying assignment of Δ is a satisfying assignment of $f(\Gamma)$, so every $y \in Y$ is mapped to $x \in X$ such that $f(x) = y \cap f(A)$. ∎

This shows that the three forms of presenting Chu spaces each have their natural morphisms, which are all equivalent. Chu maps also have a natural interpretation as simulations between processes, and we will show this in a later chapter.

Chu spaces and Chu maps form a category, denoted **Chu**. The identity map for an object is a pair of identity functions on its first two components, while $(f, g) \circ (f', g') = (f \circ f', g' \circ g)$. Some of the mathematical structure of this category is further explored in [Pra93], where Pratt shows how to embed various categories fully and faithfully in **Chu**.

4.2 Duality

In the previous chapter we had indicated briefly a duality for Chu spaces. The Chu construction was invented specifically for obtaining self-dual categories from autonomous categories [Bar91]. Here for reference, we will restate this duality, and show how to get the dual of a pdlat. In the next chapter we will apply this duality to transform imperative programs into declarative ones and vice versa.

Definition 2 The dual of (A, X, R), denoted $(A, X, R)^{\perp}$, is defined as $(X, A, R^{\smile})$, where $R^{\smile}(x, a) = R(a, x)$.

Thus the dual of a Chu space written out as a matrix is the transpose of this matrix. If $(f, g) : (A, X, R) \rightarrow (B, Y, S)$ is a Chu map, then $(g, f) : (Y, B, S^{\smile}) \rightarrow (X, A, R^{\smile})$ is also a Chu map by the symmetry of the definition, thus showing that this is a categorical duality. The involution $\mathcal{P}^{\perp\perp} \cong \mathcal{P}$ for any Chu space $\mathcal{P}$ follows from the definition.

The dual of a pdlat can be written directly as $(P, L)^{\perp} \cong (L^{\perp}, P^{\perp})$—this is the dual pdlat representation. Here $L^{\perp}$ is the set of all filters of L [Pri70], partially ordered by set inclusion. (A filter is an upwards closed subset of the lattice which is closed under all meets.) $P^{\perp}$ is the set of all upwards closed subsets of P, and it forms a lattice with the operations of set union and intersection. $L^{\perp}$ embeds in $P^{\perp}$ by sending each element $a \in L^{\perp}$ to $a \cap P$ which is an element of $P^{\perp}$, and this is injective because P generates L.

In Figure 3.5 for each pdlat we have given its dual pdlat above it. Notice that each element of P in a pdlat corresponds to a dimension of the dual pdlat, thus as we remove elements from P, we collapse the dual pdlat along the corresponding dimensions. This connection is explored in [NPW81, p.94], where a dimension is defined to be an equivalence class of prime intervals, and each dimension corresponds to an event.

Lemma 4.1 *If (P, L) is the pdlat for the Chu space $\mathcal{P}$, then $(L^{\perp}, P^{\perp})$ is the pdlat corresponding to $\mathcal{P}^{\perp}$.*

Proof: Straightforward, using the fact that $L^{\perp\perp} \cong L$ and $P^{\perp\perp} \cong P$, where $L^{\perp}$ is defined as the set of complete filters of L, ordered by inclusion, and $P^{\perp}$ is defined as the set of order filters of P, again ordered by inclusion and regarded as a lattice. ∎

4.3 The internal Hom functor

The set of all maps between two Chu spaces $\mathcal{P}= (A, X, R)$ and $\mathcal{Q}= (B, Y, S)$ can itself be turned into a Chu space. If (C, Z, T) is used to represent this Chu space, then C is the set of all maps from $\mathcal{P}$ to $\mathcal{Q}$ and $Z = (A \times Y)$, the cartesian product of A and Y. Now if $(f, g) \in C$, and $(a, y) \in Z$, then $T((f, g), (a, y)) = S(f(a), y) = R(a, g(y))$. We will denote (C, Z, T) by $\mathcal{P} \multimap \mathcal{Q}$.

Note that $\mathcal{P} \multimap \mathcal{Q}$ may not be extensional even if both $\mathcal{P}$ and $\mathcal{Q}$ are extensional.

Whenever extensionality is required, we can just identify equal rows by standardization, making the whole space extensional. However if $\mathcal{Q}$ is T_0, we can show that $\mathcal{P} \multimap \mathcal{Q}$ is also T_0. This is because for distinct elements $(f,g),(f',g') \in C$, there is some $a \in A$ such that $f(a) \neq f'(a)$. Now by the T_0 property of $\mathcal{Q}$, there is $y \in Y$ which separates $f(a)$ and $f'(a)$. Then (a,y) separates $(f,g),(f',g')$.

We can now prove that our duality is a representable duality, represented by the Chu space $\perp = (\perp_A, \perp_X, \perp_R)$, with $\perp_A = \{a,b\}$, $\perp_X = \{*\}$ and $\perp_R(a,*) = 0, \perp_R(b,*) = 1$.

Lemma 4.2 *If* $\mathcal{P} = (A,X,R)$ *is a Chu space, then* $\mathcal{P}^\perp \cong \mathcal{P} \multimap \perp$.

Proof: Let $\mathcal{P} \multimap \perp = (B,Y,S)$. For each element $x \in X$, define a map $f_x : A \to \perp_A$ by $f_x(a) = R(a,x)$, and a map $g_x : \perp_X \to X$ by $g_x(*) = x$. Now (f_x, g_x) is a Chu map, so for each $x \in X$ we have given a Chu map. It is clear from the definition of a Chu map that these are all the maps from $\mathcal{P}$ to $\perp$. Thus $B \cong X$.

By definition of $P \multimap \perp$, $Y = A \times \{*\} \cong A$. Now $R^{\smile}(x,a) = R(a,x) = R(a, g_x(*))$ $= S((f_x, g_x),(a,*))$. Thus the two Chu spaces are isomorphic.

This isomorphism is natural in $\mathcal{P}$. If we have a map $h = (h,k) : \mathcal{Q} \to \mathcal{P}$, and the isomorphism above is denoted by $\eta_{\mathcal{P}} : \mathcal{P}^\perp \to \mathcal{P} \multimap \perp$, then $\eta_{\mathcal{Q}} \circ h^\perp = (h \multimap \perp) \circ \eta_{\mathcal{P}}$. Given any $x \in X$, $\eta_{\mathcal{Q}} \circ h^\perp(x) = \eta_{\mathcal{Q}}(k(x)) = f_{k(x)}$. Now for $b \in B$, $f_{k(x)}(b) = S(b,k(x)) = R(h(b),x) = f_x(h(b))$. This is exactly $(h \multimap \perp) \circ \eta_{\mathcal{P}}(x)(b)$. We can also show this for the right adjoint of the maps, showing that the isomorphism is natural. ∎

We have already shown that the duality is categorical, giving the identity $\mathcal{P} \multimap \mathcal{Q} \cong \mathcal{Q}^\perp \multimap \mathcal{P}^\perp$.

A third identity which is rather useful is $\mathcal{P} \multimap (\mathcal{Q} \multimap \mathcal{R}) \cong \mathcal{Q} \multimap (\mathcal{P} \multimap \mathcal{R})$. The proof is straightforward.

The internal hom functor, as expected, is contravariant in the first argument and covariant in the second. The action of this functor on morphisms is by composition.

Chapter 5

Concurrency Applications

Now that we have established all the mathematical background for Chu spaces, we will look at their applications to concurrency in the next three chapters. We have already given a brief idea of this in the introduction, here we provide the details. In this chapter we will interpret individual Chu spaces as processes, leaving the algebra of Chu spaces to the next chapter.

5.1 Behavior accepted by Chu spaces

We have already mentioned how to interpret a Chu space as a process—a Chu space (A, X, R) is interpreted as the process whose set of events is A, the set of possible states is X, and the events that have occurred in a state $x \in X$ are given by $\{a \in A \mid R(a, x) = 1\}$. So a process starts off in some state x, then executes some events to reach another state x', and this is how execution progresses. Each state keeps the entire history of the process, but does not remember the order in which the events took place.

In the rest of this thesis, we will write a Chu space as a pair (A, X), where $X \in 2^A$, with R being implicit as the membership relation. When we need to write the space explicitly, we will write the elements of A as a set, and the elements of X as a set of repetition-free words over A. So for example we would write $(\{a, b, c\}, \{\epsilon, a, b, abc\})$

for the Chu space

$$\begin{array}{cc} & abc \\ w & 000 \\ x & 010 \\ y & 100 \\ z & 111 \end{array}$$

The adjointness condition for maps will then become $f(a) \in y$ iff $a \in g(y)$.

The pdlat representation gives a pictorial view of the process, viewed as an automaton. The states are given by the black dots, and the edges of the lattice give the transitions. We call the holes the *forbidden states*, as they are not states in which the system can rest. Computation proceeds in the upwards direction, so can be looked upon as a path in the automaton. Each edge is labeled by an event; the label can be determined by looking at the sets of events labeling the extremities of the edge (a hole can also be labeled by a set of events, the only difference being that that set is not a state in X).

According to the execution model given above, a process can go from any state to any other state that is above it. So the computation path does not necessarily need to follow the arcs of the lattice — that would yield an interleaving model for a Chu space. We can assume some of the boxes in the lattice as solid boxes, just like the higher dimensional automata discussed in chapter 2. Now a computational path can be any path in this n-dimensional space. The n-dimensional boxes that have all their 2^n vertices colored black are always filled up, other boxes may or may not be. In particular, boxes in which the top vertex is missing, and which do not have any black dot above the top vertex are never filled up, as they represent a conflict.

So now a path is given as a sequence of states in the automaton representing the process. This kind of semantics has also been discussed before, and is called step semantics [DDNM88]. This is the kind of semantics implicit in Petri nets [Rei85, p.20] and SCCS [Mil83]. However notice that assuming a path through the automaton means that at every instant we know exactly where each event is in its computation, or if the process is in some state, that this state is exactly known. This is not always possible or desirable—in a distributed environment it is not possible to know what state each component of a system is in precisely, nor is it necessary to know this, since it may be useless detail (for example in $a||b$ we don't need to know how much of a and b have been finished at any time). So we broaden our class of computations to one in which such we can abstract away from these details — we take a computation to be a *homotopy class* of monotone paths [Pra91, GJ92], that is a broad path free of holes, where a hole denotes a choice of which side one can go around.

A homotopy class of monotone paths is defined as a set of monotone paths, each of which can be deformed into any other smoothly i.e. without having to jump over holes. Formally, if L is the set of points in the lattice of the automaton (states, edges and interior points) partially ordered, then a path is a monotone continuous function $p : \mathbb{R} \to L$. The a set of paths P is a homotopy class if for every $p_1, p_2 \in P$, there exists a continuous $F : [0,1] \to P$ such that $F(0) = p_1$ and $F(1) = p_2$. Such a class denotes a deterministic computation — all choices have been resolved.

In practice we sometimes would like to ignore some choices, since they may not be relevant. For example, we may not want to remember which event occurred first in a mutual exclusion process. Thus we can generalize a computation to a path which does have some small holes.

Thus given a Chu space we can informally define a behavior to be any subset of its set of states (the paths can be inferred from the states in the behavior). More precisely, a behavior of (A, X) is another Chu space (A, Y), where $Y \subseteq X$. The whole process is given by the disjunction of all its behaviors, somewhat reminiscent of the trace models.

Labeling. Until now we have not considered the actions that a process executes, only the events, which were earlier defined to be occurrences of actions. We now introduce a *labeled Chu space* as a Chu space (A, X) with a labeling function $\lambda : A \to Act$, where *Act* is a set of possible actions. We will not insist that Chu maps preserve labels, though we will sometimes consider the class of Chu maps that do preserve labels i.e. $\lambda'(f(a)) = \lambda(a)$.

5.2 Dual view of a Chu space — Schedules

In the above section we have interpreted the pdlat corresponding to a Chu space as an automaton. The dual pdlat of a Chu space provides an alternative way of looking at a Chu space, as a schedule of events. The black blobs in the dual pdlat are events, and the underlying lattice structure gives the constraints between the events.

The duality between schedules and automata is the consequence of a deeper duality between events and states themselves. Traditionally events are regarded as atomic, and states as sets of events, as in event structures and pomsets. Taking this dual view allows us to consider states as atomic, and events as sets of states, with a state belonging to an event iff the event has occurred in that state. If we are willing to be more adventurous with foundations of set theory, we may do both— the resulting circularity in the membership relation can be coped with by dropping the Foundation Axiom of ZF set theory and replacing it by e.g. Aczel's Anti-Foundation Axiom. This results in states and events being treated on the same footing, something which has not been done in previous models.

Time and Information Duality. Pratt [Pra92b] has observed a duality between time and information based on the intuitions behind event-state duality. Events are regarded as instantaneous, occurring at a particular time, and they add to the total accumulated information. Dually, no information is accumulated in a state, which however adds to the total amount of time that has passed since the beginning of the process. Thus we can plot the progress of a behavior on a graph with time on the x-axis and information on the y-axis. The states are then horizontal line segments, and the events are vertical. The graph is monotonically increasing, as we take computation to be monotonic. The graph is also piecewise horizontal and vertical, but as the number of events grows to infinity it starts looking like a monotonically increasing curve.

In figure 3.5 we have drawn the pdlats (automata) and dual pdlats (schedules) corresponding to some Chu spaces. It can be seen that in the schedules, time is flowing downwards, as opposed to the automata, where time flows upwards. So we start off by doing events at the top of the lattice, and work our way down. The lattice defines the constraints — if an event is below another event, it must be done later. If an event is the join of two events in the lattice, then as soon as it is done either of the two events of which it is the join must also be done. This follows from our definition of a dual pdlat, since the states of its Chu space are just its filters intersected with its event set, so any equation like $a \vee b = c$ must be satisfied in all states.

Dual to the notion of a behavior, we define a *property* of a Chu space (A, X) to be any other Chu space (A, Y), where $X \subseteq Y \subseteq 2^A$. For example, the space $(\{a, b, c\}, \{\epsilon, a, b, abc\})$ is a property of the Chu space $(\{a, b, c\}, \{a, b, abc\})$. A Chu space is thus a conjunction or intersection of all its properties.

We will represent properties by infinitary boolean expressions, so the property ϕ of a Chu space (A, X) will represent the Chu space (A, Y), where $Y \subseteq 2^A$ is the set of all satisfying assignments of ϕ (see chapter 3). The above property can then be written as $a \vee b \equiv c$. Now if Γ is the logical formula representing a Chu space, then the Chu space has the property ϕ iff $\Gamma \models \phi$, that is $\Gamma \rightarrow \phi$ is a valid boolean formula. The set of all properties of a Chu space is its theory, which was defined as the set of logical consequences of its logical formula, and can be viewed as the set of Chu spaces with the same A but with $Y \supseteq X$ (it has $2^{(2^A - X)}$ elements).

Since there is no way of telling whether a pdlat is intended to be interpreted as a schedule or as an automaton, we will use an arrow to indicate the direction of time. An arrow pointing upwards means that the pdlat is an automaton, and a downward arrow means that it is a schedule (as time flows down in a schedule). This arrow also identifies whether a pdlat is to be interpreted as a pdlat or a dual pdlat.

5.3 Logical representation of Chu spaces — Gates as acceptors

In chapter 3 we gave the logical representation of a Chu space as an infinitary boolean formula, whose variables are the events and whose satisfying assignments are the states of the Chu space. The logical representation of a Chu space can be understood by letting each variable a stand for the proposition "a has happened". Thus execution of the process sets more variables to 1, as more events keep happening. The entire formula is built up of these atomic propositions, and must remain true at all times — that is the process must remain in a state.

Now we can represent a Chu space as a Boolean gate. Its inputs are the events of the space, and the gate computes the Boolean formula for the Chu space, and has one output. The occurrence of an event can now be considered as a toggling of an input. Since computation is monotonic, an input can be toggled from zero to one only, never back, so in a single run, it can be toggled exactly once, corresponding to the event taking place. A behavior is accepted if the output always stays 1, that is the process is always in an allowed state. So a gate is now an acceptor of a behavior, rather than a transducer as used traditionally [GP93].

Gates provide an alternative pictorial representation for Chu spaces, and are very useful in understanding the algebra of Chu spaces, as we will show in the next chapter.

5.4 Aspects of concurrency

We will now show some properties that can be used to represent various features of concurrent behavior in a process. We use Γ for the logical representation of a process, and boolean formulas for the properties.

Temporal precedence. For any two events a, b, if $\Gamma \models b \rightarrow a$, then no state contains b and not a. This means that b can be executed only after a has been executed. This means that a precedes b in time, and the collection of all such constraints defines a temporal order on the events of G. We write $a \leq b$ for any such pair of events. Note that the temporal order is the converse of the logical order, and this is the reason for the downwards flow of time for the schedule. (We always have the logical order going up.) We cannot express the property $a < b$ in our model, since this is an asynchronous model. According to our semantics, a process represented by a Chu space may evolve from any state to any other state above it, so even if $a < b$, it can in one transition go from a state in which neither is done to a state where both are done, so they occur at the same time.

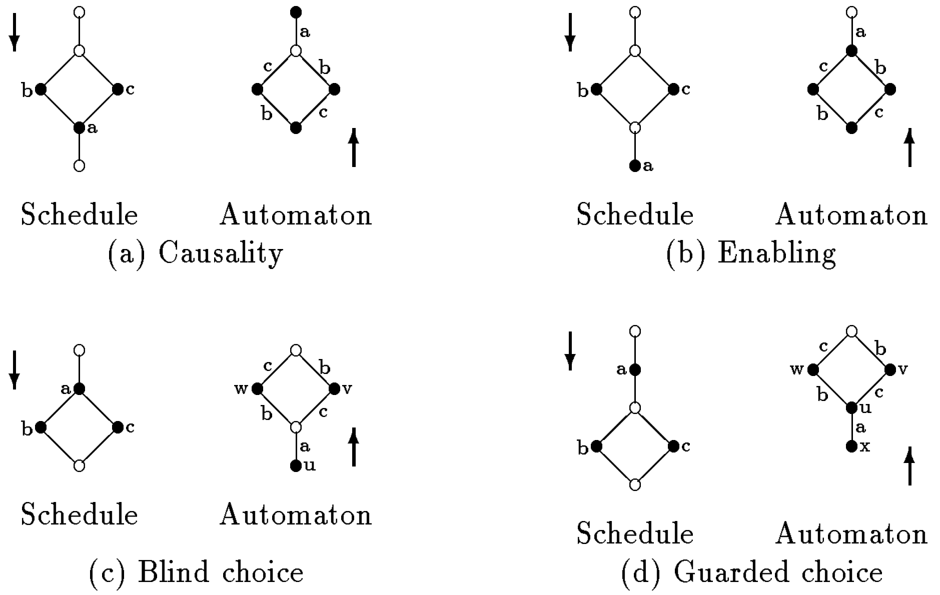


Figure 5.1: Schedules and Automata for some concurrent behaviors.

Conflict. If $\Gamma \models \neg(a \wedge b)$, then no state contains both a and b. This means that we can never execute both a and b, which is interpreted as a *conflict* between a and b. This can be generalized to conflict between arbitrary numbers of events. While this notion has been explicitly introduced in event structures [NPW81, Win86], it follows quite naturally from our definition. It is illustrated in figure 5.1(c-d), where b and c are in conflict.

Causality and enabling. If $\Gamma \models a \leftrightarrow (b \wedge c)$ then any state containing b and c must also have a. We interpret this as b and c together *cause* a, that is, as soon as the events b and c have been done, the environment will immediately do a. This can be generalized to an arbitrary formula on the right side of the implication. Note that the T_0 condition does not allow an event to be caused by a single event, as the two events are identified.

The above Chu space is distinct from the Chu space in which $\Gamma \models a \rightarrow (b \wedge c)$. Then b and c just *enable* a, which means that while it is necessary for them to be done to do a, it is not sufficient, as a need not be done immediately. The enabling condition $(b \wedge c)$ can be generalized to an arbitrary condition, including the singleton event, when it becomes temporal precedence. The difference between these is like the difference between the necessary and sufficient conditions for a theorem—the enabling formula represents the necessary conditions, the causing formula is both necessary and sufficient.

This distinction has not been made in other models. An example of this difference is the difference between two candy machines — an automatic machine would give a candy immediately after putting in a quarter and a dime, so the event of getting the candy is caused by the events of putting in a dime and the event of putting in a quarter. A manual machine would enable a user to press a button after putting in a quarter and a dime, and the machine would then give the candy. Thus in the second case the user can wait after putting in the money! (Here the pushing the button and getting the candy is regarded as one event).

Nondeterminism. Dual to the above notion is the notion of guarded vs. blind choice. If $\Gamma \models a \leftrightarrow (b \vee c)$, then as soon as a is done, one of b or c must be done immediately. This means that any information obtained by doing a could not be used in selecting between b and c, making this a *blind* choice. This is more clearly visible from the automaton side in Figure 5.1(c). The choice between states v and w has to be made at the forbidden state above u. We interpret this as being made by the environment, making this a nondeterministic choice for the automaton.

Just having $\Gamma \models (b \vee c) \rightarrow a$ would make it a *guarded* choice, whence any information obtained from a can be used to make a deliberate choice. On the automaton side, the choice between v and w is made in the state u, as $\Gamma^{\perp} \models u \leftrightarrow v \wedge w$[1], and this state contains the information gathered by doing a to decide which way to go. This is like a conditional branch in a program, where the event a checks some condition, and then either one of two alternatives is chosen.

This distinction is similar to the distinction between internal and external nondeterminism made in CSP[Hoa85], in both cases the difference being between who makes the choice, the environment or the process.

Disjunctive Enabling. If $\Gamma \models c \rightarrow a \vee b$, then in order for c to happen, at least one of a or b must have happened. For example, in a candy machine, inserting either a dollar bill or 4 quarters will enable the machine to supply a candy bar. This is

[1] $\Gamma^{\perp}$ is the dual logical representation, and has the names of states for the set of variables.

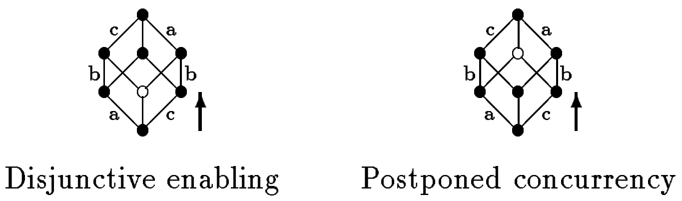


Figure 5.2: Automata for disjunctive enabling and postponed concurrency

called disjunctive enabling [Win86, Gun91, Gun92]. The dual behavior is *postponed concurrency*, given by $\Gamma \models a \wedge b \rightarrow c$, whereby a and b can be done concurrently only after c is completed. For example, c could be the event that makes available an additional resource, e.g. an extra processor, which allows concurrent execution. The automata for processes with disjunctive enabling and postponed concurrency are given in figure 5.2. The schedules can be derived from the dual pdlat of the formula $b \rightarrow c \vee a$ in figure 3.5.

5.5 Morphisms as simulations

If we interpret Chu spaces as processes, we can interpret the morphisms between them as simulations[2]. Thus when we have a Chu map $(f,g) : (A,X) \rightarrow (B,Y)$, (A,X) can simulate (B,Y) via the simulation (f,g). By a simulation of (B,Y) by (A,X) we mean here a relation $S \subseteq Y \times X$ between states and a correspondence C between computations such that whenever ySx and $y \stackrel{\alpha}{\rightarrow} y'$, where α is a set of actions, then there exists $x' \in X$ such that $y'Sx'$ and $x \stackrel{C(\alpha)}{\rightarrow} x'$. A computation here will be a set of events, so C will be a function from 2^B to 2^A.

Proposition 5.1 *Every Chu map $(f,g) : (A,X) \rightarrow (B,Y)$ determines a simulation of (B,Y) by (A,X).*

Proof: Given $y \in Y, x \in X$, define $S \subseteq Y \times X$ by ySx iff $x = g(y)$. Define C as f^{-1} : if $\alpha \subseteq B$, then $C(\alpha) = \{a \mid f(a) \in \alpha\}$.

[2] Brown, Gurr and de Paiva [BGdP91] have done the same for their category of Petri nets.

Now suppose $y \xrightarrow{\alpha} y'$ in the process (B, Y). Then $y' = \alpha \cup y$ (note that $y, y' \subseteq B$), so $C(y') = C(\alpha \cup y)$.

Now $a \in C(y')$ iff $f(a) \in y'$ iff $a \in g(y')$, by the adjointness condition for maps. Thus $C(y') = g(y')$. (Thus C is just an extension of g to all subsets of B.)

$$\begin{array}{lll} \text{Also, } a \in C(\alpha \cup y) & \text{iff} & f(a) \in \alpha \cup y \\ & \text{iff} & f(a) \in \alpha \text{ or } f(a) \in y \\ & \text{iff} & a \in C(\alpha) \text{ or } a \in g(y). \end{array}$$

Thus $g(y') = C(\alpha \cup y) = C(\alpha) \cup g(y)$, so $g(y) \xrightarrow{C(\alpha)} g(y')$. Thus S is a simulation. ∎

We can now see the connection between these simulations and the usual definition of simulations between processes — S is a simulation between processes if pSq and $p \xrightarrow{\alpha} p'$ then there exists q' such that $q \xrightarrow{\alpha} q'$ and $p'Sq'$, where α is a set or sequence of *actions*, not events. We shall use maps between labeled Chu spaces.

Corollary 5.1 *If $(f, g) : (A, X) \to (B, Y)$, and f is injective and preserves labels, and the label of any event not in the range of f is τ, the silent action, then (f, g) determines a simulation.*

Proof: If $\lambda : A \to Act$ and $\lambda' : B \to Act$ are the labeling functions, and $\lambda'(f(a)) = \lambda(a)$, then $\lambda(C(\alpha)) = \lambda'(\alpha)$, since any event in α whose pre-image is not in $C(\alpha)$ is labeled τ. ∎

Refinements. A special class of simulations is *refinements*, in which some events of a process are refined into processes themselves. These are useful in program building, where one may start with a high level view of the program, and then successively refine it to get the entire program. As a programming construct, refinements are rather like procedure calls, where a line of code on closer inspection turns out to be a big procedure. The fact that we may not know the internal structure of the procedure (e.g. calling a library function) imposes a non-interleaving semantics on us, as discussed in chapter 2.

We treat here the simple case of refining events by conflict free processes, leaving the more general refinements to the next chapter. A conflict free process (A, X) is one in which no subset of A is in conflict (has meet 0), or equivalently is one in which

$A \in X$. We will also insist upon the additional condition that $\emptyset \in X$, so there is a unique start state.

Given a Chu space (B, Y) and an event $b \in B$, we define the refinement of b by the conflict free space (C, Z) as the process (A, X), where $A = B - \{b\} \cup C$, and X is defined as follows :

$$\begin{aligned} X = \quad & \{x \mid b \notin x \in Y\} \\ & \cup \{x - \{b\} \cup C \mid b \in x \in Y\} \\ & \cup \{x - \{b\} \cup z \mid b \in x \in Y, z \in Z, x - \{b\} \in Y\} \end{aligned}$$

Now we can show a map $(f, g) : (A, X) \to (B, Y)$. $f(a)$ is a if $a \in A - C$, and $f(a) = b$ if $a \in C$. g is defined via the adjointness condition, and the fact that $C \in Z$ ensures that (f, g) is a Chu map. The intuition behind the simulation is that whenever an event other than b is executed in (B, Y), the same event is executed in (A, X). When b is done, the whole process (C, Z) is executed.

Here we have defined refinement of a single event. Composition of refinements now yields refinements of many events at a time, and also compound refinements i.e. refinement of refining events.

5.6 Programming an automaton or a schedule

The two different ways of looking at processes, schedules and automata, yield two different ways of writing programs. We can either specify a schedule by writing out all the constraints on the events, resulting in a declarative style of programming, or can specify how the process evolves from a state, giving an imperative way of programming. The duality of Chu spaces now enables us to convert between these two forms of programming without any loss of information.

The declarative style of programming allows us to specify the properties of a process, the conjunction of all of these then results in the specification of the whole process. For example, suppose we wanted to write a Chu space for a mutex process with decision events, that is, we have two events a and b to be done in mutual exclusion, along with a pair of events d_{ab} and d_{ba} that resolve which is to be done

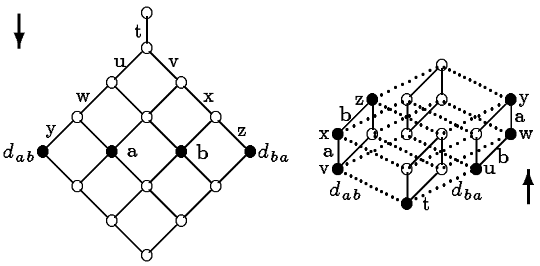


Figure 5.3: Schedule and Automaton for the mutual exclusion process.

first. Clearly, d_{ab} and d_{ba} are in conflict, so $d_{ab} \wedge d_{ba} = 0$. Also the decision is to be made before doing the events, so we get $a \vee b \to d_{ab} \vee d_{ba}$. Now d_{ab} means a should be done before b so we have $d_{ab} \to (b \to a)$. There is a symmetric condition for d_{ba}, namely $d_{ba} \to (a \to b)$. Conjoining these four conditions gives us our program.

An alternative way is to specify how the process evolves. In the beginning nothing is done, so the state is $t = \emptyset$. Then a decision is made, so the process can be in state $v = \{d_{ab}\}$ or in state $u = \{d_{ba}\}$. From state $\{d_{ab}\}$, it does a and then b, so we get states $x = \{a, d_{ab}\}$ and $z = \{a, b, d_{ab}\}$, and from $\{d_{ba}\}$ it does b and then a, so we get states $w = \{b, d_{ba}\}$ and $y = \{a, b, d_{ba}\}$.

Now we can draw the Chu spaces we get from these two specifications. Duality enables us to show that they result in the same process.

In the next chapter we will present an algebra of Chu spaces, which will give us a way of building bigger processes from smaller ones modularly. So we will be able to specify the smaller processes using one of the methods above, and then combine them using the algebraic operators, giving a powerful specification language allowing us to use imperative, declarative and algebraic specifications in one coherent framework. As an example, we will be able to write the mutual exclusion of two processes as $\mathcal{P};\mathcal{Q} \sqcup \mathcal{Q};\mathcal{P}$, or as $(d_1 \sqcup d_2);(\mathcal{P}+\mathcal{Q}) \wedge (d_1 \wedge \bigvee B \to (\mathcal{P} \text{ in maximal state})) \wedge (d_2 \wedge \bigvee A \to (\mathcal{Q} \text{ in maximal state}))$, where $\mathcal{P} = (A, X)$, and $\mathcal{Q} = (B, Y)$.

Chapter 6

Algebra of Chu Spaces

Having defined the category of Chu spaces in the previous chapters, we immediately get an algebra of Chu spaces, as our category has products and coproducts, tensor and duality. We will give elementary definitions of these operations here, along with some others, and show how they can be interpreted as constructors for making large programs out of smaller ones, giving a process algebra of Chu spaces. We will also show some of the laws that hold for Chu spaces and their algebra, and show that Chu spaces form a model for linear logic.

6.1 Definitions and interpretation

We first define the various constants used in the rest of the chapter. We have already defined the constant $\perp$ in chapter 4 as the Chu space $(\{a, b\}, \{\{b\}\})$. The other constants we need are $\top = (\{a\}, \{\{\}, \{a\}\})$, $\mathbf{0} = (\{\}, \{\emptyset\})$ and $\mathbf{1} = (\{a\}, \{\})$. Note that $\perp \cong \top^\perp$ and $\mathbf{1} \cong \mathbf{0}^\perp$. $\top$ is the elementary process $\surd$ which does one event, while $\mathbf{0}$ is the process which does nothing.

The categorical operations that we define here are product, coproduct, and tensor. We also define several other operations on Chu spaces derived from process algebra, namely choice, sequential composition and partial synchronous product, making this algebra of Chu spaces an expressive specification language. We always start with T_0 and extensional Chu spaces, though sometimes the constructions given do not yield T_0

extensional spaces. This is resolved by standardization, that is, identifying any equal rows or columns. In the following paragraphs, we will identify processes with their Chu space representations. These operations have been presented for two operands, and their generalization to infinitely many operands is straightforward.

Coproduct. Let $\mathcal{P} = (A, X)$ and $\mathcal{Q} = (B, Y)$ be Chu spaces with disjoint event sets A and B. Define the Chu space $(C, Z) \triangleq (A \cup B, \{x \cup y \mid x \in X, y \in Y\})$. Then (C, Z) is the categorical coproduct of $\mathcal{P}$ and $\mathcal{Q}$, denoted $\mathcal{P} + \mathcal{Q}$. It has injection maps $(f_1, g_1) : \mathcal{P} \to \mathcal{P} + \mathcal{Q}, f_1(a) = a, g_1(z) = z \cap A$ and $(f_2, g_2) : \mathcal{Q} \to \mathcal{P} + \mathcal{Q}$ defined by $f_2(b) = b, g_2(z) = z \cap B$. If (C', Z') is another Chu space with maps $(f_1', g_1') : \mathcal{P} \to (C', Z')$ and $(f_2', g_2') : \mathcal{Q} \to (C', Z')$, then the unique map $(h, k) : \mathcal{P} + \mathcal{Q} \to (C', Z')$ given by $h(a) = f_1'(a), h(b) = f_2'(b)$ and $k(z') = g_1'(z') \cup g_2'(z')$ shows the universality of the coproduct. The unit of coproduct is the constant $\mathbf{0}$, as $\mathcal{P} \cong \mathcal{P} + \mathbf{0}$.

If Γ and Δ were the two logical formulas representing $\mathcal{P}$and $\mathcal{Q}$, then the logical formula representing $\mathcal{P} + \mathcal{Q}$ is $\Gamma \wedge \Delta$, as the variables of $\Gamma \wedge \Delta$ are the union of the variable sets of Γ and Δ, and its satisfying assignments correspond to a satisfying assignment of Γ and a satisfying assignment of Δ, which are exactly the states of $\mathcal{P} + \mathcal{Q}$. As $\Gamma \wedge \Delta \to \Gamma$ and $\Gamma \wedge \Delta \to \Delta$ are Boolean tautologies, we get the two injection maps required for a coproduct (from the logical representation of Chu maps, chapter 4). We can now use the gate representation of Chu spaces to get the gate for their coproduct. If $|A| = 3$ and $|B| = 2$, then $\mathcal{P}$ and $\mathcal{Q}$ are represented by gates with 3 and 2 inputs respectively. The sum gate given in figure 6.1 is then the coproduct gate.

Operationally, notice that for the output of the sum gate to be 1, the outputs of both the component gates must be one. This means that any inputs of $\mathcal{P}$ can be toggled independently of the inputs of $\mathcal{Q}$, and the output of the sum will remain 1 if $\mathcal{P}$ continues to be in an acceptable state. Thus this gate represents the non-communicating parallel execution of the processes represented by $\mathcal{P}$ and $\mathcal{Q}$, since events in each may happen independently of the other.

Product. This is the dual to coproduct, defined as follows. If $\mathcal{P} = (A, X)$ and $\mathcal{Q} = (B, Y)$, then their product is $\mathcal{P} \times \mathcal{Q} \triangleq (A \times B, \{x \times B \mid x \in X\} \cup \{A \times y \mid y \in Y\})$. This is the categorical product, and the projection maps are $(f, g) : \mathcal{P} \times \mathcal{Q} \to \mathcal{P}$,

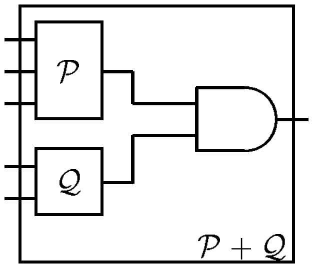


Figure 6.1: The sum of two gates.

$f((a,b)) = a$ and $g(x) = x \times B$ and a similarly defined right projection. For any other Chu space (C, Z) with maps (f_1', g_1') to $\mathcal{P}$ and (f_2', g_2') to $\mathcal{Q}$, we get a unique map (h, k) to $\mathcal{P} \times \mathcal{Q}$ via $h(c) = (f_1'(c), f_2'(c))$ and $k(x \times B) = g_1'(x), k(A \times y) = g_2'(y)$, showing that this is the categorical product.

Now $(\mathcal{P} \times \mathcal{Q})^\perp \cong \mathcal{P}^\perp + \mathcal{Q}^\perp$ via the map $(h, k) : (\mathcal{P} \times \mathcal{Q})^\perp \to \mathcal{P}^\perp + \mathcal{Q}^\perp$, where $h(x \times B) = x, h(A \times y) = y$ and $k(a \cup b) = (a, b)$. This is an isomorphism as h and k are bijective, and is a Chu map as $h(x \times B) \in a \cup b \Leftrightarrow x \in a \Leftrightarrow a \in x \Leftrightarrow (a, b) \in x \times B \Leftrightarrow x \times B \in k(a \cup b)$, and similarly for $h(A \times y)$. Thus the identity for $\times$ is $\mathbf{1}$: $\mathcal{P} \times \mathbf{1} \cong \mathcal{P}$.

The circuit for $\mathcal{P} \times \mathcal{Q}$ can now be derived from the definition. Each state comes from the first or the second set, so we construct circuits for them and form their disjunction. In a state from the first set, the input (a, b) depends only on a, so the inputs corresponding to (a, b) must be the same for any $b \in B$. This is ensured by the equivalence gates ($\equiv$), which output one iff all their inputs are the same. Finally the gate $\mathcal{P}$ judges whether the a's form a state. The circuit for the other set is similar.

We do not know a good operational interpretation for the product of two Chu spaces. However, notice that once a process is in a state from the first set of states, i.e. it is in a state of the form $x \times B, x \in X$, then it can only go to another state of this form according to the transition relation defined by subsets. So from the product construction we can derive another operation, which allows a process to do either one process or another, but not both.

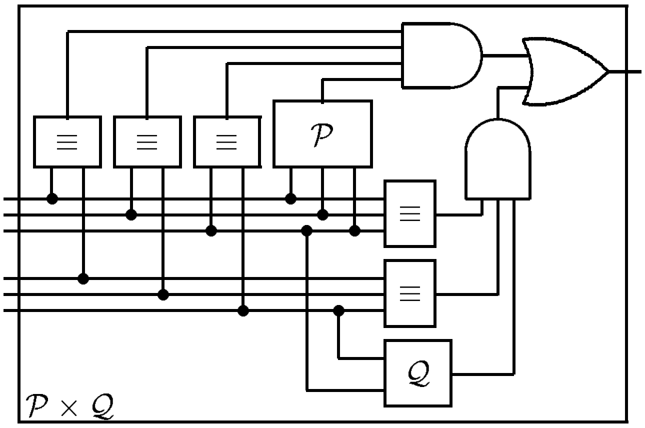


Figure 6.2: The product of two gates.

Choice. Let $\mathcal{P}$ and $\mathcal{Q}$ be two Chu spaces, each containing the event 0 which is not an element of any state[1]. Now consider their product $\mathcal{P} \times \mathcal{Q}$, and delete any events not of the form $(a, 0)$ or $(0, b)$ from the set of events, and from each state. Thus the set of events looks like the disjoint union of the event sets of $\mathcal{P}$ and $\mathcal{Q}$, and each state contains only events of the form $(a, 0)$ or only events of the form $(0, b)$. So the resulting process can do either the first process or the second, but not both.

Alternatively, we can define the choice of $\mathcal{P} = (A, X)$ and $\mathcal{Q} = (B, Y)$ as $\mathcal{P} \sqcup \mathcal{Q} = (A \cup B, X \cup Y)$, where A and B are disjoint. Note that the disjointness of A and B means that $X \cap Y = \{\}$ or $\{\{\}\}$. Thus once we do some event in $\mathcal{P}$, no events of $\mathcal{Q}$ can be executed, and vice versa. The circuit in figure 6.3 implements this definition, by forcing all the inputs of $\mathcal{Q}$ to be at 0 if any input of $\mathcal{P}$ is 1, and vice versa.

The choice operation is functorial, i.e. if there is a morphism from $\mathcal{P}$ to $\mathcal{P}$', there is a morphism from $\mathcal{P} \sqcup \mathcal{Q}$ to $\mathcal{P}' \sqcup \mathcal{Q}$ — it is the same morphism on $\mathcal{P}$ and identity on $\mathcal{Q}$. We will show that it is the categorical sum for a different category on the Chu objects in chapter 7. If $X \cap Y = \{\}$, then choice is a self dual operation, i.e. $\mathcal{P} \sqcup \mathcal{Q} \cong (\mathcal{P}^\perp \sqcup \mathcal{Q}^\perp)^\perp$. The identity for choice is the Chu space $\emptyset = (\{\}, \{\})$.

[1]The 0 event can be added by forming the sum $\mathcal{P} + \mathbf{1} = (A \cup \{0\}, X)$. This adds the 0 event iff it is not there, otherwise the two 0 events are identified by the T_0 property.

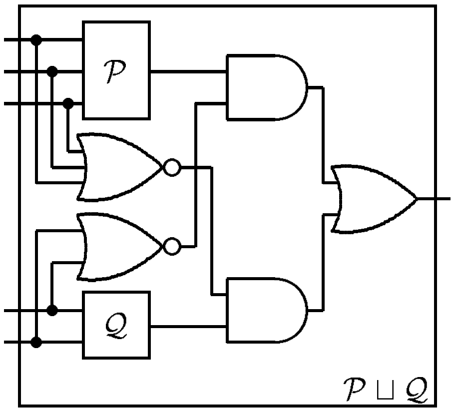


Figure 6.3: Choice of two Chu spaces.

Tensor. In chapter 4 we presented the internal homfunctor. We now give an explicit construction for its left adjoint, the tensor product. If $\mathcal{P} = (A, X)$ and $\mathcal{Q} = (B, Y)$ are two Chu spaces, then their tensor product $\mathcal{P} \otimes \mathcal{Q} \triangleq (A \times B, Z)$ where $z \in Z$ iff $z \subseteq A \times B$ and $\forall b \in B.[z_b = \{a \mid (a, b) \in z\} \in X]$ and $\forall a \in A.[z_a = \{b \mid (a, b) \in z\} \in Y]$. Thus any state in $\mathcal{P} \otimes \mathcal{Q}$ may be viewed as a bilinear map[2] $z : A \times B \to 2$ such that $z(\_, b) : A \to 2$ and $z(a, \_) : B \to 2$ are maps(states) in X and Y respectively.

Now we can prove the required adjunction:

Lemma 6.1 *$(\mathcal{P} \otimes \mathcal{Q}) \multimap \mathcal{R} \cong \mathcal{P} \multimap (\mathcal{Q} \multimap \mathcal{R})$, and this isomorphism is natural in $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{R}$.*

Proof: We first prove that $\mathcal{P} \otimes \mathcal{Q} \cong (\mathcal{P} \multimap \mathcal{Q}^{\perp})^{\perp}$, and will then show that this identity implies the isomorphism above.

If $\mathcal{P} = (A, X)$ and $\mathcal{Q} = (B, Y)$, then $(\mathcal{P} \multimap \mathcal{Q}^{\perp})^{\perp} \cong (A \times B, Z)$, where Z is the set of maps from (A, X) to (Y, B). Each such map is a pair of maps $(f, g) : (A, X) \to (Y, B)$, so $f : A \to Y$ and $g : B \to X$. Now the adjointness condition allows the identification of these maps into one bilinear map $z : (A \times B) \to 2$, and vice versa, we can derive

[2] A bilinear map is a map on two arguments, such that it is linear if one of them is held fixed. Here, linear maps are those which are members of the state set, with the states considered as maps to 2

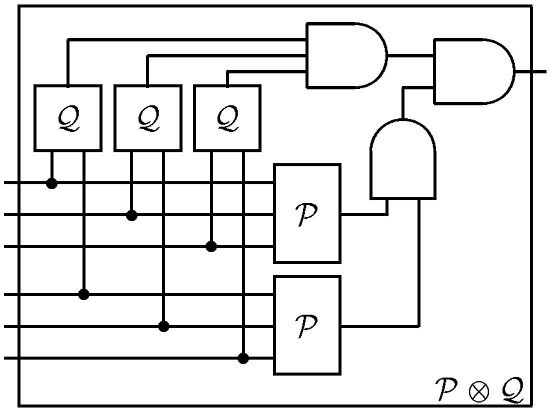


Figure 6.4: Tensor product of Chu spaces.

such a pair of maps from a bilinear map. So we have $\mathcal{P} \otimes \mathcal{Q} \cong (\mathcal{P} \multimap \mathcal{Q}^\perp)^\perp$.

Now, using the identities of section 4.3, we have $(\mathcal{P} \otimes \mathcal{Q}) \multimap \mathcal{R} \cong (\mathcal{P} \multimap \mathcal{Q}^\perp)^\perp \multimap \mathcal{R}$ $\cong \mathcal{R}^\perp \multimap (\mathcal{P} \multimap \mathcal{Q}^\perp) \cong \mathcal{P} \multimap (\mathcal{R}^\perp \multimap \mathcal{Q}^\perp) \cong \mathcal{P} \multimap (\mathcal{Q} \multimap \mathcal{R})$.

Naturality follows by a simple diagram chasing and composition. For example, if we have a map $(h, h') : \mathcal{P}' \to \mathcal{P}$, then we can substitute $h(a)$ for a in the isomorphism $(\mathcal{P} \otimes \mathcal{Q}) \multimap \mathcal{R} \cong \mathcal{P} \multimap (\mathcal{Q} \multimap \mathcal{R})$, and compose with h' for the adjoint map to get the isomorphism $(\mathcal{P}' \otimes \mathcal{Q}) \multimap \mathcal{R} \cong \mathcal{P}' \multimap (\mathcal{Q} \multimap \mathcal{R})$, showing naturality in $\mathcal{P}$. ∎

$\mathcal{P} \otimes \top \cong \mathcal{P}$ from the above lemma, so $\top$ is the identity for $\otimes$.

Since each state in the tensor product is a bilinear function, we can design a circuit which for each $a \in A$ checks that the set of inputs $(a, \_)$ form a state of Y, and vice versa for the b's. The circuit in figure 6.4 implements this idea, giving a circuit for the tensor product of $\mathcal{P}$ and $\mathcal{Q}$.

Both the symmetry of this picture and the definition show that the tensor product is commutative. It is also associative, as can be established by simplifying the above circuit after plugging in the circuit for $\mathcal{P} \otimes \mathcal{Q}$ in place of each $\mathcal{P}$ box.

Operationally, tensor represents the interaction of two processes, also known as orthocurrence [Pra86, CCMP91]. It represents one process flowing through another. For example, if process $\mathcal{P}$ is three trains running sequentially, and process $\mathcal{Q}$ represents two stations on the track, then there are $3 \times 2 = 6$ events, corresponding to each train arriving at each station. For each train, the stations must arrive in the same order, and for each station the trains must arrive in the same order too. Thus each state of the system is a bilinear combination of the states of the system, which is enforced by the above circuit. Each copy of $\mathcal{P}$ may be understood as a guard and each $\mathcal{Q}$ as a station master, who verify that the conditions are locally met.

An example of a system which is best represented by interaction is a communication channel. The channel is a process with several states (buffers) for each message, while the input process is a stream of messages. Their interaction, or tensor product then describes the behavior of the channel with messages, each event being the placing of a message in a buffer. Further restrictions can then be placed on this channel to get the desired specification of the channel behavior.

There are some other connectives derived from process algebra that we would like for Chu spaces. We present here sequential composition, restriction and partial synchronous product, others may also be defined.

Sequential composition. $\mathcal{P};\mathcal{Q}$ is the process which performs a process $\mathcal{Q}$ after finishing a process $\mathcal{P}$. So to define it we need a notion of what it means to finish a process—we will assume that if a process is in a maximal state (i.e. there is no state which is its superset) then the process is complete. Note that this means that if a process goes on forever, i.e. has no maximal states, then $\mathcal{Q}$ will never be done.

If $\mathcal{P} = (A, X)$ and $\mathcal{Q} = (B, Y)$, and $A \cap B = \emptyset$, then we define $\mathcal{P};\mathcal{Q} = (A \cup B, Z)$ where $Z = \{x \in X \mid x \text{ is not maximal}\} \cup \{x \cup y \mid x \text{ maximal in } X, y \in Y\}$. So the states of $\mathcal{P}$;$\mathcal{Q}$ are states of $\mathcal{P}$ in the beginning, and after $\mathcal{P}$ is done, $\mathcal{Q}$ is started. Since the process reached a maximal state of $\mathcal{P}$ before starting $\mathcal{Q}$, no events of $\mathcal{P}$ can be executed once $\mathcal{Q}$ has begun.

$\mathbf{0}$ is the two sided identity of sequential composition, as $\mathcal{P};\mathbf{0} \cong \mathbf{0};\mathcal{P} \cong \mathcal{P}$. This operation is functorial in its second argument, that is given a map f from $\mathcal{Q}$ to $\mathcal{Q}$' we can uniformly find a map from $\mathcal{P}$;$\mathcal{Q}$ to $\mathcal{P}$;$\mathcal{Q}$'—this map is the identity of $\mathcal{P}$, and behaves like f on $\mathcal{Q}$. It is not functorial in the first argument, as maximal states need not be mapped to maximal states by maps, so the map induced by a map from $\mathcal{P}$ to $\mathcal{P}'$ would map some states in $\mathcal{P}';\mathcal{Q}$ to sets of events that are not states in $\mathcal{P};\mathcal{Q}$.

The circuit representation of $\mathcal{P}$;$\mathcal{Q}$ in terms of the circuits for $\mathcal{P}$ and $\mathcal{Q}$ is rather

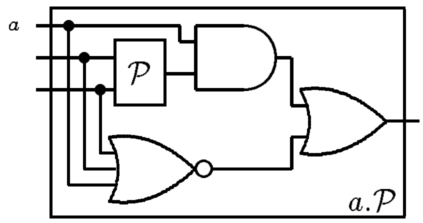


Figure 6.5: Prefixing a Chu space.

complex, since one has to determine the structure of $\mathcal{P}$ in order to see whether a state is maximal. This can be done via G. D. Plotkin's result on circuit definability [Plo94c] :- A function on Chu spaces is circuit definable iff the number of input lines of the result is an n-ary function of the number of input lines of the n arguments. The circuit is constructed by first determining the arguments via testing, and then for those arguments, giving the circuit for the answer. Suppose the function had one argument. For an argument with n inputs, there are 2^{2^n} possible values of the argument, so for each possible value we test for equivalence with the input box for all the possible 2^n input vectors in parallel, by making 2^n copies of the argument and test value. Since only one of these 2^{2^n} equivalence tests will give the result true, we can construct separate circuits for each possible value of the argument, and 'and' its output with the equivalence test result for that value. The inputs to these 2^{2^n} circuits come from outside, and their outputs are 'or'ed together to get the output.

A special case of sequential composition is prefixing, as used in CCS. Here the process $\mathcal{P}$ is a copy of the process $\top$, so it just does a single event, and then continues doing $\mathcal{Q}$. By restricting $\mathcal{P}$ to this simple process, the complexity of identifying maximal states is avoided by CCS, at the expense of making the sequential composition less powerful. For this restricted operation the circuit representation is simple, and is given in figure6.5. We will denote prefixing the event a to the Chu space $\mathcal{P}$ by $a.\mathcal{P}$.

Action Refinement. Using the idea of maximal states we can define action refinement by general Chu spaces. Recall that in chapter 5 we defined the refinement of an event by a Chu space (A, X) with $\emptyset, A \in X$. We can generalize this definition to refinement by any Chu space:

Given a Chu space (B, Y) and an event $b \in B$, we define the refinement of b by (C, Z) as the process (A, X), where $A = B - \{b\} \cup C$, and X is defined as follows :

$$\begin{aligned} X = \quad & \{x \mid b \notin x \in Y, x \cup \{b\} \notin Y\} \\ & \cup \{x - \{b\} \cup z \mid b \in x \in Y, z \text{ maximal in } Z\} \\ & \cup \{x - \{b\} \cup z \mid b \in x \in Y, z \in Z, x - \{b\} \in Y\} \end{aligned}$$

In the first clause we did not want any state x from which some b could be done, since x will come from the third clause, if $\emptyset \in Z$. If $\emptyset \notin Z$, and b could be started in x, after refinement we want the start states of (C, Z) to be merged with copies of x. This is ensured by the above definition.

In general, we will want to refine an action in a labeled Chu space. This is done by refining each event which has a particular label by a copy of the process (C, Z); these copies must be distinguished since they refine different events. The refinements can be done successively.

Restriction. The restriction operation allows us to remove certain events from a Chu space. Given a Chu space $\mathcal{P} = (A, X)$ and a set of events $B \subseteq A$, we define $\mathcal{P} \backslash B \stackrel{\triangle}{=} (A - B, \{x \in X \mid x \cap B = \emptyset\})$. Thus the events in B are no longer possible, and in fact any state of $\mathcal{P}$ that was reachable by doing some events of B is not a state any more. In the circuit representation, this means removing the inputs corresponding to the events in B from the input set, and forcing them to 0 by grounding.

Partial Synchronous Product. The operation of partial synchronous product for event structures was defined by Winskel [Win86] as the parallel composition of processes $\mathcal{P}$ and $\mathcal{Q}$ with some events of $\mathcal{P}$ synchronizing with some events of $\mathcal{Q}$. This definition can be extended to Chu spaces. The set of events of $\mathcal{P} \parallel \mathcal{Q}$, where $\mathcal{P}$ and $\mathcal{Q}$ have disjoint event sets, is $A \cup B \cup (A \times B)$. Here A and B represent the unsynchronized events, and the pairs in $A \times B$ represent the synchronizations. For the states, naturally we do not want both the pairs (a, b) and (a, c) in any state, since that would mean that a occurred in synchronization with b and then again with c. Similarly an unsynchronized event should not occur along with a synchronization of the same event. Thus the state is formed by taking a state in X and a state in Y and forming all possible combinations by pairing some events of X with some events of Y. Each pair of states can yield many new states—for example from the pair $\{a\}$ and $\{b, c\}$ we can get 3 states, $\{a, b, c\}, \{(a, b), c\}$ and $\{b, (a, c)\}$.

As an example, let $\mathcal{P} = (\{a\}, \{\phi, \{a\}\})$ and $\mathcal{Q} = (\{b, c\}, \{\phi, \{b\}, \{b, c\}\})$. Then the events in $\mathcal{P} \parallel \mathcal{Q}$ are $\{a, b, c, (a, b), (a, c)\}$. The possible states are formed from pairs of states as follows :

State of $\mathcal{P}$	State of $\mathcal{Q}$	States of $\mathcal{P} \parallel \mathcal{Q}$
ϕ	ϕ	ϕ
ϕ	$\{b\}$	$\{b\}$
ϕ	$\{b, c\}$	$\{b, c\}$
$\{a\}$	ϕ	$\{a\}$
$\{a\}$	$\{b\}$	$\{a, b\}, \{(a, b)\}$
$\{a\}$	$\{b, c\}$	$\{a, b, c\}, \{(a, b), c\}, \{b, (a, c)\}$

The identity for partial synchronous product is $\mathbf{0}$. Partial synchronous product is not functorial, for example there is a unique map from $\mathcal{P} = (\{a_1, a_2\}, \{\{\}, a_1, a_1a_2\})$ to $\mathcal{Q} = (\{b\}, \{\{\}, b\})$, but there is no natural map from $\mathcal{P} \parallel \mathcal{R}$ to $\mathcal{Q} \parallel \mathcal{R}$, where $\mathcal{R} = (\{c\}, \{\{\}, c\})$. However it is functorial if we restrict ourselves to injective maps. In the rest of this chapter, whenever we talk about partial synchronous product as a functor, we will refer to the category of Chu spaces and injective maps.

The circuit for $\mathcal{P} \parallel \mathcal{Q}$ can be drawn as in figure 6.6. We use the symbol # to denote a conflict box, its output is 1 iff at most one of the inputs is 1. $\mathcal{P}$ and $\mathcal{Q}$ each have two inputs, $\{a, b\}$ and $\{c, d\}$ respectively.

Partial synchronous product is used to define communicating parallel composition as in CCS or CSP. All the synchronizations that are not allowed by the language are restricted away.

Labels. In the above discussion, we always had unlabeled Chu spaces. The extension to labeled Chu spaces is as follows.

If λ_1 and λ_2 are labeling functions for $\mathcal{P}$ and $\mathcal{Q}$ respectively, then the labeling function for $\mathcal{P} + \mathcal{Q}$ is given by λ, where $\lambda \restriction A = \lambda_1$ and $\lambda \restriction B = \lambda_2$. This is also true

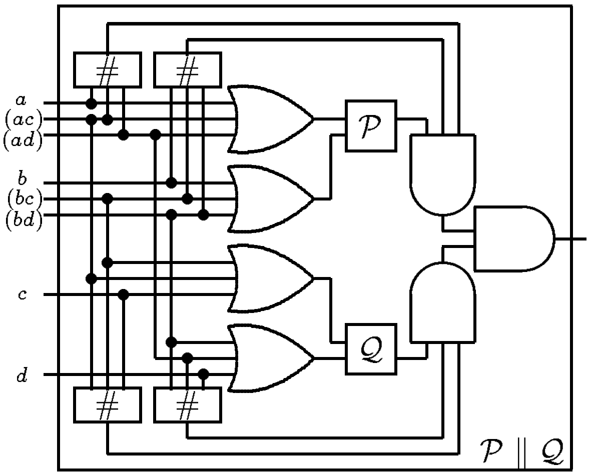


Figure 6.6: Partial Synchronous Product of $\mathcal{P}$ and $\mathcal{Q}$.

for the operations $\mathcal{P} \sqcup \mathcal{Q}$ and $\mathcal{P};\mathcal{Q}$. In $\mathcal{P} \otimes \mathcal{Q}$, event (a, b) is labeled by a pair such that $\lambda(a, b) = (\lambda_1(a), \lambda_2(b))$. The restriction operation can restrict labels instead of events, in which case $\mathcal{P}\backslash\mathcal{A}$, $\mathcal{A} \subseteq Act$ will be implemented by restricting $\mathcal{P}$ by $B = \{a \in A \mid \lambda(a) \in \mathcal{A}\}$.

Labels for events in partial synchronous product depend on the desired application. For example, in CCS, we would restrict away all events that synchronize events with non-complementary labels, and label the other pairs by τ. The singleton events are labeled as before.

We can define the operation of relabeling on Chu spaces, denoted $\mathcal{P}[f]$, where $f : Act \to Act$ is a relabeling function. If $\mathcal{P} = (A, X, \lambda)$, then define $\mathcal{P}[f] = (A, X, f \circ \lambda)$.

Recursion and the solution of recursive domain equations. We would like to get Chu spaces for processes which are defined recursively. For example, if $\surd$ is a process which does a single event and halts, then the iteration operator $\mathcal{P}^*$ can be defined as $\mathcal{P}^* = \surd \sqcup \mathcal{P};\mathcal{P}^*$, or using fixed point notation, as $\mathcal{P}^* = \mu X.\surd \sqcup \mathcal{P}; X$. We would like to know when such an equation has a solution.

We will use the techniques developed in [PS82] to obtain the solutions. The category **Chu** has an initial object, **0**, with a unique map to every other object, namely the empty map. It also has colimits of ω-chains.

In [Bar91], Barr proved that if an autonomous category is bicomplete, then the $*$-autonomous category generated from it via Chu's construction is also bicomplete. **Set** is bicomplete, thus **Chu** is also bicomplete, and from this it immediately follows that it has an initial object and colimits of ω-chains, that is, it is an ω-complete pointed category as defined in [PS82]. Here we give an elementary proof of this fact, mainly in order to establish notation.

An ω-chain is a functor from the category ω, whose objects are $0, 1, 2, \ldots$ and which has a morphism r_{mn} from m to n for every $m \leq n$, and $r_{mn} \circ r_{lm} = r_{ln}$. Thus an ω-chain is a collection of objects $A_0, A_1, \ldots$ and morphisms f_{mn} if $m < n$. (f_{mm} is the identity map, and we denote by f_m the map from A_m to A_{m+1}). A colimit of an ω-chain is like a least upper bound for posets, namely an element A, with maps $h_i : A_i \to A$ for each i such that $h_m = h_n \circ f_{mn}$ for all m, n. It is universal, so if there is any other object C and a set of maps h'_i satisfying these properties then there is a unique map $h' : A \to C$, such that $h'_i = h' \circ h_i$.

Proposition 6.1 **Chu** *has colimits of all* ω*-chains.*

Proof: Consider the ω-chain

$$P_0 \xrightarrow{f_0} P_1 \xrightarrow{f_1} P_2 \xrightarrow{f_2} P_3 \ldots$$

where $P_i = (A_i, X_i)$ etc, and the maps are pairs $f_i = (f_i, g_i)$. Now we get the other maps $f_{mn} = f_{n-1} \circ \ldots f_m$.

Define an equivalence relation $\approx$ on $\bigcup_i A_i$ by $a \approx b$ iff there are maps f_{mn}, f_{kn} such that $f_{mn}(a) = f_{kn}(b)$. Let A be the set of equivalence classes of this relation, with the equivalence class of a denoted by $[a]$. Given any $x \subseteq A$, define $x_i = \{a \in A_i \mid [a] \in x\}$. Let X be the set $\{x \subseteq A \mid \forall i. x_i \in X_i\}$. Then we claim that (A, X) is the colimit of this chain. The set of maps $(h_i, k_i) : (A_i, X_i) \to (A, X)$ is obvious, $h_i(a) = [a]$ and $k_i(x) = x_i$, and these form an adjoint pair.

Now if we have any (A', X') and a set of maps (h'_i, k'_i) which is also a candidate for the colimit, then we can define the unique map $(h', k') : (A, X) \to (A', X')$ by

$h'([a]) = h'_i(a)$, and k' as its adjoint. h' is well defined since if $[a] = [b]$ then by commutativity requirements, $h'_i(a) = h'_j(a)$. It makes $h'_i = h' \circ h_i$ true for all i, and is unique since if we had any other $h'' : (A, X) \to (A', X')$ different from h', then it would differ at some $[a]$, but this is not possible since it must be equal to $h_i(a)$. ∎

A functor F is called ω-continuous if it preserves colimits of all ω-chains. That is, if P is the colimit of an ω-chain $\mathbf{P}$ then $F(P)$ is the colimit of the ω-chain $F(\mathbf{P})$. Compositions of ω-continuous functors are continuous. Now if we have an ω-continuous functor, its least fixed point is given as the colimit of the ω-chain

$$\mathbf{0} \xrightarrow{f_0} F(\mathbf{0}) \xrightarrow{f_1} F^2(\mathbf{0}) \xrightarrow{f_2} F^3(\mathbf{0}) \ldots$$

Note that there is an arrow from $\mathbf{0}$ to $F(\mathbf{0})$ as it is the initial object. The other arrows are images of this arrow under F. If $\mathcal{Q}$ is the colimit, then so is $F(\mathcal{Q})$, thus the two are isomorphic, showing that $\mathcal{Q}$ is a solution of the equation $X = F(X)$. Now all we have to do is show that the various functors $F(X) = \mathcal{P} + X, \mathcal{P} \sqcup X, \mathcal{P}; X, \mathcal{P} \otimes X, X \backslash L$ are ω-continuous to be able to get solutions for recursive domain equations written over these operations.

$\mathcal{P} + X$ and $\mathcal{P} \otimes X$ preserve all colimits, since they have right adjoints [Mac71, p.115]. So they are automatically ω-continuous.

Proposition 6.2 *The functor $F(X) = \mathcal{P} \sqcup X$ is ω-continuous.*

Proof: Consider the ω-chain $\mathbf{Q}$

$$\mathcal{Q}_0 \xrightarrow{f_0} \mathcal{Q}_1 \xrightarrow{f_1} \mathcal{Q}_2 \xrightarrow{f_2} \mathcal{Q}_3 \ldots$$

where $Q_i = (B_i, Y_i)$. Let its colimit be $\mathcal{Q} = (B, Y)$. Let $\mathcal{P} = (A, X)$. Now $f_i : \mathcal{Q}_i \to \mathcal{Q}_{i+1}$, so the map $f'_i : \mathcal{P} \sqcup \mathcal{Q}_i \to \mathcal{P} \sqcup \mathcal{Q}_{i+1}$ is derived from f_i by letting f'_i be f on B_i, and identity on A. Its adjoint map is similarly the adjoint of f_i on Y_i and identity on X. Thus in forming the colimit of the ω-chain $F(\mathbf{Q})$, we see that each element of A forms an equivalence class under the $\approx$ relation defined above, while the other classes are as before. Also, for each $x \in X$, $\{[a] \mid a \in x\}$ is a state in the colimit, and the other states are those in Y. Thus the colimit of $F(\mathbf{Q})$ is isomorphic to $\mathcal{P} \sqcup \mathcal{Q}$, showing that F is ω-continuous. ∎

The proofs that the operations $\mathcal{P}; X$ and $\mathcal{P} \times X$ are ω-continuous are very similar, and are omitted. Note that $+, \otimes, \sqcup, \|$ and $\times$ are symmetric, so the X can be either argument. $X; P$ is not a functor, as remarked before. For restriction, $F(X) = X \backslash L$, it is necessary to specify what this means as a functor, since if we just choose L to be some set of events then isomorphic copies of X will be mapped to different images. We will understand this to mean restriction on labels, that is we restrict X on the set of all events whose labels are in L. Then we will consider label preserving morphisms only, making this an ω-continuous operation. If we are willing to restrict ourselves to the category of Chu spaces with injective Chu maps only, then $\mathcal{P} \parallel X$ is also a functor, and is $\omega - continuous$.

The functor $F(X) = \mathcal{P} \multimap X$, where $\mathcal{P} = (A, X)$ is finite, is also ω-continuous. On morphisms, this functor just composes them with elements of $\mathcal{P} \multimap X$. Now in the ω-chain $F(\mathbf{Q})$, two elements h, h' are $\approx$-equivalent iff they are $\approx$-equivalent on all elements of A, i.e. $\forall a \in A, h(a) \approx h'(a)$. This shows that $\mathcal{P} \multimap \mathcal{Q}$ is the desired colimit.

Thus we can solve recursive domain equations for a variety of operators. Now we can define more operations by using recursion. Here we mention only one — iteration, defined to be the least solution of $F(X) = \surd \sqcup \mathcal{P}; X$. The least solution to this equation is denoted $\mathcal{P}^*$, and is the process that does $\mathcal{P}$ zero or more times, terminating with a $\surd$ event. For some related work on solving recursive domain equations for Chu spaces, see [Plo94b].

6.2 Identities and Equivalences

The above operations satisfy a number of identities, which are useful in understanding the intuitive meaning of the combinators. We will prove some of these in this section.

Proposition 6.3 (The commutative and associative laws) *The operations coproduct, product, choice, tensor product and partial synchronous product are all commutative and associative up to isomorphism. The isomorphisms are natural in each* *component*[3].

Proof: The proofs of these laws follow immediately from the definitions of the operations. As an example, we will prove the commutativity of the tensor product.

Let $\mathcal{P} = (A, X)$ and $\mathcal{Q} = (B, Y)$ be two Chu spaces. We will identify the elements of X and Y with their characteristic functions. Now the isomorphism $\mathcal{P} \otimes \mathcal{Q} \cong \mathcal{Q} \otimes \mathcal{P}$ is given by the Chu map $(\gamma_{\mathcal{P}\mathcal{Q}}, \eta_{\mathcal{P}\mathcal{Q}})$, $\gamma_{\mathcal{P}\mathcal{Q}}(\langle a, b\rangle) = \langle b, a\rangle$ and $\eta_{\mathcal{P}\mathcal{Q}}(y) = x$, where $x(\langle a, b\rangle) = y(\langle b, a\rangle)$. Thus $(\gamma_{\mathcal{P}\mathcal{Q}}, \eta_{\mathcal{P}\mathcal{Q}})$ is an isomorphism between $\mathcal{P} \otimes \mathcal{Q}$ and $\mathcal{Q} \otimes \mathcal{P}$.

Now for it to be natural in $\mathcal{P}$, given a map $(f, g) : \mathcal{P} \to \mathcal{P}'$, we have to show that $(Id_{\mathcal{Q}} \otimes (f, g)) \circ (\gamma_{\mathcal{P}\mathcal{Q}}, \eta_{\mathcal{P}\mathcal{Q}}) = (\gamma_{\mathcal{P}'\mathcal{Q}}, \eta_{\mathcal{P}'\mathcal{Q}}) \circ ((f, g) \otimes Id_{\mathcal{Q}})$. For the first component, we have

$$\begin{aligned}(Id_{\mathcal{Q}} \otimes f) \circ \gamma_{\mathcal{P}\mathcal{Q}}(\langle a, b\rangle) &= (Id_{\mathcal{Q}} \otimes f)(\langle b, a\rangle) \\ &= \langle b, f(a)\rangle \\ &= \gamma_{\mathcal{P}'\mathcal{Q}}(\langle f(a), b\rangle) \\ &= (\gamma_{\mathcal{P}'\mathcal{Q}} \circ (f \otimes Id_{\mathcal{Q}}))(\langle a, b\rangle)\end{aligned}$$

For the adjoint map, given a bilinear map $h : B \times A' \to 2$, we have $(\eta_{\mathcal{P}\mathcal{Q}} \circ (Id_{\mathcal{Q}} \otimes g))(h) = h'$, where $h'(\langle a, b\rangle) = h(\langle b, f(a)\rangle)$. This is also what we get as $((g \otimes Id_{\mathcal{Q}}) \circ \eta_{\mathcal{P}'\mathcal{Q}})(h)$, so the isomorphism is natural $\mathcal{P}$. The proof that it is natural in $\mathcal{Q}$ is similar. ∎

Proposition 6.4 (Associativity of seq. comp.) $\mathcal{P}; (\mathcal{Q}; \mathcal{R}) \cong (\mathcal{P}; \mathcal{Q}); \mathcal{R}$, *and this isomorphism is natural in the third argument,* $\mathcal{R}$.

Proof: The isomorphism of the sets of events on the two sides follows from the associativity of disjoint set union. z is a state in $\mathcal{P}; (\mathcal{Q}; \mathcal{R})$ iff it is a non-maximal state in $\mathcal{P}$ or the union of a maximal state in $\mathcal{P}$ with a state in $\mathcal{Q}; \mathcal{R}$. But from the similar condition for a state to be in $\mathcal{Q}; \mathcal{R}$, we get that a state is in $\mathcal{P}; (\mathcal{Q}; \mathcal{R})$ iff it is a non-maximal state in $\mathcal{P}$ or the union of a maximal state in $\mathcal{P}$ with a non-maximal state in $\mathcal{Q}$ or the union of a maximal state from each of $\mathcal{P}$ and $\mathcal{Q}$, and a state in

[3]Naturality for commutativity and associativity of partial synchronous product holds only in the category of Chu spaces and injective Chu maps.

$\mathcal{R}$. This is also what we get from the right hand side, giving the isomorphism. Its naturality follows from set-theoretic arguments. ∎

We have already given the identity elements for each of these operations. These two propositions give the laws which hold for expressions having only one operator on each side. There are some distributivity laws between operators, and some De Morgan laws.

Proposition 6.5

1. $(\mathcal{P} + \mathcal{Q}) \otimes \mathcal{R} \cong (\mathcal{P} \otimes \mathcal{R}) + (\mathcal{Q} \otimes \mathcal{R})$
2. $(\mathcal{P} \times \mathcal{Q})^{\perp} \cong \mathcal{P}^{\perp} + \mathcal{Q}^{\perp}$
3. $(\mathcal{P} \sqcup \mathcal{Q})^{\perp} \cong \mathcal{P}^{\perp} \sqcup \mathcal{Q}^{\perp}$

Proof: (1) This follows from the fact that $\otimes$ has a right adjoint and thus preserves all colimits, in particular coproducts. Let $\mathcal{P} = (A, X), \mathcal{Q} = (B, Y), \mathcal{R} = (C, Z)$. The isomorphism of the sets of events on the two sides follows from the distributivity of the cartesian product of sets over disjoint union.

To show that the state sets are isomorphic, let u be a state in $(\mathcal{P} + \mathcal{Q}) \otimes \mathcal{R}$. Let $u = u_1 \cup u_2$, where u_1 is the set of events whose first component came from A, and u_2 those whose first component came from B. Now for any $c \in C, a \in A$, we have $(u_1)_c = u_c \cap A \in X$, and $(u_1)_a = u_a \in Z$. Thus u_1 is a state of $(\mathcal{P} \otimes \mathcal{R})$, and similarly u_2 is a state of $(\mathcal{Q} \otimes \mathcal{R})$, making u a state of the right side. The reverse inclusion can be proved similarly.

The last two identities were proved in the previous section. ∎

Note that $\otimes$ does not distribute over $\sqcup$, because in $(\mathcal{P} \sqcup \mathcal{Q}) \otimes \mathcal{R}$ a state can have component states from either $\mathcal{P}$ or $\mathcal{Q}$, while in $(\mathcal{P} \otimes \mathcal{R}) \sqcup (\mathcal{Q} \otimes \mathcal{R})$, they can be only from one of $\mathcal{P}$ or $\mathcal{Q}$.

The following laws were inspired by Milner's CCS [Mil89], and we will use these to construct a Chu space model for CCS. They assume that the Chu spaces are labeled.

Proposition 6.6 *Let $\mathcal{P} = (A, X, \lambda)$, $\mathcal{Q} = (B, Y, \mu)$ and $\mathcal{R}$ be Chu spaces and $K, L \subseteq$ Act, and $f, f' :$ Act $\to$ Act. Then the following isomorphisms hold.*

1. $\mathcal{P} \backslash L \cong \mathcal{P}$ *if* $\lambda(A) \cup L = \emptyset$

2. $\mathcal{P}\backslash K\backslash L \cong \mathcal{P}\backslash(K \cup L)$
3. $\mathcal{P}[f]\backslash L \cong \mathcal{P}\backslash f^{-1}(L)[f]$
4. $\mathcal{P}[Id] \cong \mathcal{P}$
5. $\mathcal{P}[f] \cong \mathcal{P}[f']$ *if* $f \restriction \lambda(A) = f' \restriction \lambda(A)$
6. $\mathcal{P}[f][f'] \cong \mathcal{P}[f' \circ f]$
7. $(\mathcal{P} \sqcup \mathcal{Q})\backslash L \cong \mathcal{P}\backslash L \sqcup \mathcal{Q}\backslash L$
8. $(\mathcal{P} \sqcup \mathcal{Q})[f] \cong \mathcal{P}[f] \sqcup \mathcal{Q}[f]$
9. $(\mathcal{P} + \mathcal{Q})\backslash L \cong \mathcal{P}\backslash L + \mathcal{Q}\backslash L$
10. $(\mathcal{P} + \mathcal{Q})[f] \cong \mathcal{P}[f] + \mathcal{Q}[f]$

Proof: Each of these follow from trivial set theoretic arguments, hence the proofs are omitted. ∎

Several other laws for the CCS connectives will be presented in the next chapter.

One law that we would like to have is $\mathcal{P} \sqcup \mathcal{P} \cong \mathcal{P}$, since a choice between $\mathcal{P}$ and $\mathcal{P}$ is just the same as doing $\mathcal{P}$. However, since we require the event sets in choice to be disjoint, we cannot have this. To overcome this deficiency, we propose using *history preserving bisimulations*[Dev88, OGG88, GG89] for equivalence between Chu spaces, rather than isomorphism. We choose history preserving bisimulations since they preserve the causality and choice structure, so are considerably finer than bisimulations, which identify $a; b \sqcup b; a$ with $a + b$, which is contrary to the semantics of our model. However they are sufficiently coarse to permit several equivalences, which we shall show now.

History preserving bisimulations. Let $\mathcal{P} = (A, X, \lambda)$ be a labeled Chu space. As defined for unlabeled spaces in chapter 5, the *history* of a state $x \in X$ is the labeled Chu space $h(x) = (x, \{x' \in X \mid x' \subseteq x\}, \lambda \restriction x)$. Now define a history preserving bisimulation (h.p.b.) between $\mathcal{P}$ and $\mathcal{Q} = (B, Y, \mu)$ as a relation $R \subseteq (X \times Y \times \mathbf{2}^{A \times B})$ such that $(x, y, f) \in R$ iff

1. f is a function from x to y and determines a label preserving Chu isomorphism between $h(x)$ and $h(y)$.

2. $x \subseteq x'$ implies $\exists y' \in Y, f'$ such that $y \subseteq y', (x', y', f') \in R, f' \restriction x = f$ and $\lambda(x' - x) = \mu(y' - y)$.

3. $y \subseteq y'$ implies $\exists x' \in X, f'$ such that $x \subseteq x', (x', y', f') \in R, f' \restriction x = f$ and $\lambda(x' - x) = \mu(y' - y)$.

In [JNW93], Joyal, Nielsen and Winskel define a strong history preserving bisimulation as a history preserving bisimulation with two additional properties—if $(x, y, f) \in R, x' \subseteq x$, implies $\exists y' \in Y, f'$ such that $y' \subseteq y, (x', y', f') \in R, f \restriction x' = f'$, and a similar condition for $y' \subseteq y$. All the results of this section are valid for strong history preserving bisimulations also.

Say $\mathcal{P}$ is equivalent to $\mathcal{Q}$, $\mathcal{P} \sim \mathcal{Q}$, if there exists an h.p.b. between them. We will also write $x \sim x'$ if there is a triple (x, x', f) in the h.p.b. We first show that $\sim$ is a congruence over several operators.

Proposition 6.7 *Let* $\mathcal{P}_1 \sim \mathcal{P}_2$. *Then*

1. $\mathcal{P}_1 + \mathcal{Q} \sim \mathcal{P}_2 + \mathcal{Q}$

2. $\mathcal{P}_1 \sqcup \mathcal{Q} \sim \mathcal{P}_2 \sqcup \mathcal{Q}$

3. $\mathcal{P}_1; \mathcal{Q} \sim \mathcal{P}_2; \mathcal{Q}$

4. $\mathcal{Q}; \mathcal{P}_1 \sim \mathcal{Q}; \mathcal{P}_2$

5. $\mathcal{P}_1 \parallel \mathcal{Q} \sim \mathcal{P}_2 \parallel \mathcal{Q}$

6. $\mathcal{P}_1 \backslash L \sim \mathcal{P}_2 \backslash L, L \subseteq Act$

7. $\mathcal{P}_1[f] \sim \mathcal{P}_2[f], f : Act \rightarrow Act$

Proof: Let $\mathcal{P} = (A, X, \lambda), \mathcal{P}' = (A', X', \lambda'), \mathcal{Q} = (B, Y, \mu)$. Let R be an h.p.b. between $\mathcal{P}$ and $\mathcal{P}$'.

1. From the relation R, construct the relation $R' = \{(x \cup y, x' \cup y, f \cup Id_y) \mid (x, x', f) \in R, y \in Y\}$. Then R' is an h.p.b. between $\mathcal{P}_1 + \mathcal{Q}$ and $\mathcal{P}_2 + \mathcal{Q}$. Given any triple $(x \cup y, x' \cup y, f \cup Id_y) \in R'$, $f \cup Id_y$ is an isomorphism between $h(x \cup y)$ and $h(x' \cup y)$. Also, if a transition from a state $x \cup y$ to $x_1 \cup y_1$ takes place, then that means there was a transition from x to x_1 possible in $\mathcal{P}_1$. So there is a triple $(x_1, x_1', f') \in R$, and thus the triple $(x_1 \cup y_1, x_1' \cup y_1, f' \cup Id_{y_1})$ in R' would satisfy the requirements. Similarly we can verify the third condition, so $\mathcal{P}_1 + \mathcal{Q} \sim \mathcal{P}_2 + \mathcal{Q}$ via R'.

2. Construct the relation $R' = R \cup \{(y, y, Id_y) \mid y \in Y\}$. This can be shown to be a h.p.b. between $\mathcal{P}_1 \sqcup \mathcal{Q}$ and $\mathcal{P}_2 \sqcup \mathcal{Q}$.

3. In the relation R note that maximal states can only be related to maximal states. So from R, construct the relation $R' = \{(x, x', f) \mid (x, x', f) \in R, x \text{ not maximal}\} \cup \{(x \cup y, x' \cup y, f \cup Id_y) \mid (x, x', f) \in R, x \text{ maximal}, y \in Y\}$. Then R' can be shown to be an h.p.b.

4. From the relation R we construct $R' = \{(y, y, Id_y) \mid y \in Y, y \text{ not maximal}\} \cup \{(y \cup x, y \cup x', Id_y \cup f) \mid y \text{ maximal in } Y, (x, x', f) \in R\}$. Then R' is a h.p.b. between $\mathcal{Q}; \mathcal{P}_1$ and $\mathcal{Q}; \mathcal{P}_2$.

5. In the definition of $\mathcal{P}_1 \parallel \mathcal{Q}$, we saw that each state was generated by a state $x \in X$ and a state $y \in Y$. So when building R', for each triple $(x, x', f) \in R$ and each state z in $\mathcal{P}_1 \parallel \mathcal{Q}$ built from x, we introduce a triple $(z, z', g) \in R'$, where $z' = \{f(a) \mid a \in z \cap A\} \cup \{b \mid b \in z \cap B\} \cup \{(f(a), b) \mid (a, b) \in z\}$. The g is clear from this definition, and we can show that R' is a h.p.b.

   The last two cases are rather trivial.

∎

Notice that the operators $\times, \otimes$ were missing from this theorem. The reason is that $\sim$ is not a congruence for these. For $\otimes$, if $\mathcal{P} = x.\mathbf{0}$, then $\mathcal{P} \sqcup \mathcal{P} \sim \mathcal{P}$, but if we tensor it with $\mathcal{Q} = (c \to a) \wedge (c \to b)$, then the two sides are not bisimilar. So when we use the $\sim$ relation for observationally equivalent processes, we will have to leave out these two operators from the algebra.

We can extend the equivalence relation to equivalence between functors, say $F \sim G$ if for all Chu spaces $\mathcal{P}$, $F(\mathcal{P}) \sim G(\mathcal{P})$ *uniformly*. By uniformly we mean that if $(f,g) : \mathcal{P} \to \mathcal{Q}$ is a Chu map, then if the state y in $F(\mathcal{Q})$ is related to y' in $G(\mathcal{Q})$ with the history isomorphism h', then the state $F(g(y))$ in $F(\mathcal{P})$ must be related to the state $G(g(y'))$ in $G(\mathcal{P})$ with the history isomorphism h, with $h \circ F(f) = G(f) \circ h'$. The following proposition can be used to show that equivalence is preserved under recursive definition.

Proposition 6.8 *Let* $\mathbf{P}$ *and* $\mathbf{Q}$ *be two* ω*-chains*

$$\mathbf{P} \;:\; \mathcal{P}_0 \xrightarrow{(f_0,g_0)} \mathcal{P}_1 \xrightarrow{(f_1,g_1)} \mathcal{P}_2 \xrightarrow{(f_2,g_2)} \mathcal{P}_3 \ldots$$

$$\mathbf{Q} \;:\; \mathcal{Q}_0 \xrightarrow{(f_0',g_0')} \mathcal{Q}_1 \xrightarrow{(f_1',g_1')} \mathcal{Q}_2 \xrightarrow{(f_2',g_2')} \mathcal{Q}_3 \ldots$$

and $\forall i. \mathcal{P}_i \sim \mathcal{Q}_i$. *Furthermore let the equivalence be uniform, i.e. if* $x_i \sim y_i$*, then* $g_{i-1}(x_i) \sim g'_{i-1}(y_i)$ *and the* f*'s commute with the history isomorphisms. Then the colimits of the chain are also equivalent,* $\mathcal{P} \sim \mathcal{Q}$.

Proof: Let $\mathcal{P}_i = (A_i, X_i)$, and $\mathcal{Q}_i = (B_i, Y_i)$. Also let $\mathcal{P} = (A, X)$, and $\mathcal{Q} = (B, Y)$.

Let x be a state in the colimit $\mathcal{P}$. Recall that $x_i = \{a \in A_i \mid [a] \in x\}$, the image of x under the maps from $\mathcal{P}_i$ to $\mathcal{P}$. By definition of the colimit, $x_i \in X_i$. Now by the adjointness condition, we can show that $g_{i-1}(x_i) = x_{i-1}$, and the same for the y's.

Now define the relation R by $x \sim y$ iff $\forall i. x_i \sim y_i$. Since the history isomorphisms between x_i and y_i commute with the f's, we can define an isomorphism between $h(x)$ and $h(y)$. We will show that this defines an h.p.b. between $\mathcal{P}$ and $\mathcal{Q}$.

Let $x \sim y$. Suppose there is a transition $x \to x'$ in $\mathcal{P}$. This means that there must be transitions $x_i \to x_i'$ in $\mathcal{P}_i$ for each i. Now since $x_i \sim y_i$, there must be at least one z_i such that we have the same transition $y_i \to z_i$ in $\mathcal{Q}_i$. In fact we will choose the z_i's such that $g'_{i-1}(z_i) = z_{i-1}$, this is always possible. Let $y' = \bigcup_i \{[b] \mid b \in z_i\}$.

Now since $g'_{i-1}(z_i) = z_{i-1}$ for all i, y' is a state in $\mathcal{Q}$. By a set-theoretic argument, we can show that we have a transition $y \to y'$, and since there are isomorphisms between the histories of z_i and x_i', there is an isomorphism between $h(x')$ and $h(y')$.

The proof of the third condition is symmetrical. Thus $\mathcal{P} \sim \mathcal{Q}$. ∎

Corollary 6.1 *Given two functors $F \sim G$, their least fixed points are also equivalent.*

Proof: Since $F \sim G$, the chains $\mathbf{0} \to F(\mathbf{0}) \to F^2(\mathbf{0}) \to \ldots$ and $\mathbf{0} \to G(\mathbf{0}) \to G^2(\mathbf{0}) \to \ldots$ satisfy all the conditions of the above proposition. Thus the colimits, which are the least fixed points of the functors, are bisimilar. ∎

The equivalence $\sim$ yields several intuitively obvious equivalences between Chu spaces.

Proposition 6.9 *The following equivalences hold between Chu spaces.*

1. $\mathcal{P} \sqcup \mathcal{P} \sim \mathcal{P}$
2. $(\mathcal{P} \sqcup \mathcal{Q}) + \mathcal{R} \sim (\mathcal{P} + \mathcal{R}) \sqcup (\mathcal{Q} + \mathcal{R})$
3. $(\mathcal{P} \sqcup \mathcal{Q}); \mathcal{R} \sim (\mathcal{P}; \mathcal{R}) \sqcup (\mathcal{Q}; \mathcal{R})$, *where $\mathcal{P}$ or $\mathcal{Q}$ have a non-empty state each.*
4. $(\mathcal{P} \parallel \mathcal{Q}) + \mathcal{R} \sim (\mathcal{P} + \mathcal{R}) \parallel (\mathcal{Q} + \mathcal{R})$

Proof: Let $\mathcal{P} = (A, X, \lambda), \mathcal{Q} = (B, Y, \mu), \mathcal{R} = (C, Z, \nu)$.

1. Since the event sets of the two copies of $\mathcal{P}$ on the l.h.s. are not disjoint in $\mathcal{P} \sqcup \mathcal{P}$, we will add a subscript to denote where they came from. States will similarly carry a suffix. Now define the relation $R = \{(x_1, x, f : a_1 \mapsto a) \mid x \in X\} \cup \{(x_2, x, f : a_2 \mapsto a) \mid x \in X\}$. It is clear that f determines an isomorphism between $h(x_1)$ and $h(x)$. Also, if a transition $x_1 \to x_1'$ is made on the left, then a similar transition $x \to x'$ can be made on the right, satisfying the second condition of the definition of a h.p.b. The third condition is similarly satisfied.

2. Any state on the left is of the form $x \cup z, x \in X, z \in Z$ or $y \cup z, y \in Y, z \in Z$. Since $\mathcal{R}$ is repeated on the right, events and states of the two copies have to be distinguished by a subscript. So we relate $x \cup z$ to $x \cup z_1$, and $y \cup z$ to $y \cup z_1$. This gives us a history preserving bisimulation between the two sides.

3. The proof is similar to the one above, and follows from the fact that any maximal state of $\mathcal{P} \sqcup \mathcal{Q}$ is a maximal state of $\mathcal{P}$ or a maximal state of $\mathcal{Q}$. The condition is needed for the converse, that is any maximal state of $\mathcal{P}$ or $\mathcal{Q}$ is a maximal state of $\mathcal{P} \sqcup \mathcal{Q}$.

4. Similar to the proof of (2).

∎

The only reason that these equivalences were not isomorphisms was that we needed the two components of choice to be disjoint. History preserving equivalence takes care of that. The intuitive reason for the first equivalence has been mentioned above. The second and fourth equivalences hold as having a choice in parallel with a process is equivalent to making the choice right away, and then executing the pair of processes in parallel. The third equivalence holds as the choice between $\mathcal{P}$ and $\mathcal{Q}$ must be made in the beginning anyway, so it does not matter if we keep one copy or many copies of $\mathcal{R}$ afterwards.

As remarked above, the main reason why we chose history preserving bisimulation was that they preserve the causal structure of processes. In particular, we have $a; b \sqcup b; a \not\sim a + b$, where a, b denote single action processes. This is reflected by the fact that h.p.b. equivalence is preserved under action refinement, and the two processes above can be distinguished by action refinement. Thus we need a finer equivalence than strong bisimulation equivalence etc.

Proposition 6.10 (Action refinement) *Let $\mathcal{P} = (A, X, \lambda)$ and $\mathcal{P}' = (A', X', \lambda')$ be two processes with $\mathcal{P} \sim \mathcal{P}'$. Let $b \in Act$ be refined by the process $\mathcal{Q} = (B, Y, \mu)$ to give* ref $\mathcal{P} = (C, Z, \nu)$ *and* ref $\mathcal{P}' = (C', Z', \nu')$*. Then* ref $\mathcal{P} \sim$ ref $\mathcal{P}'$*.*

Proof: We will restate the definition of refinement given earlier in a form suitable for refining many events at the same time. Let $A_b = \{a_i \in A \mid \lambda(a_i) = b\}$ be the set of events labeled b in $\mathcal{P}$. Then the events of *ref* $\mathcal{P}$ are $A - A_b \cup B \times A_b$, that is we replace each event in A_b, an a_i, with a copy of B. The states are defined by replacing the a_i's by states of Y as follows—if $x \in X$, then replace each $a_i \in x$ by $h_i \times \{a_i\}$, where h_i is maximal in Y. Secondly, if $x \xrightarrow{a_i} x'$, then every state generated from x in the first step is deleted, and we add a set of states generated from x' in which all the other a_i's are replaced by maximal states, and this a_i is replaced by any state of Y multiplied by $\{a_i\}$. Note that in both steps we generate a new state for each possible way of replacing the a_i's.

Let R be an h.p.b. between $\mathcal{P}$ and $\mathcal{P}'$. Define a relation R' between $ref\,\mathcal{P}$ and $ref\,\mathcal{P}'$ as follows. If $(x, x', f) \in R$ then since f is a label preserving isomorphism, both x and x' must have the same number of events labeled b, and these events are related by f. Now if z is derived from x and z' from x', such that each $a_i \in z$ was replaced by the same state from Y as $f(a_i) \in z'$ then $(z, z', f') \in R$, where f' is f on the non-a_i's, and $f'(b, a_i) = (b, f(a_i))$.

Now we can show that R' is a history preserving bisimulation. Since f is a bijection on the elements of x, it is clear that f' is a label preserving isomorphism between $h(z)$ and $h(z')$. Now suppose there is a transition $z \to z_1$ in $ref\,\mathcal{P}$. Some of the events in the transition are $\mathcal{P}$ events, others are from some copies of $\mathcal{Q}$. Thus this transition can be mimicked in $\mathcal{P}$ by doing the same $\mathcal{P}$ events and some a_i's, the ones that correspond to the events from the copies of $\mathcal{Q}$, from the state x that generated z to a state x_1 which generated z_1. Since $z \sim z'$, there must be x' such that $x \sim x'$. Now since R is a bisimulation, we must have an x_1' obtained by taking the same transitions from x', such that $x_1 \sim x_1'$. Now from x_1' we must have generated a state z_1' by replacing the a_i's by the same states of Y as in z_1, and thus we can show that $z_1 \sim z_1'$. The converse condition is satisfied similarly, showing that R' is an h.p.b. ∎

6.3 Connection with linear logic

As pointed out by Barr[Bar91], and also by Lafont and Streicher[LS91], the category of Chu spaces forms a model for linear logic developed by Girard [Gir87]. We have defined most of the linear logic operators above, the only ones that have not been defined are $\mathcal{P} \oplus \mathcal{Q} \triangleq (\mathcal{P}^\perp \otimes \mathcal{Q}^\perp)^\perp$, called "par", $!(A, X) \triangleq (A, 2^A)$, and $?(A, X) \triangleq (2^X, X)$. We have used a different notation than Girard, our notation confirms to standard categorical terminology.

Seely[See89] defines a Girard category and has proved that every Girard category is a model of linear logic. A Girard category is a symmetric monoidal closed category with finite products and a comonad $!\mathcal{P}$ such that $!(\mathcal{P} \times \mathcal{Q}) \cong !\mathcal{P} \otimes !\mathcal{Q}$ is a natural isomorphism, and $!\mathbf{1} \cong \top$. These conditions can easily be verified for **Chu**, so it is a model of linear logic.

Thus we can use linear logic as a verification system for Chu spaces. It does need to be extended though, since it does not have several important operators, like choice and sequential composition.

Completeness. The definition we use for the truth of a formula is the one suggested by Seely — A formula is valid if for every instantiation of its variables, the set A is non-empty. **Chu** is not a complete model of linear logic, as there are formulas which are not true in linear logic but which hold in **Chu**. An example of such a formula is $\mathcal{P} \multimap (\mathcal{P} \otimes \mathcal{P})$.

Chapter 7

Comparison with other models

In the previous chapters we have presented our model, Chu spaces, and have equipped it with an algebra to make it a language for specification of concurrent programs. In this chapter we will compare various other models of concurrency, and see how Chu relates to these. Since Chu spaces do not impose many constraints on the structure of the processes they represent, it should come as no surprise that they can embed a lot of these models. The more interesting results are those which show that the generality does not lose the power of these models.

7.1 Event Structures and Petri nets

Petri nets were the first of the models for concurrency, and event structures were developed in [NPW81] to equip them with an algebra, and bridge the gap between automata theory and domain theory. Chu spaces generalize event structures by removing those restrictions on event structures which are designed to make them match closely to Petri nets.

An event structure is defined as a triple $(E, Con, \vdash)$, where E is a set of events, Con is a non-empty collection of finite subsets of E, such that $X \in Con$ and $Y \subseteq X \Rightarrow Y \in Con$, and $\vdash \subseteq Con \times E$ satisfies $X \vdash e$ and $X \subseteq Y \in Con \Rightarrow Y \vdash e$. Con is the consistency predicate, giving the conflict-free finite subsets of E, while $\vdash$ is the enabling relation.

To every event structure $(E, Con, \vdash)$ we can now associate a Chu space, whose set of events is E, and whose logical formula Γ is derived from Con and $\vdash$. For every finite subset of E, $X \notin Con$, add the clause $\bigwedge X \rightarrow 0$ to the formula Γ. For an event e, let $X_i \vdash e$ be the elements of the enabling relation with e on the right side. For each event e, add the clause $e \rightarrow \bigvee_i \bigwedge X_i$ to Γ. (Note that $\bigwedge \emptyset = 1$ and $\bigvee \emptyset = 0$. $a \rightarrow 0$ can be written as $\neg a$.) Now Γ is the conjunction of all these clauses.

Thus we can form a Chu space corresponding to an event structure. However, several different event structures could correspond to the same Chu space. This is because the axioms for event structures are not strong enough to deduce all the properties that can be deduced by boolean logic. For example in the event structure with $E = \{a, b, c\}$, $Con = 2^E - \{a, b, c\}$ and $c \vdash a$ and other tuples added according to the axiom above, it is clear that the set $\{a, b\}$ is not a consistent set, since a requires c and $\{a, b, c\}$ is not consistent. This deduction is made by the boolean logic, but not by the event structure rules. So two event structures which looked like the one above except that one did not have $\{a, b\} \in Con$ would be mapped to the same Chu space.

A configuration of an event structure is a set of events which could have occurred in some execution of the process it represents. Thus for every finite set of events in a configuration, it should be consistent, and secondly, for each event e, some set of events enabling it should have occurred before it. This is expressed by saying that it is secured, i.e. there is a finite sequence of events in the configuration, $e_0, e_1, \ldots, e_n = e$, such that $e_0, \ldots e_i \vdash e_{i+1}$ for all $i < n$. The collection of all configurations of an event structure $E = (E, Con, \vdash)$ is denoted $\mathcal{F}(E)$.

Proposition 7.1 *Each configuration of an event structure is a state of the corresponding Chu space.*

Proof: Since every finite set in a configuration must be consistent, thus all the clauses added to Γ by Con are satisfied. Also, every event has some set of events which enable it in the configuration, so all the clauses due to $\vdash$ are also satisfied, thus satisfying the entire formula Γ. Thus the configuration is a state ∎

There may however be some states that are not configurations, due to the fact that they are not secured (every state must be consistent). For example, if $E =$

$\{a, b\}, Con = 2^E, a \vdash b, b \vdash a$, then the only configuration possible is the empty one. However ab is also a state of the Chu space. We can eliminate the states that are unsecured if we wish to.

An alternative embedding of $(E, Con, \vdash)$ is to map it to the Chu space $(E, \mathcal{F}(E))$. This is also called the Chu configuration structure corresponding to $(E, Con, \vdash)$. For related work on embedding event structures into Chu spaces, see [Plo94a].

Thus the objects of Winskel's category of event structures can be embedded in **Chu**, in such a way that their concurrency properties are preserved. However the morphisms of Winskel's category are not the same as the ones for Chu spaces. Event morphisms preserve consistency of events, whereas Chu morphisms preserve conflict. We can provide an alternative category whose objects are the same as **Chu**, but whose morphisms are different — Given (A, X, R) and (B, Y, S), a *hypergraph* morphism is a pair of maps $f : A \to B$ and $g : X \to Y$, such that $R(a, x) = S(f(a), g(x))$ for all $a \in A, x \in X$. This category is called **Hgr**, since the objects can be interpreted as hypergraphs[Ber73], with A being the vertices of the graph, and X the set of hyperedges. Then these morphisms correspond to hypergraph morphisms [DW80]. Notice that here the direction of g is the same as that of f, so if X is interpreted as a subset of the power set 2^A, then these morphisms are based on the covariant powerset functor on sets, whereas Chu morphisms are based on the contravariant powerset functor. For **Hgr** morphisms $g(x) = f(x)$, where $f(x) = \{f(a) \mid a \in x\}$.

If we embed an event structure as a configuration structure, the synchronous morphisms of Winskel [Win86] are special cases of hypergraph morphisms, namely those morphisms which are injections when restricted to any state ($\forall x \in X.$ $[f\restriction_x$ is injective $]$). A finite set $X \subseteq E$ is consistent iff there is a configuration $x \in \mathcal{F}(E)$ with $X \subseteq x$. Thus if (f, g) is a hypergraph morphism, then $f(X) \subseteq g(x)$, so $f(X)$ is consistent. Also $X \vdash e$ iff X is consistent and there is a configuration $x \in \mathcal{F}(E)$ such that $e \in x \subseteq X \cup \{e\}$. Now $f(X)$ is consistent, and $f(e) \in g(x)$ and $g(x) \subseteq f(X \cup \{e\})$. Thus $f(X) \vdash f(e)$.

The main reason why we prefer to use the category **Chu** rather than **Hgr** is that **Hgr** lacks the duality principle we saw in chapter 4. This duality is important as it enables us to generalize various Stone dualities, as well as convert between automata and schedules. This cannot be done in **Hgr**.

Two of the operations we had defined earlier correspond naturally to the sum and product operations defined on the category of event structures [Win86]. Choice corresponds to sum of event structures, and in fact is coproduct in the category **Hgr**. Partial synchronous product is the product of event structures. If we changed the definition of morphisms of **Hgr** to partial maps (i.e. made f partial and then g is just f on sets), then this would be product in the new category.

Proposition 7.2 *If $E_1 = (E_1, Con_1, \vdash_1)$ and $E_2 = (E_2, Con_2, \vdash_2)$ are two event structures, and $\mathcal{P}_1 = (E_1, \mathcal{F}(E_1))$ and $\mathcal{P}_2 = (E_2, \mathcal{F}(E_2))$ their Chu configuration structures, then the choice $\mathcal{P}_1 \sqcup \mathcal{P}_2$ corresponds to the event structure $E_1 + E_2$.*

Proof: The event set of $E_1 + E_2$ is the disjoint union of E_1 and E_2, which is the same as the events of $\mathcal{P}_1 \sqcup \mathcal{P}_2$. A subset $X \subseteq E_1 \uplus E_2$ is consistent iff X belongs only to one of E_1 or E_2, and is consistent there. Thus any configuration of the sum can consist of events only in E_1 or events only in E_2, except for the empty configuration, which is common. The vdash relation is similarly the union of the vdash relations of E_1 and E_2, so any configuration of E_1 is a configuration of the sum, and the same for E_2. Thus the choice $\mathcal{P}_1 \sqcup \mathcal{P}_2$ is isomorphic to $(E_1 \uplus E_2, \mathcal{F}(E_1) + \mathcal{F}(E_2))$. ∎

The fact that choice is coproduct for **Hgr** can be easily verified. There are maps $(f_i, g_i) : \mathcal{P}_i \to \mathcal{P}_1 \sqcup \mathcal{P}_2$ given by $f_1(a) = a$ and $g_1(x) = x$, and similarly for $\mathcal{P}_2$. If any other $\mathcal{Q}$ has maps $(f_i', g_i') : \mathcal{P}_i \to \mathcal{Q}$, then the unique map from $\mathcal{P}_1 \sqcup \mathcal{P}_2 \to \mathcal{Q}$ can be given by the juxtaposition of the f_i''s.

In order to match up partial synchronous product exactly with the product of event structures, it is necessary to add one more condition on the states — if a state x was generated from states x_1 and x_2, then there should be a Chu space map from $h(x_1)$[1] to $h(x)$, carrying each event to the event it forms in the construction, and a similar map from $h(x_2)$. This eliminates states like $\{(a,d),(b,c)\}$ in $a.b.\mathbf{0} \parallel c.d.\mathbf{0}$, since $b \to a$ is a property of the the history of the state $\{a,b\}$, but $(b,c) \to (a,d)$ is not a property of the history of $\{(a,d),(b,c)\}$.

[1] $h(x)$ was defined in the previous chapter as the history of $x = (x, \{y \in X \mid y \subseteq x\})$.

Proposition 7.3 *If $E_1 = (E_1, Con_1, \vdash_1)$ and $E_2 = (E_2, Con_2, \vdash_2)$ are two event structures, and $\mathcal{P}_1 = (E_1, \mathcal{F}(E_1))$ and $\mathcal{P}_2 = (E_2, \mathcal{F}(E_2))$ their Chu configuration structures, then the partial synchronous product $\mathcal{P}_1 \parallel \mathcal{P}_2$ corresponds to the event structure $E_1 \times E_2$.*

Proof: The events of $\mathcal{P}_1 \parallel \mathcal{P}_2$ clearly correspond to the events of $E_1 \times E_2$. In proposition 2.2.3 on [Win86, p.349] Winskel has characterized the configurations of $E_1 \times E_2$. These are those subsets of the event sets such that they originate from a configuration of E_1 and a configuration of E_2, do not synchronize any event of E_1 with two different events of E_2 or vice versa, and two more conditions ensuring that every event is finitely supported. All of these can be shown to be equivalent to the conditions we have placed on states of $\mathcal{P}_1 \parallel \mathcal{P}_2$. ∎

The proof that parallel composition is the product in the category of partial hypergraph morphisms is routine. If we restrict $\mathcal{P}_1 \parallel \mathcal{P}_2$ to the events from $A \times B$, then we get the synchronous product of $\mathcal{P}_1$ and $\mathcal{P}_2$, and this is the product in the category **Hgr**.

There is no universally accepted definition of sequential composition for event structures. Baeten and Vaandrager[BV92] define sequential composition, and our definition does agree with theirs operationally. They define a special event, a $\surd$ which is interpreted as the last event of the process represented by the event structure. Sequential composition $E_0; E_1$ is then defined by refining each $\surd$ of E_0 with E_1. Since any state in which $\surd$ has been performed is a maximal state, our definition of $\mathcal{P}; \mathcal{Q}$ makes the states of $\mathcal{P}; \mathcal{Q}$ be the configurations of $E_0; E_1$, except for an extra $\surd$ event in some states. This can be hidden by labeling it with a τ.

Thus Chu spaces can mimic the behavior of event structures by being able to represent any concurrent phenomenon that event structures can represent (mainly conflict and enabling), and can also be combined to duplicate the various operators that event structures possess. The definition of a Chu space is simpler than the definition of an event structure, and Chu spaces can represent more phenomena, like causality and nondeterminism (chapter 5).

In an event structure, the enabling relation is quite simple — an event can be enabled by one of several sets of events. In [Gun91, Gun92] Gunawardena strengthened this to allow for more complicated enabling conditions to get a new model for concurrency called causal automata. In a causal automaton, each event is associated with a finitary boolean formula over the set of events, and is enabled any time the formula becomes true (where a literal becomes true whenever it has occurred as an event). Now in fact the consistency predicate is no longer necessary, as it can be encoded into the enabling relation. In order to deal with infinitary enabling formulae, Gunawardena introduced geometric automata, where the formulae could be infinite with some restrictions.

Geometric automata cannot be encoded by Chu spaces. The reason is that in a geometric automaton, the enabling formula must be true only at the instant an event is taking place, it could become false later. But in a Chu space, if we have $a \to \phi$, then ϕ must always stay true after a has occurred. Thus geometric automata can represent deadlock between two events, by making each the enabling formula for the other, but deadlock cannot be represented by Chu spaces. Another difference is that the enabling formula must have become true *before* the event takes place, while in a Chu space it could become true at the same time. Geometric automata are however an interleaving model of concurrency. They cannot represent the distinction between causality and enabling, and between nondeterminism and choice.

Petri nets and Chu spaces. Event structures were originally developed to provide an algebra for Petri nets by Nielsen, Plotkin and Winskel in [NPW81]. They considered safe Petri nets, that is those in which no place can ever have more than one token. They then unfolded the net to remove all loops, resulting in an occurrence net, which is acyclic, so no transition gets fired more than once. This can then be represented as an event structure.

Since Chu spaces embed event structures, Chu spaces can represent any behavior that safe Petri nets can model. Winskel [Win86] gives a translation function from safe nets to event structures, which we extend to Chu spaces. Winskel also gives a way of constructing an occurrence net from a prime event structure, but this is not possible for Chu spaces, since Chu spaces model behaviors which cannot be modeled by Petri nets. For example, Petri nets cannot distinguish between causality and enabling,

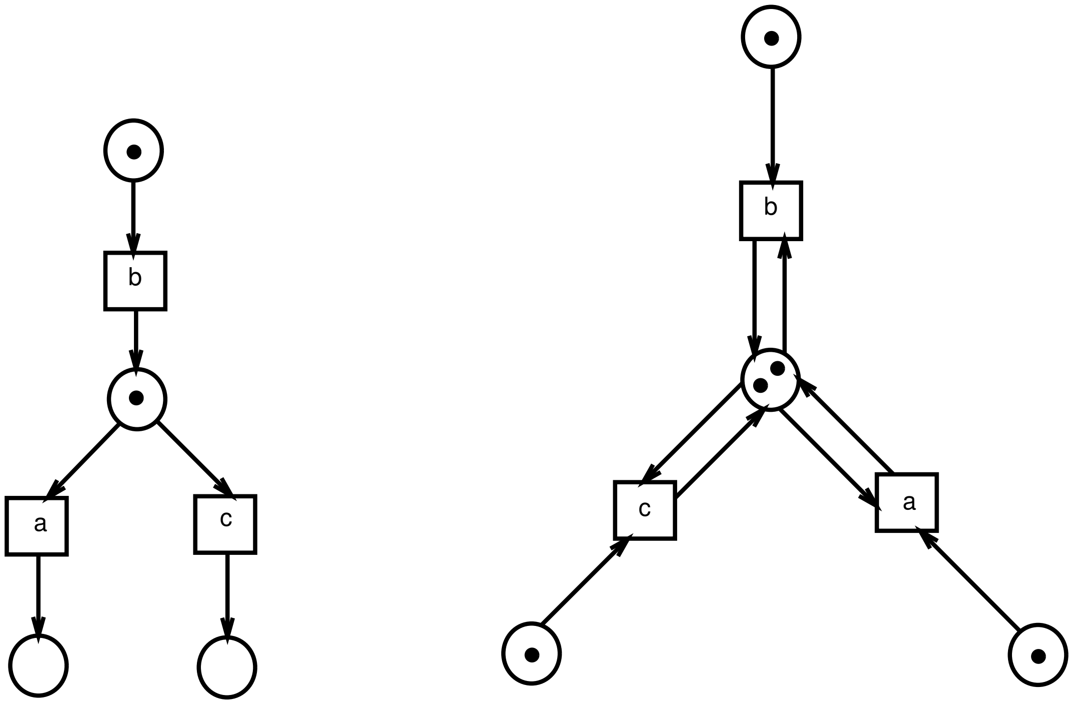


Figure 7.1: A Petri net for postponed concurrency, and one which cannot be represented as a Chu space.

while Chu spaces can.

Chu spaces can mimic the behavior of some unsafe Petri nets that cannot be modeled as event structures. An example of this is postponed concurrency (figure 5.2), which can be represented by an unsafe Petri net. However, it is not possible to represent the behavior of each Petri net as a Chu space faithfully, where by faithful we mean as a Chu space that is history preserving bisimilar to the Petri net. An example of such a Petri net is given in figure 7.1, this is the net that can do at most two events out of three concurrently (proposed by Rob van Glabbeek).

A more syntactic approach to representing Petri nets by matrices using the Chu construction has been followed by Brown and Gurr [BG90]. They model a general

Petri net as a pair of matrices representing the pre and post conditions. The morphisms must satisfy adjointness conditions for both matrices, and are shown to be simulations of Petri nets.

7.2 A Chu Semantics for CCS and CSP

The calculus of communicating systems was developed by Robin Milner as an algebraic means of specifying concurrent systems. The atomic processes are actions, which cannot be further subdivided (as this is an interleaving model). A process executes an action to evolve into another process. Actions are labeled from a set of labels called Act, which has a set of labels (labels are denoted by $a, b, \ldots$), their complementary labels denoted by $\bar{a}, \bar{b}, \ldots$ and a special label τ. Processes can be built up from other processes by six operators:

$$P ::= \mathbf{0} \mid a.P \mid P \sqcup P \mid P|P \mid P \backslash L \mid P[f] \mid \mu x.\, P$$

The basic process is $\mathbf{0}$, which does nothing. $a.P$ is called prefixing, the action labeled a is executed and then the process behaves like P. $P \sqcup Q$ is choice, this process nondeterministically chooses between processes P and Q, and then continues doing the chosen process. We may have a choice between an infinite number of processes. The other operators are parallel composition, restriction, relabeling and recursion. The parallel composition is a communicating parallel composition, where the two components can either do their individual actions independently, or can simultaneously do a pair of complementary actions, which then are labeled with a τ, a silent action.

We can associate a labeled Chu space with each process. $\mathbf{0}$ is the Chu space $\mathbf{0}$, and the other connectives are those described in chapter 6. $P|P$ is modeled by partial synchronous product, subject to the condition on histories specified in the previous section. Its singleton events are labeled as they were before, and any pair of events whose pair of labels are not complementary is removed by restriction. Each pair is then labeled by a τ. The restriction operator is on labels rather than events. Since all the operators are continuous, recursive definitions are possible. Morphisms between these Chu spaces are Chu maps (f, g) in which f preserves labels.

Thus Chu spaces form a model for CCS. This model does not satisfy the interleaving law — we do not have $a.\mathbf{0} \parallel b.\mathbf{0} \cong a.b.\mathbf{0} \sqcup b.a.\mathbf{0}$. Later we will show some kinds of equivalences for which this law does hold. We will now prove a close connection between the semantics of a CCS expression and its Chu space automaton.

Proposition 7.4 *Given a CCS expression P and its Chu space $\mathcal{P}$, it is possible to label each state x of the Chu space with a CCS expression $P(x)$ such that*

1. *$P(\emptyset) = P$, that is the start state is labeled with P.*

2. *There is a transition in the automaton $x \xrightarrow{e} x'$ iff there is a CCS transition $P(x) \xrightarrow{\lambda(e)} P(x')$*

Proof: We will show this by induction on the structure of CCS expressions.

1. It holds for $\mathbf{0}$ trivially, as there is only one state which is labeled with $\mathbf{0}$, and no transitions.

2. If $P = a.Q$, then the Chu space for a is $(\{e\}, \{\emptyset, \{e\}\}, \lambda : e \mapsto a)$. So there is one maximal state $\{e\}$. Now any state of $\mathcal{P}$ will be either the state $\emptyset$ or $y \cup \{e\}$, where y is a state of $\mathcal{Q}$, the Chu space for Q. Label $\emptyset$ with P and each state $y \cup \{e\}$ with the expression labeling y in $\mathcal{Q}$. The conditions of the theorem are satisfied by the induction hypothesis.

3. If $P = P_1 \sqcup P_2$, label the empty state with P. Any other state in $\mathcal{P}$ comes uniquely from a state in $\mathcal{P}_1$ or $\mathcal{P}_2$, the Chu spaces for P_1 and P_2. Label it with the expression with which it was labeled in that Chu space. Now there is a transition from $\emptyset$ to a state in $\mathcal{P}$ iff there was a similar transition in either $\mathcal{P}_1$ or $\mathcal{P}_2$, which are exactly the possible CCS transitions from P.

4. If $P = P_1 \parallel P_2$, each state x of $\mathcal{P}$ comes from a state x_1 of $\mathcal{P}_1$ and a state x_2 of $\mathcal{P}_2$. Let $P(x) = P(x_1) \parallel P(x_2)$, the parallel composition of the processes labeling of the pair of states. Now the state $\emptyset$ is labeled with P, since it could arise only from the $\emptyset$ states in $\mathcal{P}_1$ and $\mathcal{P}_2$, which by the induction hypothesis

were labeled by P_1 and P_2 respectively. Also if $P(x) = P(x_1) \parallel P(x_2)$ and there is a transition $x \xrightarrow{e} x'$, then $e = e_1$ or e_2 or (e_1, e_2). Then there must have been a transition $x_1 \xrightarrow{e_1} x_1'$ or $x_2 \xrightarrow{e_2} x_2'$ or both, by looking at the construction of x and x'. Also, if both occurred, the labels on e_1 and e_2 must be complementary. These are exactly the possible CCS transitions from $P(x_1) \parallel P(x_2)$, so the proposition holds.

5. The cases $P = P' \backslash L$ and $P = P'[f]$ are trivial. Each new state x comes from a state x', and has the label is $P(x) = P(x')\backslash L$ or $P(x')[f]$.

6. $P = \mu A.Q[A]$. We will write this in the equational form, $P = Q(P)$. Now by induction hypothesis, we can label the Chu spaces for $\mathbf{0}, Q(\mathbf{0}), Q(Q(\mathbf{0})), \ldots$. The maps between these which form these spaces into an ω-chain take each state labeled by $p(Q^i(\mathbf{0}))$, where p is any process expression in the Chu space for $Q(\mathbf{0})$, to $p(Q^{i-1}(\mathbf{0}))$ for $i > 0$, and all the states in $p(\mathbf{0})$ to a state labeled $\mathbf{0}$. Now if x is any state in the colimit, label it with $p(P)$ if a map takes it to $p(Q^i(\mathbf{0}))$, the commutativity ensures that this is well defined. Thus the start state will be labeled P. The second condition follows as for each i, the Chu space for $Q^i(\mathbf{0})$ satisfies it.

∎

This theorem shows that the Chu space for a CCS expression is strongly bisimilar to the derivation tree for that expression. It also shows that there is a strong bisimulation between Chu spaces which represent strongly bisimilar CCS expressions, thus showing that if we define an equivalence relation between two Chu spaces to be bisimulation equivalence, then all the CCS laws held under strong bisimulation will be true for this equivalence. In particular, the Chu spaces for $a.\mathbf{0} \parallel b.\mathbf{0}$ and $a.b.\mathbf{0} \sqcup b.a.\mathbf{0}$ are bisimilar. The main difference between the semantics of a CCS expression and its Chu space is that Chu spaces allow execution of many events at the same time, whereas CCS allows only one at a time. However strong bisimulation considers only one event at a time, which is why this proposition holds.

We have already drawn a connection between simulations of processes and Chu morphisms in chapter 4.

The various laws that Chu spaces satisfy have already been stated in chapter 6. The laws $(a.Q)\backslash L \cong \mathbf{0}$ if $a \in L, a.Q\backslash L$ otherwise, and $(a.Q)[f] \cong f(a).Q[f]$ are also valid. It can be seen that the only CCS laws that Chu spaces do not satisfy are the expansion law, the τ-laws [Mil89] and the idempotence of choice. The last can be remedied by using history preserving bisimulation as our notion of Chu space equivalence. Successively coarser equivalences enable the satisfaction of the other laws. It is possible to define equivalences between Chu spaces based upon each of the equivalences between processes [vG90]. We will not go into these here.

A Chu space semantics can similarly be given for the language of Communicating Sequential Processes of Hoare [BHR84, Hoa78, Hoa85]. Some operators of CSP — prefixing, choice, renaming and concealment can be modeled as above. Concurrence $\mathcal{P} \parallel \mathcal{Q}$ can be modeled by partial synchronous product, with any singleton events from $\mathcal{P}$ restricted away if they also occur in $\mathcal{Q}$, and vice versa (actions which occur in both processes must synchronize). Also, any paired event the labels of whose components are not the same is restricted away, and the label of the other paired events is the label of either component. Interleaving is modeled by sum of Chu spaces, as there is no communication, while communication can be handled with labels. Sequential composition can be done using our sequential composition operator. Most of the standard CSP laws hold for this model.

Once again the semantics of CSP is interleaving, but this model is non-interleaving. We cannot model nondeterministic choice of CSP very well, since CSP choice does not respect branching time, whereas Chu spaces do. If we decide to ignore this law, then nondeterministic choice can be modeled by the choice between $\delta_1.\mathcal{P} \sqcup \delta_2.\mathcal{Q}$, with the start state deleted, and the δ labeled as silent actions. This process is forced to start out in either $\mathcal{P}$ or $\mathcal{Q}$, so the choice is nondeterministic.

Chapter 8

Future Research

We indicate several possibilities for research opened up by our work on Chu spaces.

Expressiveness of Chu spaces. We would like to get a better idea of the kinds of concurrency features that can or cannot be represented by Chu spaces. In particular, we would like to understand the relation of Chu spaces with Petri nets and other models better, by showing which processes representable in these models can be represented as Chu spaces. In addition, the notion of what it means for a process to be represented as a Chu space could be investigated further. For example, we say that a Petri net is adequately represented as a Chu space if one can define a history preserving bisimulation between them. The notion of which equivalence to use is of course governed by what are the interesting distinctions, and thus various equivalences could be investigated.

In this thesis we did not investigate the ideas of fair computations for Chu spaces. It is possible to define these, and further research on the representation of various kinds of fairness promises to be interesting.

Extending the Algebra. We have presented some of the possible operators for an algebra of Chu spaces, these could be augmented by others. In addition, while we presented a number of identities, it would be nice to know all the possible identities between Chu spaces, given by a set of axioms.

Logic for Chu spaces. While we have indicated that linear logic is a logic for Chu spaces, it is not entirely satisfactory for various reasons. It is not complete, as there are theorems which are true for Chu spaces which are not valid in linear logic. Thus we can either develop a logic which would match the logic of Chu spaces, or strengthen the definition of a Chu space theorem to eliminate all non-theorems of linear logic. Secondly, linear logic does not have many of the operators in the algebra of Chu spaces. So if we wish to use linear logic as a verification logic, it must be augmented to be able to prove formulas containing these operators.

Applications. The algebra of Chu spaces suggests a powerful specification language, as was indicated in chapter 5. A language with its semantics based on Chu spaces would be a very useful tool, and should make specification and verification of concurrent systems a lot easier — this is the goal for which Chu spaces were constructed.

Bibliography

[Bar79] M. Barr. $*$-*Autonomous categories, LNM 752*. Springer-Verlag, 1979.

[Bar91] M. Barr. $*$-Autonomous categories and linear logic. *Math Structures in Comp. Sci.*, 1(2), 1991.

[Ber73] Claude Berge. *Graphs and Hypergraphs (transl: E. Minieka)*. North-Holland, Amsterdam, 1973.

[Bes86] E. Best. Cosy: its relation to nets and csp. In *Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, LNCS 255*, Bad-Honnef, September 1986. Springer-Verlag.

[BG90] C. Brown and D. Gurr. A categorical linear framework for Petri nets. In J. Mitchell, editor, *Logic in Computer Science*, pages 208–218. IEEE Computer Society, June 1990.

[BGdP91] C. Brown, D. Gurr, and V. de Paiva. A linear specification language for Petri nets. Technical Report DAIMI PB-363, Computer Science Department, Aarhus University, October 1991.

[BHR84] S.D. Brookes, C.A.R. Hoare, and A.D. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31:560–599, 1984.

[BV92] J.C.M. Baeten and F.W. Vaandrager. An algebra for process creation. *Acta Informatica*, 29(4):303–334, 1992.

[Cas91] Ross Casley. *On the Specification of Concurrent Systems*. PhD thesis, Stanford University, January 1991.

[CCMP91] R.T Casley, R.F. Crew, J. Meseguer, and V.R. Pratt. Temporal structures. *Math. Structures in Comp. Sci.*, 1(2):179–213, July 1991.

[Cre91] R.F. Crew. *Metric Process Models.* PhD thesis, Stanford University, December 1991.

[DDNM88] P. Degano, R. De Nicola, and U. Montanari. A distributed operational semantics for CCS based on condition/event systems. *Acta Informatica*, 26(1/2):59–91, October 1988.

[Dev88] R. Devillers. On the definition of a bisimulation notion based on partial words. *Petri Net Newsletter*, 29:16–19, 1988.

[dP89] V. de Paiva. The dialectica categories. In *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*, pages 47–62, held June 1987, Boulder, Colorado, 1989.

[Dro89] M. Droste. Event structures and domains. *Theoretical Computer Science*, 68:37–47, 1989.

[DW80] W. Dörfler and D.A. Waller. A category-theoretical approach to hypergraphs. *Archives of Mathematics*, 34(2):185–192, 1980.

[GG89] R.J. van Glabbeek and U. Goltz. Equivalence notions for concurrent systems and refinement of actions. In A. Kreczmar and G. Mirkowska, editors, *Proc. Conf. on Mathematical Foundations of Computer Science*, volume 379 of *Lecture Notes in Computer Science*, pages 237–248. Springer-Verlag, 1989.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gis88] J.L. Gischer. The equational theory of pomsets. *Theoretical Computer Science*, 61:199–224, 1988.

[GJ92] E. Goubault and T.P. Jensen. Homology of higher dimensional automata. In *Proc. of CONCUR'92, LNCS 630*, pages 254–268, Stonybrook, New York, August 1992. Springer-Verlag.

[GP87] H. Gaifman and V.R. Pratt. Partial order models of concurrency and the computation of functions. In *Proc. 2nd Annual IEEE Symp. on Logic in Computer Science*, pages 72–85, Ithaca, NY, June 1987.

[GP93] V. Gupta and V.R. Pratt. Gates accept concurrent behavior. In *Proc. 34th Ann. IEEE Symp. on Foundations of Comp. Sci.*, pages 62–71, November 1993.

[Gra81] J. Grabowski. On partial languages. *Fundamenta Informaticae*, IV.2:427–498, 1981.

[Gun91] J. Gunawardena. Geometric logic, causality and event structures. In J. C. M. Baeten and J. F. Groote, editors, *CONCUR'91 - 2nd International Conference on Concurrency Theory*, pages 266–280. Springer LNCS 527, 1991.

[Gun92] J. Gunawardena. Causal automata. *Theoretical Computer Science*, 101:265–288, 1992.

[Gup93] V. Gupta. Concurrent kripke structures. In *Proceedings of the North American Process Algebra Workshop, Cornell CS-TR-93-1369*, August 1993.

[GV87] R.J. van Glabbeek and F.W. Vaandrager. Petri net models for algebraic theories of concurrency. In *Proc. PARLE, II, LNCS 259*, pages 224–242. Springer-Verlag, 1987.

[Hoa78] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–672, August 1978.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[JNW93] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation and open maps. In M.Y.Vardi, editor, *Logic in Computer Science*, pages 418–427. IEEE Computer Society, July 1993.

[Joh82] P.T. Johnstone. *Stone Spaces.* Cambridge University Press, 1982.

[LR75] P.E. Lauer and R.H.Campbell. Formal semantics of a class of high-level primitives for coordinating concurrent processes. *Acta Informatica*, 5:297–332, 1975.

[LS91] Y. Lafont and T. Streicher. Games semantics for linear logic. In *Proc. 6th Annual IEEE Symp. on Logic in Computer Science*, pages 43–49, Amsterdam, July 1991.

[Mac71] S. Mac Lane. *Categories for the Working Mathematician.* Springer-Verlag, 1971.

[Mil80] R. Milner. *A Calculus of Communicating Systems, LNCS 92.* Springer-Verlag, 1980.

[Mil83] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[MM90] J. Meseguer and U. Montanari. Petri nets are monoids. *Information and Control*, 88(2):105–155, October 1990.

[MMT87] R. McKenzie, G. McNulty, and W. Taylor. *Algebras, Lattices, Varieties, Volume I.* Wadsworth & Brooks/Cole, Monterey, CA, 1987.

[NPW81] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures, and domains, part I. *Theoretical Computer Science*, 13, 1981.

[OGG88] E.-R. Olderog, U. Goltz, and R.J. van Glabbeek. Combining compositionality and concurrency, summary of a GMDworkshop, Königswinter, March 1988. Arbeitspapiere der GMD 320, Gesellschaft für Mathematik und Datenverarbeitung, 1988.

[Par81] D. Park. Concurrency and automata on infinite sequences. In *Proc. Theoretical Computer Science, LNCS 104*, pages 167–183. Springer-Verlag, 1981.

[Pet62a] C.A. Petri. Fundamentals of a theory of asynchronous information flow. In *Proc. IFIP Congress 62*, pages 386–390, Munich, 1962. North-Holland, Amsterdam.

[Pet62b] C.A. Petri. *Kommunikation mit Automaten.* PhD thesis, Schriften des Institutes für Instrumentelle Mathematik, 1962.

[Plo94a] G.D. Plotkin. Notes on event structures and chu. Manuscript available as pub/gdp2.dvi by anonymous FTP from Boole.Stanford.EDU, February 1994.

[Plo94b] G.D. Plotkin. Notes on the chu construction and recursion. Manuscript available as pub/gdp.dvi by anonymous FTP from Boole.Stanford.EDU, February 1994.

[Plo94c] G.D. Plotkin. Personal communication. July 1994.

[PP92] G Michele Pinna and Axel Poigne. On the nature of events. In *Proc. 17th Symposium on Mathematical Foundations of Computer Science, LNCS 629*, 1992.

[Pra82] V.R. Pratt. On the composition of processes. In *Proceedings of the Ninth Annual ACM Symposium on Principles of Programming Languages*, January 1982.

[Pra86] V.R. Pratt. Modeling concurrency with partial orders. *Int. J. of Parallel Programming*, 15(1):33–71, February 1986.

[Pra91] V.R. Pratt. Modeling concurrency with geometry. In *Proc. 18th Ann. ACM Symposium on Principles of Programming Languages*, pages 311–322, January 1991.

[Pra92a] V.R. Pratt. Arithmetic + logic + geometry = concurrency. In *Proc. First Latin American Symposium on Theoretical Informatics, LNCS 583*, pages 430–447, São Paulo, Brazil, April 1992. Springer-Verlag.

[Pra92b] V.R. Pratt. The duality of time and information. In *Proc. of CONCUR'92, LNCS 630*, pages 237–253, Stonybrook, New York, August 1992. Springer-Verlag.

[Pra92c] V.R. Pratt. Event spaces and their linear logic. In *AMAST'91: Algebraic Methodology and Software Technology*, Workshops in Computing, pages 1–23, Iowa City, 1992. Springer-Verlag.

[Pra93] V.R. Pratt. The second calculus of binary relations. In *Proceedings of MFCS'93*, pages 142–155, Gdańsk, Poland, 1993. Springer-Verlag.

[Pri70] H.A. Priestley. Representation of distributive lattices. *Bull. London Math. Soc.*, 2:186–190, 1970.

[PS82] G.D. Plotkin and M.B. Smyth. The category-theoretic solution of recursive domain equations. *SIAM Journal on Computing*, 11(4):761–783, 1982.

[Rei85] W. Reisig. *Petri Nets: An Introduction.* Springer-Verlag, 1985.

[See89] R.A.G Seely. Linear logic, $*$-autonomous categories and cofree algebras. In *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*, pages 371–382, held June 1987, Boulder, Colorado, 1989.

[vG90] R. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions.* PhD thesis, Vrije Universiteit te Amsterdam, May 1990.

[Win86] G. Winskel. Event structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, LNCS 255*, Bad-Honnef, September 1986. Springer-Verlag.

[Win88] G. Winskel. A category of labelled Petri nets and compositional proof system. In *Proc. 3rd Annual Symposium on Logic in Computer Science*, Edinburgh, 1988. Computer Society Press.