Universidade Federal do Rio Grande do Sul
Instituto de Informática
Curso de Pós-Graduação em Ciência da Computação

Estratégias de Computação Seqüenciais e Paralelas sobre Espaços Coerentes

por

Ruben Gerardo Schneider Sellanes

Dissertação submetida como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Antônio Carlos da Rocha Costa
Orientador

Porto Alegre, março de 1996

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Schneider Sellanes, Ruben Gerardo

Estratégias de Computação Seqüenciais e Paralelas sobre Espaços Coerentes / Ruben Gerardo Schneider Sellanes. — Porto Alegre: CPGCC da UFRGS, 1995.
144 p.: il.

Dissertação (mestrado) — Universidade Federal do Rio Grande do Sul, Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, BR-RS, 1996. Orientador: Costa, Antônio Carlos da Rocha.

1. Espaços coerentes. 2. Estruturas de Dados Concretas. 3. Domínios Concretos. 4. Estruturas de Eventos. 5. Domínios de Eventos. 6. Funções Lineares. 7. Funções Estáveis. 8. Funções Seqüenciais. 9. Algoritmos Seqüenciais. 10. Algoritmos Lineares. 11. Seqüencialidade. 12. Paralelismo. 13. Semântica. I. Costa, Antônio Carlos da Rocha. II. Estratégias de Computação Seqüenciais e Paralelas sobre Espaços Coerentes.

A Fernando (in memoriam)
y a mis padres
Guillermo y Elizabeth

AGRADECIMENTOS

Desejo agradecer a todas as pessoas que contribuiram de alguma forma para a realização deste trabalho, especialmente ao meu orientador, Prof. Dr. Antônio Carlos da Rocha Costa, pela dedicação e orientação dispensados e por todo o apoio que me foi oferecido.

Quero agradecer também, de uma forma especial, à Prof. Dra. Laira Vieira Toscani pelas suas sugestões e análise crítica, fundamentalmente pela amizade e apoio demonstrados nestes anos do curso de mestrado, assim como também ao Prof. Dr. Simão Toscani.

Agradeço também a todos os meus amigos e colegas do mestrado, especialmente ao Elton Dietrich e a Liara Leal pela sincera amizade, assim como também a todo o grupo da Matemática Computacional do CPGCC.

Um agradecimento especial às colegas Graçaliz Dimuro e Renata Reiser pela ajuda no português e no conteúdo do trabalho, e particularmente a meu amigo e colega Afonso Cardoso sem o qual a impressão teria sido impossível.

Agradeço especialmente a Glaucia pelo carinho, paciência e apoio dispensados durante a realização do curso de mestrado, aos meus pais Guillermo e Elizabeth pela ajuda, incentivo, apoio e compreensão, assim como também pelo exemplo de dedicação e honestidade, que sempre me demonstraram.

Finalmente agradeço ao CNPq pelo suporte financeiro que me foi concedido durante o curso de mestrado.

SUMÁRIO

LISTA DE ABREVIATURAS

CCC	Categoria Cartesiana Fechada
cds	Estrutura de Dados Concreta
cpo	Ordem Parcial Completa
sss	Se e somente se

LISTA DE FIGURAS

RESUMO

As estruturas de dados concretas (cds) são quaternas (C, V, E, $\vdash$) que contêm um conjunto C de células, um conjunto V de valores, um conjunto E de eventos e uma relação de habilitação $\vdash$. O conjunto de estados de uma cds é um domínio concreto, que pode ser considerada a parte "abstrata" das cds. Da mesma maneira tem-se que os domínios de eventos (que são generalizações dos domínios concretos) são a parte abstrata das estruturas de eventos. Mostra-se a relação dos domínios concretos e domínios de eventos com os espaços coerentes, assim como também das teias de espaços coerentes com as cds e estruturas de eventos. Intuitivamente, uma cds é uma teia de um espaço coerente se toda célula c de C não é habilitada por nenhum evento (ou equivalentemente, é habilitada pelo conjunto vazio), isto é, $\forall c \in C, \emptyset \vdash c$. Outra forma de expressar isto é dizer que uma cds é uma teia de um espaço coerente se o conjunto de estados da cds é um espaço coerente.

Definem-se os *algoritmos lineares* como sendo estados de uma cds no estilo dos algoritmos seqüenciais do Curien ([CUR 86]). Em particular as cds consideradas são teias de espaços coerentes. Mostra-se como obter a cds $!A \multimap B$, a partir de uma função estável f: $A \to B$. O algoritmo linear desta cds possui todas as *estratégias de computação* (seqüenciais e paralelas) que computam a função subjacente f, o que implica que os algoritmos lineares podem ser considerados *meta-algoritmos*. Mostra-se que para toda estratégia de computação seqüencial de um algoritmo linear, existe um algoritmo seqüencial de Curien que computa a mesma função, e vice-versa.

A definição de estratégia de computação é dada de maneira tal que permite se dar semântica a segmentos de programas. Define-se uma operação de *composição* de estratégias, de forma tal que se pode obter uma estratégia de computação de um programa, a partir da composição das estratégias dos segmentos.

Palavras-chave: Espaços coerentes, Estruturas de Dados Concretas, Domínios Concretos, Estruturas de Eventos, Domínios de Eventos, Funções Lineares, Funções Estáveis, Funções Seqüenciais, Algoritmos Seqüenciais, Algoritmos Lineares, Seqüencialidade, Paralelismo, Semântica.

Title: "Sequential and Parallel Computational Strategies of Coherence Spaces"

ABSTRACT

The concrete data structures, or cds, $(C, V, E, \vdash)$ consists of a set C of cells, a set V of values, a set E of events and an enabling relation $\vdash$. The set of states of a cds is a concrete domain, that can be considered the "abstract" counterpart of the cds. In the same way we have that the events domains (that are more general that the concretes domains) are the abstract counterpart of the events structures. We show the relation between the concretes domains and events domains with the coherence spaces, as just as the relation between the cds and events structures with webs of coherence spaces. Intuitivelly, a cds is a web of a coherence space if any cell c is not enabled for any event, i.e. $\forall c \in C, \emptyset \vdash c$. We can say that a cds is a web of a coherence space if the set of states of the cds is a coherence space.

We define the *linear algorithms* as states of a cds following the Curien's sequential algorithms ([CUR 86]). In particular the cds considered are webs of coherence spaces. We show how to obtain a cds $!A \multimap B$ from a stable function $f{:}A \rightarrow B$. The linear algorithm of this cds contain all the *computational strategies* (sequentials and parallels) that compute the subjacent function f; this implies that the linear algorithms can be considered a kind of *meta-algorithms*. We show that for all sequential computational strategy of a linear algorithm exists a Curien's sequential algorithm that compute the same function and conversely.

We define the computational strategies in such a way that we can give semantic of segments of programs. We define a *composition* operation for strategies. This operation has the advantage that we can obtain the computational strategy of a program as the composition of the segments of it.

Keywords: Coherence Spaces, Concrete Data Structures, Concrete Domains, Events Structures, Events Domains, Linear Functions, Stables Functions, Sequential Functions, Sequential Algorithms, Linear Algorithms, Sequentiality, Parallelism, Semantic.

1 INTRODUÇÃO

Este capítulo tem quatro seções. Na primeira se apresenta um resumo histórico do porque da definição de tantos domínios diferentes, e as tentativas de obter uma semântica totalmente abstrata. Introduzem-se também os algoritmos lineares. Na segunda seção se consideram as motivações e objetivos do trabalho. Na seção seguinte se apresentam os trabalhos relacionados, e na última se mostra a organização do texto.

1.1 Domínios, Semântica Denotacional e Lógica Linear

Segundo Zhang em [ZHA 89], as linguagens de programação são as linguagens que executam computações. O alcance delas vai desde as linguagens mais teóricas como o cálculo-λ com várias estratégias de avaliação, até as linguagens usadas normalmente, como por exemplo Pascal, Lisp, C, etc. No desenvolvimento de linguagens de programação de alto nível, os programadores "aprenderam" a necessidade de dar um significado preciso à variedade de sintaxe. A aproximação mais conhecida deste problema é a de Scott e Strachey. Sua aproximação é baseada na idéia que a semântica de uma linguagem de programação deveria ser formalmente descrita em termos de um número menor de construções matemáticas em ordens parciais de informação. A parte matemática é chamada de teoria dos domínios e o método que usa a teoria dos domínios para especificar o significado de uma linguagem de programação é chamada de semântica denotacional.

A semântica denotacional de uma linguagem de programação é dada atribuindo a cada peça do programa, um elemento no domínio. Devido à possibilidade de um programa ser aplicado a si mesmo, os domínios devem ter, muitas vezes, algumas propriedades especiais. Tipicamente, pode se requerer que um domínio seja isomorfo a seu próprio espaço de funções. Isto é impossível com o "framework" clássico da teoria de conjuntos, pois a exponenciação sempre produz uma cardinalidade maior em conjuntos não triviais. A idéia de Scott foi trabalhar, não diretamente com valores e objetos de computação, senão com informação acerca da computação. Em seu "framework" de ordens parciais completas e funções contínuas, Scott mostrou que é possível obter soluções não triviais de equações tais como $D \cong [D \to D]$. As propriedades e construções em cpo's foram bem estudadas, ao ponto que atualmente já é possível resolver equações recursivas de domínios e usar essas construções para dar a semântica denotacional da maioria das linguagens de programação.

Uma ordem parcial completa (cpo) é uma ordem parcial que tem menor elemento (bottom) e para todo conjunto dirigido de elementos, existe o supremo (menor cota superior). Para dois elementos x e y, $x \leq y$ significa que a informação contida em x está contida na informação que está em y. O elemento bottom $\perp$, tem um conteúdo vazio de informação. Os elementos que têm um conteúdo finito de informação, têm um papel importante na teoria. Intuitivamente, estes elementos têm informação que podem ser processadas por uma computação em tempo finito. A estes elementos chama-se elementos finitos. Pelo que concerne às computações, usualmente é suficiente trabalhar com cpos ω-algébricos, que têm um conjunto de elementos enumerável na base. Uma função entre ordens parciais completas D e E, pode ser vista como uma computação que faz uso de alguma informação de entrada em D e produz alguma informação de saída em E. De acordo com a tese de Scott, tal função deveria ser contínua. A continuidade requer que a função seja monotônica - quanto mais informação na entrada, mais informação na saída - e, se $x_0 \leq x_1 \ldots \leq x_i \leq \ldots$ é uma cadeia em D, então a informação de saída para $f(\sup_i x_i)$ deve ser possível de ser aproximada pelos $f(x_i)$'s (tão próximo quanto seja requerido). Uma característica importante das cpos, é que a coleção de funções contínuas entre cpos é também uma cpo, com a ordem ponto a ponto; isto significa que as funções em si são associadas com elementos de informação e portanto habilita o tratamento de computações de alta ordem. Desafortunadamente, as funções contínuas entre cpos ω-algébricos não formam necessariamente uma cpo ω-algébrico. Entretanto, existem subcategorias de cpos ω-algébricos que têm esta característica. O "framework" mais conhecido são os domínios de Scott, aqueles cpos ω-algébricos que são consistentemente completos. Os domínios de Scott têm muitas propriedades úteis, em particular, eles são fechados sobre construções como a soma, produto, "lifting" e espaço de funções.

Há muitos "frameworks" para semântica denotacional, entre os quais se encontra a menos "standard", mas importante, categoria dos dI-domínios. Os dI-domínios foram definidos por Berry a partir do estudo do problema da abstração total para o cálculo-λ tipado. São um tipo especial de domínios de Scott que têm uma natureza operacional. As funções entre dI-domínios são as funções estáveis com uma ordem que tem em conta a maneira como são computadas. Os dI-domínios podem ser representados como estruturas de eventos de Winskel, que mostra sua intuição computacional. Podem ser representados também como sistemas de informação, que mostra o aspecto lógico dos dI-domínios. Os dI-domínios estão sendo cada vez mais populares, principalmente devido ao fato de possuírem uma estrutura mais elaborada, e devido ao fato que há outra categoria de dI-domínios descobertos por Winskel, onde podem ser usadas construções como o produto parcialmente sincrônico, para modelar melhor linguagens como CCS e CSP. A conexão desta categoria com outros modelos mais concretos para processos concorrentes, como redes de Petri, fazem com que sejam ainda mais atrativos. Uma subcategoria importante dos dI-domínios são os espaços coerentes, levado à popularidade por Girard. Os espaços coerentes foram criados para modelar o sistema F e para dar semântica da lógica linear.

Segundo [BRO 93] (pag. 80), os problemas da caracterização da seqüencialidade na teoria dos domínios surgiram a partir de trabalhos sobre fundamentação na semântica denotacional e operacional do cálculo-λ. Dana Scott introduziu LCF, uma lógica para funções computáveis baseada em uma lógica combinatória tipada simples. Scott desenvolveu um modelo no qual os tipos de dados são modelados como domínios e as frases de tipo funcional denotam funções contínuas. Questionou se era desejável ter funções intuitivamente paralelas na semântica, como por exemplo o "ou paralelo". Plotkin desenvolveu uma semântica operacional de uma variante do cálculo-λ da lógica combinatória de Scott, chamado de PCF ([PLO 77]), que deu o caráter essencialmente seqüencial da linguagem. Plotkin provou que o modelo de Scott não é totalmente abstrato ("fully abstract"). Isto é, que há termos com denotações diferentes que são iguais operacionalmente, no sentido que um pode ser substituído pelo outro em qualquer contexto de programa sem alterar o resultado da avaliação do programa.

O modelo de Milner [MIL 77] é totalmente abstrato (sintaticamente), mas deixa em aberto o problema de encontrar uma caracterização semântica de funções seqüenciais, o que implica na necessidade de uma noção de seqüencialidade na teoria dos domínios, que seja um refinamento da noção de continuidade. Mais o menos na mesma época, Gérard Berry introduziu a noção de estabilidade para estudar formas canônicas e computações ótimas para programas recursivos. Como afirma Brookes em [BRO 93], pag. 181, Berry mostrou que o cálculo-λ é sintaticamente estável e seqüencial, e introduziu um modelo de funções estáveis entre dI-domínios, usando a ordem estável ao invés da pontual. As funções estáveis e a ordem estável tem propriedades algébricas interessantes, como por exemplo o fato de que a categoria dos dI-domínios e funções estáveis é cartesiana fechada. Droste [DRO 93] estudou algumas das propriedades dos domínios estáveis. Entretanto que a noção de estabilidade refina a de continuidade de uma forma aparente, a classe das funções estáveis de Berry ainda contém algumas funções intuitivamente não seqüenciais. Parece não ter sido possível dar uma definição adequada de funções seqüenciais usando os domínios de Scott e dI-domínios, e portanto o problema de encontrar uma semântica satisfatória da noção de seqüencialidade permanece em aberto.

Em [KAH 93], Kahn e Plotkin propuseram o primeiro “framework” geral da teoria dos domínios no qual pode ser dada uma noção de computação incremental e funções seqüenciais. Começando com a idéia que os domínios deveriam ser (pelo menos) ordens parciais completas coerentes e ω-algébricas, foram introduzidas progressivamente suposições adicionais até chegar ao conceito de “domínios concretos”, que podem servir como modelos de dados, e serem equipados naturalmente com uma noção de computação incremental. Eles introduziram também as “matrizes de informação”, que são descrições abstratas de estruturas de dados, construídas a partir de células. Kahn e Plotkin apresentaram um teorema de representação estabelecendo que domínios concretos são a contrapartida, na teoria dos domínios, das “matrizes de informação”. As matrizes de informação foram chamadas depois, por Berry, de estruturas de dados concretas (cds). Em [KAH 93] é mostrado também que os domínios concretos são fechados com respeito ao produto cartesiano, soma separada e seções superiores, mas não com respeito à exponenciação. Kahn e Plotkin identificaram uma subclasse importante que consiste dos domínios concretos distributivos. Todo domínio concreto distributivo é um dI-domínio, e toda função seqüencial de Kahn-Plotkin entre domínios concretos distributivos é também estável. O contrário não é verdade pois há funções estáveis que não são seqüenciais.

Nielsen et al. [NIE 81] desenvolveram mais tarde, uma abordagem para modelar redes de Petri baseados em “domínios de eventos”, com uma representação concreta usando “estruturas de eventos”. Winskel desenvolveu uma teoria extensa de estruturas de eventos e domínios de eventos ([WIN 81]).

Berry e Curien ([BER 82]) introduziram uma teoria de algoritmos seqüenciais entre estruturas de dados concretas. Um algoritmo seqüencial pode ser visto como uma função seqüencial de Kahn-Plotkin junto com uma estratégia de computação (seqüencial) para essa função. A categoria de domínios concretos e algoritmos seqüenciais é cartesiana fechada (a categoria de domínios concretos com funções seqüenciais não é cartesiana fechada). O modelo de algoritmos de Berry e Curien é totalmente abstrato com respeito a uma forma intensional do comportamento de programas (no qual pode ser observado a estratégia de computação).

Bucciarelli ([BUC 93]), tentou resolver o problema da abstração total sem êxito, mas definiu um “framework” mais amplo e simples que a classe de domínios concretos, chamado de “estrutura seqüencial”. Uma estrutura seqüencial é dada por um domínio e em conjunto de propriedades lineares, ou observações, sobre o mesmo. As propriedades lineares tentam substituir as células. Tanto as estruturas de dados concretas como os espaços coerentes podem ser representados como estruturas seqüenciais, ainda mais, as estruturas de dados concretas são uma subcategoria fechada das estruturas seqüenciais com habilitação.

Zhang resume as relações entre varias categorias em [ZHA 89] (pag. 179). Entre elas, a categoria COH_S dos espaços coerentes e funções estáveis é uma subcategoria fechada completa da categoria SEV_S das estruturas de eventos estáveis com funções estáveis. Ainda mais, neste trabalho se dá uma caracterização da relação entre espaços coerentes, domínios concretos e domínios de eventos (e através dos teoremas de representação, entre teias de espaços coerentes, estruturas de eventos e estruturas de dados concretas). Zhang demonstra também que COH_L dos espaços coerentes e funções lineares não é cartesiana fechada (resultado também dado por Troelstra em [TRO 92], capítulo 10). O anterior sugere a idéia de definir “algoritmos lineares”, e estudar esta categoria (espaços coerentes e “algoritmos lineares”), tentando fazer o mesmo que Berry e Curien em [BER 82] com as estruturas de dados concretas e funções seqüenciais, e que resultou na categoria dos “algoritmos seqüenciais”.

Desde sua concepção ([GIR 87]), a lógica linear tem sido promissora como formalismo para servir como interface entre lógica e a ciência da computação.

Abramsky ([ABR 93]) dá a interpretação computacional da lógica linear e salienta que, do ponto de vista lógico, a lógica linear combina a simetria da lógica clássica, com o conteúdo construtivo da lógica intuicionística. Entretanto, do ponto de vista computacional, ela oferece uma perspectiva lógica de resultados computacionais, tais como controle de recursos e ordem de avaliação. O conceito mais importante (para o desenvolvimento do trabalho) talvez seja o de "linearização" de uma função estável (ver [GIR 89], pag. 101), onde a implicação intuicionística ($\Rightarrow$) pode ser expressa como desdobramento em dois conectivos da lógica linear: o operador "of course"(denotado pelo símbolo "!") e a implicação linear ($\multimap$), i.e. $A \Rightarrow B = !A \multimap B$. A igualdade anterior pode ser lida como: dada uma função estável de um espaço coerente A em outro B, a função do espaço coerente $!A$ em B é linear (de forma análoga, dada uma função linear é possível definir uma função estável). A igualdade anterior dá, claramente, uma forma de "linearizar" uma função estável.

Neste trabalho se definem os "algoritmos lineares", como estados de uma cds $!A \multimap B$, seguindo a definição de Curien dos algoritmos seqüenciais. Se bem que, são muito parecidos na forma, com os algoritmos seqüenciais, os algoritmos lineares (em particular as estratégias de computação) são mais gerais, pois permitem ler e escrever mais de uma célula por cada operação. Por isto poderia se dizer que os primeiros estão incluídos nos segundos. Isto é, dada uma função estável, se obtém a função linearizada (aplicando o operador "of course"), e sobre esse novo domínio se define a cds $!A \multimap B$. Um estado desta cds é um conjunto de eventos que contém todas as estratégias de computação (seqüenciais e paralelas) para a função estável original. Deste ponto de vista, um algoritmo linear seria uma espécie de *meta-algoritmo* ou *intérprete*.

As estratégias de computação são definidas de forma a ser possível classificá-las em seqüenciais e paralelas, e em parciais e totais. Esta última classificação permite dar semântica a segmentos de programas, obtendo assim uma semântica composicional: a semântica de um programa pode ser visto como a composição das semânticas dos seus segmentos. Os algoritmos lineares, definidos como partes de cds, dão uma semântica denotacional das funções estáveis, mas podem ser considerados também como uma semântica operacional, pois eles explicitam, operacionalmente, como as computações são realizadas.

1.2 O problema proposto

1.2.1 Motivação

A motivação original deste trabalho foi estudar a computabilidade dos espaços coerentes, tendo em mente a possibilidade de representar a Matemática Intervalar neste domínio, já que, aparentemente, daria uma representação alternativa mais simples que outras anteriormente definidas (ver [ACI 89], [ACI 93], [CLA 94], [CLA 94a], [DIM 91] e [DIM 95]). Havia, em princípio, duas possibilidades: o ponto de vista “topológico”, e o ponto de vista de “programação”. O primeiro enfoque implicava definir uma topologia nos espaços coerentes, obviamente tendo que definir conceitos como continuidade, conjuntos abertos, etc. A outra aproximação (a seguida neste trabalho) foi mais “computacional”: as funções no espaço coerente são computáveis, se existe um algoritmo que as computa, seguindo a definição dos algoritmos seqüenciais do Curien. Esta última aproximação derivou na definição dos *algoritmos lineares*.

Na medida que o trabalho foi evoluindo, a própria definição de algoritmo linear foi tomando importância, justificando o estudo dos mesmos, em detrimento da sua aplicação original (no caso, a representação da Matemática Intervalar nos espaços coerentes).

1.2.2 Objetivo

Estudar os espaços coerentes, e definir um conceito similar aos algoritmos seqüenciais do Curien (ver [CUR 86]) nos espaços coerentes, considerando funções estáveis, e não somente as seqüenciais.

1.3 Trabalhos relacionados

Na seção 1.1 se introduziu uma história resumida dos domínios em computação assim como também da lógica linear, que estão, direta ou indiretamente relacionados com este trabalho. Considerando a definição dos Algoritmos Lineares como o ponto principal deste trabalho, é inegável que os trabalhos de Curien e Berry

([CUR 86] e [BER 82]) são fundamentais. Estes trabalhos tiveram como motivação a abstração total da linguagem PCF (ver [PLO 77], [MIL 77] e [BUC 93]).

Uma boa introdução à semântica de linguagens é dada em [WAT 91]. Acerca de domínios existem muitos trabalhos (por exemplo [SMY 83], [ZHA 95], [DRO 93]), mas em particular os domínios concretos ([KHA 93]) e domínios de eventos (ver [NIE 81], [DRO 89] e [PIN 95]) são importantes neste trabalho pela relação com as cds. Smyth ([SMY 77]), Asperti ([ASP 90]) e Constable ([CON 93]) têm estudado a computabilidade nos domínios. [COS 94] apresenta uma interpretação operacional da lógica linear em termos de fluxo de recursos.

Os espaços coerentes foram introduzidos por Girard ([GIR 87], [GIR 89], [TRO 92]) para dar semântica da Lógica Linear ([WAD 91], [TRO 92a], [SCE 90]). Em [DIM 95], Dimuro representa a Matemática Intervalar nos Espaços Coerentes. Y. Lafont definiu uma máquina abstrata linear ([LAF 88]) para programação funcional. Abramsky ([ABR 93]) apresenta uma interpretação computacional da Lógica Linear, e Lincoln et al., em [LIN 93], mostra como linearizar a implicação intuicionística. Uma linguagem de programação lógica e apresentada em [HAR 95]. Zhang ([ZHA 89]), na sua tese de doutorado relaciona os espaços coerentes (como categoria) com outras categorias de domínios.

1.4 Organização do texto

No capítulo 2 se apresentam alguns resultados básicos da teoria dos domínios, considerados necessários para a compreensão dos capítulos seguintes, assim como o conceito de domínio concreto. A fonte principal dos domínios concretos é [KHA 93]. Sobre ordens parciais, reticulados, teoria dos domínios se pode consultar [STO 94] e [DAV 90]. Alguns resultados sobre os domínios concretos e cds podem ser encontrados em [BER 82] e [CUR 86].

O terceiro capítulo trata das estruturas de dados concretas, que são a parte "concreta" dos domínios concretos, assim como também dos algoritmos seqüenciais de Curien. Este capítulo está baseado fundamentalmente em [CUR 86] e [BER 82].

O capítulo 4 introduz os espaços coerentes de Girard, as funções estáveis e lineares sobre os mesmos. Mostra-se como linearizar uma função estável através do operador “of course”, se apresentam vários exemplos e se dá uma interpretação computacional deste operador. O conteúdo deste capítulo está baseado principalmente em [GIR 89] e [TRO 92].

Os capítulos 5 e 6 são a contribuição deste trabalho. No capítulo 5 se relacionam os espaços coerentes com os domínios concretos e domínios de eventos, assim como também as teias de espaços coerentes com as estruturas de dados concretas e estruturas de eventos. O capítulo 6 apresenta os *algoritmos lineares* e as *estratégias de computação*, que são os conceitos principais deste trabalho. A definição dos algoritmos lineares lembra os algoritmos seqüenciais de Curien, exceto em alguns detalhes, cuja diferença fundamental se encontra no domínio de entrada e na definição das operações. Os algoritmos lineares podem ser vistos como *meta-algoritmos*, entretanto que as estratégias de computação são realmente os algoritmos (ou programas) que computam uma função dada.

Por último, no capítulo 7 se apresentam as conclusões assim como sugestões de trabalhos futuros.

2 CONCEITOS BÁSICOS

Neste capítulo se apresentam alguns conceitos preliminares sobre teoria dos domínios, e domínios concretos. O propósito da teoria dos domínios concretos é encontrar um "framework" satisfatório para as noções de co-rotina de computação e seqüencialidade de avaliação. Este capítulo é baseado principalmente em [KHA 93], onde também se podem encontrar as provas das proposições apresentadas.

2.1 Domínios de computação

O leitor familiarizado com conceitos introdutórios da Teoria dos Domínios pode dispensar a leitura desta seção.

2.1.1 Definição (ordem parcial)

Uma *ordem parcial* é um par $(D, \leq)$, onde D é um conjunto não vazio e $\leq$ é uma relação binária que satisfaz:

(i) $\forall x \in D$, $x \leq x$ (reflexividade)

(ii) $\forall x, y \in D$, $x \leq y$, $y \leq x \Rightarrow x = y$ (antissimetria)

(iii) $\forall x, y, z \in D$, $x \leq y$, $y \leq z \Rightarrow x \leq z$ (transitividade). ■

2.1.2 Definição (elementos comparáveis)

Dois elementos são *comparáveis* quando, ou $x \leq y$, ou $y \leq x$. Quando isto não acontece, então os elementos são *incomparáveis*, e esta relação é escrita como $x \parallel y$.

Uma ordem parcial na qual quaisquer dois elementos são comparáveis, é denominada *cadeia*. ■

2.1.3 Definição (supremo e ínfimo)

Seja $(D, \leq)$ uma ordem parcial, H um subconjunto de D e x um elemento de D. x é uma *cota superior* de H se e somente se $\forall y \in H$, $y \leq x$, e x é uma *cota inferior* de H se e somente se $\forall y \in H$, $x \leq y$. O elemento x é a *menor cota superior* (ou *supremo*) de H se e somente se é uma cota superior de H e $\forall z$ cota superior de H, $x \leq z$. Similarmente, x é a *maior cota inferior* (ou *ínfimo*) de H se e somente se é uma cota inferior de H e $\forall z$ cota inferior de H, $z \leq x$. Um elemento x em H é um *máximo* se e somente se x é o supremo de H, e é o *mínimo* se e somente se é o ínfimo de H. ■

Denotar-se-á o supremo (ínfimo) x de H por $x = \sup(H)$ ($x = \inf(H)$) ou $x = \bigcup H$ ($x = \bigcap H$). Se $H = \{a, b\}$, as notações são $x = a \vee b$ e $x = a \wedge b$ respectivamente (se a e b são conjuntos, pode-se escrever também $x = a \cup b$ e $x = a \cap b$).

2.1.4 Definição (elementos compatíveis)

Dois elementos x e y de D são *compatíveis* se $\{x, y\}$ tem cota superior. Esta relação é denotada por $x \uparrow y$, e seu complemento, a *relação de incompatibilidade*, é escrita como $x \# y$. ■

2.1.5 Definição (conjunto dirigido)

Em uma ordem parcial $(D, \leq)$, um subconjunto X de D é *dirigido* se e somente se X é não vazio e

$\forall x, y \in X$, $\exists z \in X$ tal que $x \leq z$, $y \leq z$. ■

Pode observar-se que, por definição, todo subconjunto que é uma cadeia, é dirigido.

2.1.6 Definição (ordem parcial completa)

Uma ordem parcial $(D, \leq)$ é *completa* (ou *cpo*) se e somente se

1. D tem um elemento mínimo $\perp$,

2. qualquer subconjunto dirigido X de D tem supremo. ■

2.1.7 Definição (ordem parcial condicionalmente completa)

Uma ordem parcial $(D, \leq)$ é *condicionalmente completa* se e somente se qualquer subconjunto X de D que tem uma cota superior, tem supremo. ■

Observações: (i) Já que D é não vazio, o subconjunto vazio $\varnothing$ tem uma cota superior. Portanto, se $(D, \leq)$ é condicionalmente completo, D deve ter um elemento mínimo $\perp = \bigcup\varnothing$.

(ii) A terminologia utilizada aqui, apesar de ser usual, pode não ser a ideal, já que uma ordem parcial pode ser completa sem ser condicionalmente completa.

2.1.8 Proposição

Uma ordem parcial completa (cpo) $(D, \leq)$ é condicionalmente completa se e somente se todo par de elementos compatíveis (x, y) tem supremo $x \vee y$. ■

2.1.9 Definição (conjunto consistente)

Em uma ordem parcial $(D, \leq)$, um subconjunto X de D é *consistente* se e somente se quaisquer dois elementos em X são compatíveis. ■

2.1.10 Definição (ordem parcial consistentemente completa)

Uma ordem parcial $(D, \leq)$ é *consistentemente completa*[1] se e somente se qualquer subconjunto consistente X de D tem supremo. ■

Observações: (i) Um subconjunto que tem cota superior é consistente. Portanto, se uma ordem parcial é consistentemente completa, é conseqüentemente condicionalmente completo.

---

[1]Em [KHA 93] esta propriedade é chamada de *coerente*. Preferiu-se utilizar a expressão *consistentemente completa* (amplamente utilizada por muitos autores) para evitar ambigüidade com a definição de coerência sobre espaços coerentes de Girard.

(ii) O conjunto vazio $\varnothing$ é consistente. Tem como supremo o elemento $\perp$. Um conjunto dirigido é consistente. Portanto, se uma ordem parcial é consistentemente completa, é também completo.

2.1.11 Proposição

Uma cpo $(D, \leq)$ é consistentemente completa se e somente se qualquer tripla consistente $\{x, y, z\}$ tem supremo. ■

2.1.12 Definição (elemento compacto)

Em uma ordem parcial $(D, \leq)$, um elemento x é *isolado* (ou *compacto*) se e somente se, em qualquer conjunto dirigido cujo supremo domina[2] x, se pode encontrar um elemento y que domina x. Isto é:

$$\forall X \subset D,\ X \text{ dirigido e } x \leq \bigcup X \Rightarrow \exists y \in X,\ x \leq y. \ ■$$

Notação: O conjunto de elementos isolados menores (em relação à ordem $\leq$) que x é denotado $\mathcal{A}(x)$. Um elemento em $\mathcal{A}(x)$ é chamado uma *aproximação* de x. O conjunto de todos os elementos isolados em $(D, \leq)$ se escreve $\mathcal{A}(D)$, ou D^0. Um elemento x é isolado se e somente se $x \in \mathcal{A}(x)$, portanto $\mathcal{A}(D) = \bigcup_{x \in D} \mathcal{A}(x)$.

2.1.13 Proposição

Seja uma ordem parcial condicionalmente completa $(D, \leq)$, então:

1. Se dois elementos isolados a e b são compatíveis, então $a \vee b$ é isolado.

2. Para todo x, o conjunto $\mathcal{A}(x)$ é dirigido. ■

2.1.14 Definição (ordem parcial algébrica)

Uma ordem parcial $(D, \leq)$ é *algébrica* se e somente se para todo x em D, o conjunto $\mathcal{A}(x)$ é dirigido, e

---

[2]Diz-se que um elemento x *domina* y em uma ordem parcial $(D, \leq)$ se $y \leq x$.

$x = \bigcup \mathcal{A}(x)$.

Se $\mathcal{A}(D)$ é enumerável, então $(D, \leq)$ é *ω-algébrico*. ■

A seguir apresenta-se a definição mais importante desta seção, a de *domínio de computação*.

2.1.15 Definição (domínio de computação)

Uma ordem parcial consistentemente completa e ω-algébrica é chamada de *domínio de computação*.[3] ■

Notação: A partir de agora, ao referir-se a uma ordem parcial $(D, \leq)$, se não houver confusão, simplesmente se escreverá D.

As seguintes proposições são importantes para mostrar que a categoria dos domínios de computação, que tem como objetos domínios de computação e como morfismos as funções contínuas, é uma categoria cartesiana fechada.

2.1.16 Proposição

O produto cartesiano de um número contável de domínios de computação é um domínio de computação. ■

2.1.17 Proposição

Se D e E são domínios de computação, o conjunto $[D \to E]$ de todas as funções contínuas de D em E, junto com a ordem natural, é um domínio de computação. ■

Esta proposição permite que, partindo de domínios de computação, se construa uma hierarquia de domínios de computação, tais como $[D \to E]$, $[D \to [D \to E]]$, $[[D \to E] \to [D \to E]]$, etc. Considerando os domínios de computação $[D \to E]$ como sendo a interpretação semântica das funções de uma linguagem, a hierarquia

[3] Alguns autores definem *domínio* como sendo uma cpo algébrica, e *domínios de Scott* como sendo cpo's ω-algébricas consistentemente completas.

anterior permitiria fornecer uma semântica para linguagens em que funções têm como argumento funções.

2.2 Domínios concretos

Os domínios concretos foram introduzidos por Khan & Plotkin (ver [KHA 93]) com a intenção de obter uma representação capaz de diferenciar os dados das funções de uma computação (no trabalho de Scott não existe tal diferença de representação). No processo da caracterização axiomática desta classe de domínios, se seguirão dois princípios fundamentais:

1. (M. Smyth) Todos os axiomas postulados, especificam propriedades dos elementos isolados em domínios de computação. Outros elementos são construídos a partir dos elementos isolados por um mecanismo de *limites*; suas propriedades serão deduzidas das propriedades dos elementos isolados.

2. A classe dos domínios de computação que será definida, deve ser fechada para certas construções elementares, tais como produtos cartesianos finitos e infinitos ou seções superiores ("upper sections"). Entretanto, não precisa ser fechada para a *exponenciação*, isto é, para a construção do espaço de funções.[4]

2.2.1 Os elementos isolados

Os elementos isolados são quantidades finitas de informação em domínios de computação. Ao trabalhar com dados, seria desejável raciocinar por indução nestes elementos. Isto implicaria que os elementos isolados deveriam ser bem-fundados com respeito à relação $\leq$, isto é, não deveria existir uma cadeia infinita $\{x_1, x_2,..., x_n,...\}$, com $x_1 > x_2 >... > x_n >...$. Desta forma, um elemento isolado não pode ser decomposto indefinidamente, e por isto a condição (F).

[4]Na definição dos domínios concretos, Khan & Plotkin não se preocuparam com o espaço funcional pois o seu objetivo era criar justamente um domínio de dados, e não de funções. Esta é a razão pela qual os domínios não tem que ser necessariamente fechados com respeito à exponenciação.

2.2.1.1 Propriedade (F)

Entre dois elementos isolados comparáveis diferentes, qualquer cadeia de elementos isolados é finita. ■

A propriedade (F)[5] pode ser expressa como:

(F) $\forall x \in D^0$, $\{y \in D \mid y \leq x\}$ é finito.

Segundo [ZHA 89] (pag. 150) a propriedade (F) é também necessária devido ao fato de que as funções estáveis entre domínios de Scott distributivos ω-algébricos, com a ordem estável, não necessariamente formam um domínio ω-algébrico. Portanto podem existir muitos elementos finitos no espaço de funções.

2.2.1.2 Definição (ideal)

Em uma ordem parcial condicionalmente completa (D, $\leq$), um *ideal* é um subconjunto não vazio J de D tal que

1. $\forall x \in J$, $\forall y \in D$, $y \leq x \Rightarrow y \in J$ (i.e., J é fechado inferiormente)

2. $\forall x,y \in J$, $y \uparrow x \Rightarrow x \vee y \in J$. ■

Em um domínio de computação, a propriedade (F) é equivalente às seguintes propriedades:

2.2.1.3 Propriedade (F_1)

Entre dois elementos isolados comparáveis, qualquer cadeia é finita. ■

2.2.1.4 Propriedade (F_2)

O conjunto de elementos isolados é um ideal bem-fundado. ■

[5]Esta propriedade é chamada (I) em [KHA 93] e [ZHA 89], mas neste trabalho segue-se a notação de Curien para evitar ambigüidade com a propriedade (I) dos domínios de eventos de Droste.

2.2.1.5 Definição (relação de cobertura)

Sejam (D, $\leq$) uma ordem parcial, e x e y dois elementos de D. Diz-se que y *cobre* x, denota-se $x \prec y$, se e somente se

1. $x < y$,

2. $\forall z,\ x < z \leq y \Rightarrow z = y$. ■

2.2.1.6 Definição (átomo)

Seja (D, $\leq$) uma ordem parcial com menor elemento $\perp$. Um *átomo* é um elemento $y \in D$ que cobre $\perp$, e diz-se que D é *atômico* se e somente se qualquer elemento $x \in D$, diferente de $\perp$ domina um átomo. Isto é, tem-se que:

$$\forall x \neq \perp, \exists y \text{ tal que } \perp \prec y \leq x. \blacksquare$$

2.2.1.7 Proposição

Um domínio de computação com a propriedade (F) é atômico. ■

A propriedade (F) e suas derivadas (F_1) e (F_2), assim como a atomicidade, são propriedades muito úteis para domínios de computação.

2.2.1.8 Definição (seção superior)

Seja (D, $\leq$) uma ordem parcial e dois elementos x e y em D, tal que $x \leq y$. O *intervalo* $[x, y]$ é o conjunto $\{z \mid x \leq z \leq y\}$ e a *seção superior* de x, denotada por $[x)$, é o conjunto $\{z \mid x \leq z\}$. Logicamente, intervalos e seções superiores herdam a ordem parcial $\leq$. ■

2.2.1.9 Proposição

Intervalos e seções superiores de domínios de computação são domínios de computação. ■

Se um domínio de computação tem a propriedade (F), não necessariamente suas seções superiores tem esta propriedade (ver a figura 4(a) da página 204 de [KHA

93]). É preciso, portanto, uma propriedade mais forte que a propriedade (F) para preservar essa propriedade.

2.2.2 A relação de cobertura

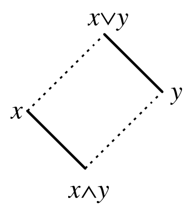
Os elementos isolados de $([x), \leq)$ têm, exceto para x, a forma $x \vee d$, com d isolado, compatível e incomparável com x. A seguinte propriedade postula uma caracterização similar dos átomos em uma seção superior.

2.2.2.1 Propriedade (C)

Se x e y são elementos isolados compatíveis, então

$$x \wedge y \prec x \Rightarrow y \prec x \vee y. \blacksquare$$

A propriedade (C) fica bem caracterizada pelo diagrama abaixo:



Enquanto a propriedade (F) não exclui nenhum domínio finito, a propriedade (C) sim. Assim, por exemplo, alguns dos domínios da Figura 2.1 não cumprem a propriedade (C). A ordem parcial na Figura 2.1(a) não é condicionalmente completa, a ordem parcial da Figura 2.1(b) não é consistente. As ordens parciais da Figura 2.1(c) e (d) não satisfazem a propriedade (C).

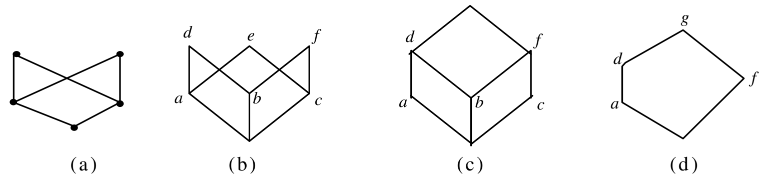


Figura 2.1 - Cpo's que falham em ser domínios com a prop. (C)

Dado um domínio de computação que tem a propriedade (F), a propriedade (C) é equivalente às seguintes propriedades:

2.2.2.2 Propriedade (C_0)

Se x e y são dois elementos compatíveis, então

$$x \wedge y \prec x \Rightarrow y \prec x \vee y. \blacksquare$$

2.2.2.3 Propriedade (C_1)

Se x e y são dois elementos compatíveis diferentes, então

$$\exists z, z \prec x, z \prec y \Rightarrow x \prec x \vee y, y \prec x \vee y. \blacksquare$$

2.2.2.4 Corolário

Em um domínio de computação D que satisfaz as propriedades (F) e (C), qualquer seção superior (e qualquer intervalo) é atômico. ■

2.2.2.5 Teorema

Qualquer seção superior $[x)$ e qualquer intervalo $[x, y]$ em um domínio de computação que satisfaz as propriedades (F) e (C), é um domínio de computação que também satisfaz estas propriedades. ■

2.2.3 A relação de incompatibilidade

As propriedades (C) e (F) concernem unicamente a estruturas de sub-reticulados em domínios de computação. É necessário examinar mais cuidadosamente a relação de incompatibilidade.

Introduz-se agora uma nova propriedade que restringe a forma como a incompatibilidade deve aparecer.

2.2.3.1 Propriedade (Q)

Se x e y são dois elementos isolados incompatíveis, então

$$x \wedge y \prec x \Rightarrow \exists! t,\ t \# x,\ x \wedge y \prec t \leq y. \blacksquare$$

Muitos domínios simples não apresentam esta propriedade, como por exemplo os domínios da Figura 2.2. O domínio da Figura 2.2(a) não tem a propriedade (Q) pois c e a são dois elementos incompatíveis tal que $c \wedge a \prec c$, mas não existe um elemento t incompatível com c tal que $c \wedge a \prec t \leq a$. No domínio da Figura 2.2(b) considerem-se os elemento incompatíveis c e a. Novamente tem-se que $c \wedge a \prec c$, mas não um elemento t incompatível com c tal que $c \wedge a \prec t \leq a$.

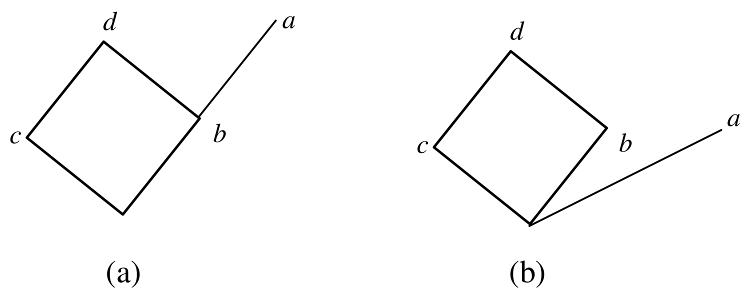


Figura 2.2 - Domínios sem a propriedade (Q)

Em um domínio de computação que cumpre as propriedades (F) e (C), a propriedade (Q) é equivalente à propriedade (Q_0) definida a seguir.

2.2.3.2 Propriedade (Q_0)

Se x e y são dois elementos incompatíveis, então

$$x \wedge y \prec x \Rightarrow \exists! t,\ t \# x,\ x \wedge y \prec t \leq y. \blacksquare$$

2.2.3.3 Corolário

Em um domínio de computação que satisfaz as propriedades (F), (C) e (Q), uma seção superior também satisfaz estas propriedades. ■

2.2.4 A relação de projetividade

Nesta seção se introduz a noção de *projetividade*, com o fim de caracterizar a propriedade (R), mas se segue [CUR 86] ao invés de [KHA 93] por simplicidade.

2.2.4.1 Definição (cadeia de cobertura, intervalo primo, zigzag, equivalência)

Seja (D, $\leq$) uma ordem parcial, então

- uma *cadeia de cobertura* (ou simplesmente *cadeia*) de x até y é uma seqüência $x = x_0,..., x_n = y$, denotada por $x_0 \prec ... \prec x_n$, tal que $\forall 0\leq i<n$, $x_i \prec x_{i+1}$; se C é uma cadeia $x_0 \prec ... \prec x_n$ de x até y, então $x' \prec x_0 \prec ... \prec x_n$ é denotado por $x' \prec C$ (assim como $C \prec y'$ se $y \prec y'$)

- um *intervalo primo* de D, denotado por $[x, x']$, é tal que x, x' são elementos isolados e $x \prec x'$; (ou seja, um intervalo primo de x em x' contém somente x e x')

- definimos a relação $\prec$ de *cobertura* entre intervalos primos como

$$[x, x'] \prec [y, y'] \text{ sss } x \prec y,\ y \neq x' \text{ e } x' \prec y'$$

- se s, t são elementos (intervalos primos) de D, chamamos *zigzag* de s até t, uma seqüência $s = s_0,..., s_n = t$ de elementos (intervalos primos) de D tal que

$$\forall i<n,\ s_i \prec s_{i+1} \text{ ou } s_{i+1} \prec s_i$$

- chamamos *equivalência* (denotada por $\succ\!\prec$) o fecho reflexivo, simétrico e transitivo da relação $\prec$ em intervalos primos, isto é, $[x, x'] \succ\!\prec [y, y']$ se e somente se existe um zigzag de $[x,x']$ em $[y, y']$; a classe de equivalência de $[x, x']$ é denotada por $[x, x']_{\succ\prec}$. Dois intervalos $[x, x']$ e $[y, y']$ tais que $[x, x'] \succ\!\prec [y, y']$ são chamados *equivalentes* ou *projetivos*. ■

Segundo [KHA 93] (pag. 224), se dois intervalos são projetivos, deveriam representar o mesmo incremento elementar de informação (possivelmente tendo lugar em dois estados globais diferentes). Uma *decisão elementar*, ou simplesmente, uma *decisão*, é uma classe de equivalência de intervalos primos. Mas, tal interpretação de projetividade apresenta uma inconsistência, que pode ser eliminada postulando uma propriedade adicional (a propriedade (R)). Seja a ordem parcial da seguinte figura:

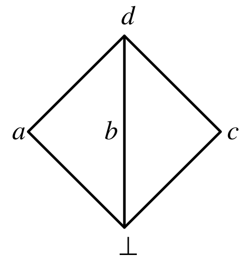


Figura 2.3 - Domínio que não tem a propriedade (R)

que não tem a seguinte propriedade:

2.2.4.2 Propriedade (R)

Se $[x, y]$ e $[x, z]$ são dois intervalos projetivos primos com extremos isolados, então $y = z$. Em símbolos:

$$\forall x,y,z \in D^0,\ [x, y] \succ\!\!\prec [x, z] \Rightarrow y = z. \blacksquare$$

Esta propriedade pode ser expressa como: se x é um elemento isolado, então dois incrementos elementares diferentes a partir de x são duas decisões diferentes.

Em um domínio de computação que satisfaz as propriedades (F) e (C), a propriedade (R) é equivalente à seguinte:

2.2.4.3 Propriedade (R_0)

Se $[x, y]$ e $[x, z]$ são dois intervalos projetivos primos, então $y = z$. ■

2.2.4.4 Corolário

Em um domínio de computação que satisfaz as propriedades (F), (C) e (Q) e (R), qualquer seção superior (e qualquer intervalo) também satisfaz estas propriedades. ■

Se um domínio de computação é um reticulado que satisfaz as propriedades (F) e (C), a propriedade (R) é equivalente à seguinte:

2.2.4.5 Propriedade (R_T)

Nenhum subreticulado é isomorfo ao retículo da Figura 2.3. ■

Tendo definidas as propriedades anteriores, pode-se definir os *domínios concretos*.

2.2.4.6 Definição (domínio concreto)

Um *domínio concreto* é uma cpo ω-algébrica $(D, \leq)$ que satisfaz os seguintes axiomas para quaisquer elementos isolados (compactos) $x, x', y, y' \in D$:

(F) $\{y \in D \mid y \leq x\}$ é finito;
(C) se $x \prec y$, $x \prec z$, $z \neq y$ e $y \uparrow z$, então existe $y \vee z$, e $y \prec y \vee z$, $z \prec y \vee z$;
(R) $[x, y] \succ\!\prec [x, z] \Rightarrow y = z$.
(Q) se z, x, y são elementos isolados tais que $z \prec x$, $z \prec y$ e $x \# y$, então existe uma única x' tal que $z \prec x' \leq y$ e $x \# x'$. ■

A definição de domínio concreto dada aqui é a do Curien (ver [CUR 86]) e não a de [KHA 93], por consistência com o capítulo seguinte.

Neste capítulo foram definidas as propriedades essenciais para que um domínio de computação seja considerado um domínio de dados ao invés de um domínio funcional. Em [KHA 93] é definida a *matriz de informação*, que Curien denomina de *estrutura de dados concreta* (cds), que é a parte concreta dos domínios concretos. Este conceito assim como outros relacionados, serão estudados no capítulo seguinte.

3 ALGORITMOS SEQÜENCIAIS EM ESTRUTURAS DE DADOS CONCRETAS

Neste capítulo se apresenta a teoria das estruturas de dados concretas (ou cds, "concrete data structures") e algoritmos seqüenciais. As estruturas de dados concretas têm sido introduzidas, entre outras razões, para dar uma definição semântica à seqüencialidade.

3.1 Introdução

Assim como máquinas manipulam representações de números, e não números, se trabalhará com representações concretas de conjuntos e funções e não com conjuntos e funções diretamente. A motivação da construção dos algoritmos seqüenciais foi, segundo o próprio autor (Curien), garantir que duas sentenças de programas tenham o mesmo significado, se têm o mesmo comportamento no computador (é o problema conhecido como "*full abstraction*", que poderia traduzir-se como *abstração total*). Restringindo este problema às linguagens de programação seqüenciais, se tem que ter em conta a definição semântica das funções seqüenciais. A idéia é substituir conjuntos e funções por descrições de conjuntos e funções: *estruturas de dados concretas* (cds) e *algoritmos seqüenciais*. Estes objetos são de natureza sintática, e levam naturalmente à definição de uma linguagem de programação chamada CDS0 (que não será vista neste trabalho), onde o problema da abstração total é resolvido.

As cds podem ser vistas como especificações de conjuntos, cujos elementos podem ser construídos por partes, preenchendo células, ou lugares, com valores, o que lembra os *records* de muitas linguagens de programação imperativas. Um evento é um par formado por uma célula e um valor, e os elementos destas estruturas são conjuntos de eventos permitidos pela especificação. As funções foram substituídas por especificações de possíveis formas de computá-las seqüencialmente: os *algoritmos seqüenciais*, construídos por partes, lendo a entrada também por partes. A divisão da informação em partes discretas (ou eventos que consistem de células com valor), dá uma boa descrição da seqüencialidade: funções seqüenciais são funções f tais que para todo dado de entrada x, e qualquer célula c' da estrutura de saída na qual queremos incrementar a informação de saída $f(x)$, existe pelo menos uma célula c na estrutura de entrada, que deve ser preenchida. Os algoritmos seqüenciais escolhem computar tais células "necessárias", e já que existem muitas delas, haverá muitos algoritmos que computam a função dada.

Na seção 2 se introduzem as cds. Na seção 3 se relacionam as cds com estruturas abstratas: os domínios concretos. Na seção 4 se apresentam as funções seqüenciais, e na seção 5 se vem os algoritmos seqüenciais.

Os modelos usuais de conjuntos e funções (na realidade, cpo's e funções contínuas, para poder manipular recursão, segundo D. Scott), satisfazem somente a metade da propriedade de abstração total: se duas expressões têm a mesma denotação, então têm o mesmo comportamento. O ponto é que a semântica usual faz uma diferenciação sutil entre objetos. A situação típica (mostrada por G. Plotkin) é aquela na qual duas funções diferem em um argumento que nunca será o valor de uma expressão, mas que são denotados por expressões que têm o mesmo comportamento. Tal argumento é o "ou paralelo". Portanto a idéia é construir modelos que excluam o "ou paralelo" da semântica. Seria desejável construir também um modelo que contenha somente objetos seqüenciais. Berry construiu uma semântica intermediária que chamou de "estável", mas forte suficiente para excluir o "ou paralelo".

3.2 Estruturas de dados concretas

Define-se as estruturas de dados concretas (cds) e seus estados, que formam uma ordem parcial completa com a inclusão como ordem.

3.2.1 Definição (estruturas de dados concretas (cds))

Uma *estrutura de dados concreta* (cds) $(C, V, E, \vdash)$ é dado por três conjuntos C, V, E de *células*, de *valores* e de *eventos* respectivamente, tal que

$E \subseteq C \times V$ e $\forall c \in C, \exists v \in V$ tal que $(c, v) \in E$ ("qualquer célula pode ser preenchida"),

e a relação $\vdash$, chamada de *relação de acessibilidade* entre partes finitas de E e elementos de C. Diz-se que $\{e_1,..., e_n\}$é uma *habilitação* de c se $\{e_1,..., e_n\} \vdash c$, e se escreve simplesmente $e_1,..., e_n \vdash c$. Uma célula c tal que $\vdash c$, é chamada *inicial*. C e V são considerados contáveis. ■

3.2.2 Definição (estado de uma cds)

Um *estado* é um subconjunto x de E tal que

(1) $(c, v_1), (c, v_2) \in x \Rightarrow v_1 = v_2$

(2) se $(c, v) \in x$, então existe uma seqüência de eventos $e_1,..., e_n = (c, v)$ tal que $e_i = (c_i, v_i) \in x$ e $\{ e_i \mid j < i\}$contêm uma habilitação de c_i para todo $i \leq n$.

As condições (1) e (2) são chamadas de *consistência* e *segurança* respectivamente. A seqüência descrita na condição (2) é uma *dedução*.

O conjunto de estados de uma cds M ordenadas por inclusão é uma ordem parcial denotada por $(D(M), \leq)$ (ou $(D(M), \subseteq)$). Se D é isomorfo com $D(M)$, dizemos que M *gera* ou *representa* D. ■

Escrever-se-á $M = (C_M, V_M, E_M, \vdash_M)$, ou simplesmente $M = (C, V, E, \vdash)$, $M' = (C', V', E', \vdash')$, ..., quando não der lugar a confusão.

3.2.3 Proposição

Seja M uma cds. Então $(D(M), \leq)$ é uma cpo ω-algébrica e coerente, onde o elemento mínimo é o estado vazio, e o supremo é dado pela união de conjuntos.■

Portanto os supremos (sup) podem ser denotados indiferentemente por $\vee$ ou $\cup$.

3.2.4 Definição (célula preenchida, habilitada e acessível)

Seja x um conjunto de eventos de uma cds. Uma célula c é

- *preenchida* (com v) em x sss $(c, v) \in x$
- *habilitada* em x sss x contem uma habilitação de c
- *acessível* a partir de x sss está habilitada, mas não está preenchida em x. ■

Denotar-se-á com $F(x)$, $E(x)$, $A(x)$ os conjuntos de células preenchidas, habilitadas e acessíveis em, ou a partir de, x ("F", "E", "A" pelos nomes em inglês: Filled, Enabled e Accessible).

Escreve-se $x \prec y$ se $x<y$ e $(\forall z, x<z\leq y \Rightarrow z = y)$ ("y cobre x"). Escreve-se $x<_c y$ ($x \prec_c y$) sss $c \in A(x)$, $c \in F(y)$ e $x<y$ ($x \prec y$), ou seja $x \prec_c y$ significa que o estado y é igual a x a menos de uma célula c que estava acessível em x e está preenchida em y.

3.2.5 Definição (cds bem-fundada)

Uma cds é *bem-fundada* se o fecho reflexivo da relação << definida em C como

$$c_1 << c_2 \text{ sss uma habilitação de } c_2 \text{ contém um evento } (c_1, v)$$

é bem-fundada, i.e. não existe uma seqüência infinita $(c_n)_{n\geq 0}$ tal que $c_{n+1} << c_n << ...c_0$. ■

3.2.6 Definição (cds estável)

Uma cds é chamada *estável* se para todo estado x e célula c habilitada em x, se $X, X' \vdash c$ e $X, X' \subseteq x$, então $X = X'$. ■

3.2.7 Definição (cds determinísitca)

Uma cds bem-fundada e estável é chamada *dcds* ("d" de determinístico). ■

Em [CUR 86], se mostra que a classe de cpo's geradas por cds estáveis é a mesma que a classe de cpo's gerada por cds bem-fundadas e estáveis, e que a condição de estabilidade corresponde à distributividade da cpo gerada.

3.2.8 Definição (cds seqüencial e filiforme)

Uma cds é chamada *seqüencial* se para toda célula d, qualquer estado x de M tal que

$$d \notin F(x) \text{ e } \exists\, y \geq x \text{ tal que } d \in F(y)$$

existe uma célula c tal que

$$c \in A(x) \text{ e } \forall\, y \geq x,\ d \in F(y) \Rightarrow c \in F(y).$$

Tal célula é chamada *índice de seqüencialidade* de M para d em x.
Uma cds M é chamada *filiforme* se toda habilitação contem no máximo um evento. ■

3.2.9 Proposição

Uma cds estável e filiforme é seqüencial. ■

3.2.10 Definição (produto de cds)

Sejam M, M' duas cds. Definimos o *produto* $M \times M' = (C, V, E, \vdash)$ de M e M' como

- $C = \{c.1 \mid c \in C_M\} \cup \{c'.2 \mid c' \in C_{M'}\}$
- $V = V_M \cup V_{M'}$
- $E = \{(c.1, v) \mid (c, v) \in E_M\} \cup \{(c'.2, v') \mid (c', v') \in E_{M'}\}$
- $(c_1.1, v_1),\ldots, (c_n.1, v_n) \vdash c.1$ sss $(c_1, v_1),\ldots, (c_n, v_n) \vdash c$ (e simetricamente com "'" e "2")

Claramente $M \times M'$ gera $D(M) \times D(M')$ (o produto de conjuntos usual). Se x, x' são estados de M, M', escrevemos

$$(x, x') = \{(c.1, v) \mid (c, v) \in x\} \cup \{(c'.2, v') \mid (c', v') \in x'\}.$$

Podemos definir o produto de qualquer número (finito ou infinito) de cds da mesma maneira. ■

3.3 Teoremas de representação

Nesta seção se apresenta a parte abstrata das cds: os domínios concretos (de G. Khan e G. Plotkin [KHA 93]), que são ordens parciais ω-algébricas que verificam 4 axiomas adicionais. As ordens parciais *D*(*M*) geram a classe de todos os domínios concretos quando *M* pertence à classe de todas os cds. Este resultado é chamado de teorema da representação.

G. Winskel trabalhou em estruturas que são mais gerais que estruturas de dados concretas, inspiradas por redes de Petri: as estruturas de eventos, onde não temos células nem valores, senão somente eventos. As ordens parciais correspondentes às estruturas de eventos são chamadas de domínios de eventos. Nesta seção se verão estas definições assim como também algumas propriedades e relações entre as mesmas.

3.3.1 Definição (estrutura de eventos)

Uma estrutura de eventos E = (E, #, $\vdash$) é um conjunto contável E de *eventos*, uma relação binária simétrica "#" em E de *conflito* e uma relação de *habilitação* "$\vdash$" entre partes finitas de E e elementos de E. Um *estado* de (E, #, $\vdash$) é um conjunto *x* de eventos que verifica as seguintes condições de *consistência* e *segurança*:

1. $e_1 \in x \wedge e_1 \# e_2 \Rightarrow e_2 \notin x$
2. $e \in x \Rightarrow \exists\, e_0, e_1, ..., e_n \in x$ tal que $(\forall i \leq n, \exists\, X_i \subseteq \{ e_j \,/\, j < i \}; X_i \vdash e_j)$.

O conjunto de estados de *E* ordenado por inclusão é denotado por ($D(E)$, ≤), que segundo Droste, em [DRO 89], é chamado de *domínio de eventos canônico associado com E.* ■

A diferença principal entre estas estruturas e as cds é a ausência de células; a noção de conflito é dada a priori. A relação formal entre cds e estruturas de eventos é estabelecida mais adiante.

O significado destas definições é claro em $D(E)$, o conjuntos de estados de uma estrutura de eventos. Uma cadeia $(x_i)_{i \leq n}$ de x até $x \cup \{e_1,..., e_n\}$ é tal que

$$\forall 0 \leq i < n, x_i = x_{i-1} \cup \{e_{s(i)}\}$$

onde s é uma permutação sobre $\{1,..., n\}$ (não qualquer permutação: todos os x_i devem ser seguros (ter a propriedade de segurança)). Se $[x, x'] \succ\!\prec [y, y']$, então $\exists e \in E$ tal que $x' = x \cup \{e\}$, $y' = y \cup \{e\}$. Portanto estas definições permitem, intuitivamente, considerar os eventos independentemente da parte abstrata. Os axiomas dos domínios de eventos estabelecem propriedades destas cadeias e relações de equivalência.

3.3.2 Definição (domínio de eventos)

Um *domínio de eventos* é um cpo ω-algébrico $(D, \leq)$ que satisfaz as seguintes condições para quaisquer elementos isolados (compactos) $x, x', y, y' \in D$:

(F) $\{y \in D \mid y \leq x\}$ é finito;

(C) se $x \prec y$, $x \prec z$, $z \neq y$ e $y \uparrow z$, então existe $y \vee z$, e $y \prec y \vee z$, $z \prec y \vee z$;

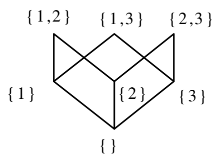
(R) $[x, y] \succ\!\prec [x, z] \Rightarrow y = z$.

(V) $[x, x'] \succ\!\prec [y, y']$, $[x, x''] \succ\!\prec [y, y'']$ e $x' \uparrow x'' \Rightarrow y' \uparrow y''$. ■

Em particular, o axioma (F) implica que $\prec$ restrita a elementos isolados é bem-fundada.

3.3.3 Exemplo

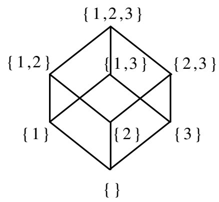
Seja $E = (E, \#, \vdash)$ uma estrutura de eventos tal que: $E = \{1,2,3\}$ e $\forall A \subseteq \text{Fin}(E) \backslash \{E\}$, $A \vdash i$, $\forall i \in E$. O domínio de eventos canônico $(D(E), \subseteq)$ associado com E pode ser visto na seguinte figura:



■

3.3.4 Exemplo

Seja $E = (E, \#, \vdash)$ uma estrutura de eventos tal que: $E = \{1,2,3\}$ e $\forall A \subseteq \mathrm{Fin}(E)$, $A \vdash i$, $\forall i \in E$. O domínio de eventos canônico $(D(E), \subseteq)$ associado com E pode ser visto na seguinte figura:



■

Na seção 2.2 de [CUR 86] são desenvolvidos vários lemas que preparam para o teorema da representação de Winskel, que serão omitidos aqui. Apresenta-se somente o teorema que relaciona as estruturas de eventos com os domínios de eventos (teorema da representação).

3.3.5 Teorema (G. Winskel)

Seja $(E, \#, \vdash)$ uma estrutura de eventos. $D(E)$, ordenado por inclusão, o domínio de eventos associado a E. Seja D um domínio de eventos. Denotamos com E_D o conjunto das classes de equivalência de intervalos primos de D. Definimos as relações # e $\vdash$ (para quaisquer elementos isolados) como:

$$z \prec z',\ z \prec z'',\ z' \# z'' \Rightarrow [z, z']_{\succ\prec} \# [z, z'']_{\succ\prec}$$

$$s(x) \vdash [x, x']_{\succ\prec}$$

e s é um isomorfismo de D em $D(E_D)$. ■

(Ver prova em [CUR 86], pag. 137).

Pode-se estabelecer a relação entre cds e estruturas de eventos de uma forma mais precisa.

3.3.6 Proposição

- Se $M = (C, V, E, \vdash)$ é uma cds, pode-se definir a relação # em E como

$$e_1 \# e_2 \text{ sss } \exists c, v_1, v_2 \text{ tal que } e_1 = (c, v_1), e_2 = (c, v_2) \text{ e } v_1 \neq v_2.$$

Escreve-se $X \vdash (c, v)$ se $X \vdash c$ e $(c, v) \in E$. Claramente M e $(E, \#, \vdash)$ têm os mesmos conjuntos de estados.

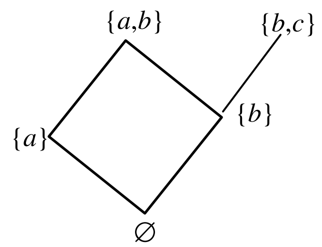
- Reciprocamente, se $(E, \#, \vdash)$ é uma estrutura de eventos, é possível achar uma cds M tal que $D(M)$ e $D(E)$ sejam isomorfos. Pode-se ver que as seguintes condições são suficientes:

- o fecho reflexivo de # é transitivo
- $\forall X, e_1, e_2, X \vdash e_1, e_1 \# e_2 \Rightarrow X \vdash e_2$. ■

Constrói-se M considerando como células as classes de equivalência do fecho reflexivo de #, como valores os eventos de E, como eventos as tuplas $([e], e)$ onde $[e]$ denota a classe de e, e $X \vdash [e]$ sss $X \vdash e$.

Relacionar-se-á a parte abstrata das cds, os domínios concretos (definidos no capítulo 1), com os domínios de eventos, mas antes se verá intuitivamente, por que a propriedade (Q) diferencia um domínio de eventos de um domínio concreto.

Seja por exemplo o domínio da seguinte figura:



que é um domínio de eventos, mas não um domínio concreto, pois não cumpre a propriedade (Q). A estrutura de eventos correspondente é $(E, \#, \vdash)$, onde $E = \{a, b, c\}$, com $a \# c$ e $\vdash a$, $\vdash b$, $b \vdash c$. Entretanto, nas cds não é possível especificar que $a \# c$, (pois nas cds $M = (C, V, E, \vdash)$ não existe relação de conflito), portanto o domínio de eventos anterior não poderia ser um domínio concreto, isto é, o conjunto de estados de uma cds.

3.3.7 Proposição

Um domínio concreto é um domínio de eventos tal que para todos os elementos isolados x, x', x'', y, y',

$$[x, x'] \succ\!\prec [y, y'], x \prec x'' \text{ e } x'\#x'' \Rightarrow \exists y'' \text{ tal que } [x, x''] \succ\!\prec [y, y''] \text{ e } y'\#y''. \blacksquare$$

Na prova da proposição anterior (ver [CUR 86], pag 139) somente é usado o axioma (C), portanto

$$\text{(Q), (C)} \Rightarrow \text{(V).}$$

A proposição anterior é usada na prova do seguinte teorema de representação:

3.3.8 Teorema

O conjunto de estados de uma cds, ordenados por inclusão, é um domínio concreto. Além disso, se D é um domínio concreto, então existe uma cds M tal que $D(M)$ e D são isomorfos. ■

3.3.9 Definição (cpo distributiva)

Uma cpo consistentemente completa é chamada *distributiva* se para todos os elementos x, y, z tal que $y\uparrow z$,

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z). \blacksquare$$

O seguinte lema estabelece uma conseqüência essencial da distributividade para um domínio concreto.

3.3.10 Lema

Se D é um domínio concreto distributivo, a seguinte propriedade é válida:

(M) qualquer classe de equivalência de intervalos primos e tem um "mínimo" representativo $[x, x']$, i.e. tal que $\forall [y, y'] \in e, x \leq y$. ■

No caso distributivo temos o seguinte teorema de representação:

3.3.11 Teorema

Se M é uma cds estável, então $D(M)$ é um domínio concreto distributivo. Se D é um domínio concreto distributivo, então existe uma dcds $M(D)$ tal que $D(M(D))$ e D são isomorfos. ■

Como resumo desta seção, se pode dizer que a condição de ser bem-fundado não restringe a generalidade das cds estáveis. Observe também que a proposição anterior não é limitada a cds e domínios concretos. As estruturas de eventos estáveis podem ser definidas da mesma maneira que as cds estáveis, e estas estruturas correspondem aos domínios de eventos distributivos.

Além das construções de Khan-Plotkin, tem-se mostrado outros teoremas de representação entre estruturas concretas e domínios de ordem teórica, entre as que se têm:

Estruturas concretas:	**Estruturas de ordem (domínios)**
Estruturas de dados concretas	Domínios concretos
Estruturas de eventos	Domínios de eventos
Variante das Estruturas de eventos	Cpo's algébricos coerentes primos
Sistemas de Informação	Cpo's algébricos consistentemente completos

3.4 Funções seqüenciais

Ver-se-á nesta seção o conceito de função estável de Berry, que é uma noção intermediária entre as funções contínuas e as seqüenciais. Segundo Curien [CUR 86] (pag. 150):

3.4.1 Definição (função estável)

Sejam D, D' duas cpo's. Uma função contínua f: $D \to D'$ é chamada *estável* se para todo $x \in D$, $x' \in D'$, $x' \leq f(x)$, existe um ponto $M(f, x, x')$ tal que $\forall z \leq x$, $x' \leq f(z)$ se e somente se $M(f, x, x') \leq z$. ■

3.4.2 Definição (função condicionalmente multiplicativa)

Uma função contínua f:$D \to D'$ é chamada *condicionalmente multiplicativa*[6] (*mc*) $\forall x, y, x \uparrow y \Rightarrow f(x \wedge y) = f(x) \wedge f(y)$ (em particular, $x \wedge y$ existe). ■

Nem todas as funções contínuas são estáveis, seja por exemplo: *por*: D(Bool) $\times$ D(Bool) $\to$ D(Bool) definida como

$$por(x, y) = \mathrm{T} \quad \text{sss } x = \mathrm{T} \text{ ou } y = \mathrm{T}$$
$$por(x, y) = \mathrm{F} \quad \text{sss } x = y = \mathrm{F}$$

não é estável pois não existe $M(por,$ (T, T), T).

A seguinte proposição relaciona as funções estáveis com as condicionalmente multiplicativas.

3.4.3 Proposição

Se M e M' são duas cds, com M estável e f é contínua de $D(M)$ em $D(M')$, então as seguintes propriedades são equivalentes:

1. f é estável
2. f é *mc*
3. $\forall X$ superiormente cotado, $f(\bigwedge X) = \bigwedge f(X)$
4. $\forall x, c'$ tal que $c' \in F(f(x))$, existe um estado $MF(f, x, c')$ tal que $\forall z \leq x$, $c' \in F(f(z))$
5. quaisquer dois elementos do conjunto $MF(f, c')$ de pontos minimais z tal que $c' \in F(f(z))$ são incompatíveis. ■

---

[6]Girard chama ([GIR 89], pag. 58) função estável o que Curien chama de função condicionalmente multiplicativa (ver também [TRO 92], pag. 94).

A importância desta proposição, neste trabalho, é que, toda cds que é teia de um espaço coerente é estável, pois $\forall c \in C$, $\vdash c$ (ver definição 3.2.6), e portanto, falar de função estável e condicionalmente multiplicativa é indiferente.

3.4.4 Definição (função seqüencial)

Sejam M e M' duas cds. Uma função contínua de $D(M)$ em $D(M')$ é chamada *seqüencial* em x, se para todo $c' \in A(f(x))$ (toda célula acessível a partir de $f(x)$), é válida uma das seguintes propriedades:

1. ou $A(x) = \emptyset$, e portanto x não tem nenhum estado "acima" dele;
2. ou $\exists c \in A(x)$ tal que $\forall y \in D(M)$, $x \leq y$ e $c' \in F(f(y)) \Rightarrow c \in F(y)$.[7]

Tal célula c é chamada de *índice de seqüencialidade* de f em x para c'. O índice é chamado *estrito* se 1. não é válido. ■

A interpretação intuitiva desses conceitos é a seguinte: a primeira condição diz que o preenchimento das células de saída parou em x; a segunda expressa que para cada célula preenchida na saída ocorre que uma célula é preenchida na entrada. A seqüencialidade diz, portanto, que cada evento na saída exige um evento na entrada; e o índice de seqüencialidade de uma célula (evento) de saída em um certo estado é a célula (evento) na entrada que deve ser preenchida para obter a célula (evento) na saída, no estado dado. A interpretação anterior sugere que as funções seqüenciais são estritas.

Todas as funções seqüenciais são estáveis, mas a recíproca não é verdadeira: existem funções estáveis que não são seqüenciais.

3.4.5 Exemplos

Os seguintes exemplo foram tomados de [CUR 86] e [BRO 90].

[7] A definição de seqüencialidade dada aqui foi a de Brookes & Geva (ver [BRO 90]). A definição original ([CUR 86]) estabelece as seguintes condições: 1. $\forall y \geq x$, $c' \notin F(f(y))$; 2. $\forall x \in D(M)$, $\forall c' \in C'$, $\exists c \in A(x)$ tal que $\forall y \in D(M)$, $f(x) <_{c'} f(y) \Rightarrow x <_{c} y$.

Seja a cds dos Booleanos: Bool, que tem uma única célula inicial c que pode ser preenchida com os valores T (verdadeiro) e F (falso). Seja a cds Bool $\times$ Bool, produto cartesiano da anterior, que tem duas células iniciais $c.1$ e $c.2$, onde cada uma pode ser preenchida com os valores T e F. Esta última tem nove estados ($\varnothing$,$\{(c.1, \perp), (c.2, \mathrm{F})\}$, $\{(c.1, \mathrm{T}), (c.2, \mathrm{F})\}$, ...).

3.4.5.1 Função *sor*

Defina a função or-estrita *sor*: $D(\mathrm{Bool} \times \mathrm{Bool}) \rightarrow D(\mathrm{Bool})$, como sendo a menor função monotônica que satisfaz:

$sor(\{(c.1, \mathrm{T}), (c.2, \mathrm{T})\}) = \{(c, \mathrm{T})\}$

$sor(\{(c.1, \mathrm{T}), (c.2, \mathrm{F})\}) = \{(c, \mathrm{T})\}$

$sor(\{(c.1, \mathrm{F}), (c.2, \mathrm{T})\}) = \{(c, \mathrm{T})\}$

$sor(\{(c.1, \mathrm{F}), (c.2, \mathrm{F})\}) = \{(c, \mathrm{F})\}$.

sor é estável e seqüencial. Tanto $c.1$ quanto $c.2$ são índices de seqüencialidade no estado $\varnothing$, para c.

3.4.5.2 Função *lor*

Defina a função or-estrita-a-esquerda *lor*: $D(\mathrm{Bool} \times \mathrm{Bool}) \rightarrow D(\mathrm{Bool})$, como sendo a menor função monotônica que satisfaz:

$lor(\{(c.1, \mathrm{T}), (c.2, \perp)\}) = \{(c, \mathrm{T})\}$

$lor(\{(c.1, \mathrm{F}), (c.2, \mathrm{T})\}) = \{(c, \mathrm{T})\}$

$lor(\{(c.1, \mathrm{F}), (c.2, \mathrm{F})\}) = \{(c, \mathrm{F})\}$.

lor é estável e seqüencial, com $c.1$ como índice de seqüencialidade no estado $\varnothing$, para c.

3.4.5.3 Função *ror*

Defina a função or-estrita-a-direita *ror*: D(Bool × Bool) → D(Bool), como sendo a menor função monotônica que satisfaz:

ror({(*c*.1, ⊥), (*c*.2,T)}) = {(*c*, T)}

ror({(*c*.1, T), (*c*.2, F)}) = {(*c*, T)}

ror({(*c*.1, F), (*c*.2, F)}) = {(*c*, F)}.

ror é estável e seqüencial, com *c*.2 como índice de seqüencialidade no estado ∅, para *c*.

3.4.5.4 Função *por*

Defina a função or-paralelo *por*: D(Bool × Bool) → D(Bool), como sendo a menor função monotônica que satisfaz:

por({(*c*.1, ⊥), (*c*.2, T)}) = {(*c*, T)}

por({(*c*.1, T), (*c*.2, ⊥)}) = {(*c*, T)}

por({(*c*.1, F), (*c*.2, F)}) = {(*c*, F)}.

por não é estável nem seqüencial: não tem índice de seqüencialidade no estado ∅, para *c*; e não há um único estado minimal de Bool × Bool menor que {(*c*.1, T), (*c*.2, T)} para o qual *por* dá o valor {(*c*, T)}, portanto nenhum estado de Bool × Bool serve como *M*(*por*, {(*c*.1, T), (*c*.2, T)}, {(*c*, T)}).

3.4.5.5 Função *gf*

Defina a função *gf*: D((Bool × Bool) × Bool) → D(Bool), como sendo a menor função monotônica que satisfaz:

gf({(*c*.1, T), (*c*.2, F), (*c*.3, ⊥)}) = {(*c*, T)}

gf({(*c*.1, ⊥), (*c*.2, T), (*c*.3, F)}) = {(*c*, T)}

gf({(*c*.1, F), (*c*.2, ⊥), (*c*.3, T)}) = {(*c*, T)}

$gf(\{(c.1, \mathrm{F}), (c.2, \mathrm{F}), (c.3, \mathrm{F})\}) = \{(c, \mathrm{F})\}$.

A função *gf* é estável, mas não é seqüencial: não tem índice de seqüencialidade no estado ∅ para *c*. ■

A seguinte proposição explicita a natureza finita das funções seqüenciais.

3.4.6 Proposição

Sejam *M* uma cds, *M'* uma dcds e $f{:}D(M) \rightarrow D(M')$ uma função contínua. *f* é seqüencial se e somente se é seqüencial em todo ponto finito. ■

Definem-se agora funções seqüenciais entre domínios concretos.

3.4.7 Definição (função seqüencial entre domínios concretos)

Sejam *D, D'* dois domínios concretos, seja f: $D \rightarrow D'$; *f* é chamada *seqüencial* se existem duas cds *M, M'* e dois isomorfismos i: $D \rightarrow D(M)$ e i': $D' \rightarrow D(M')$ tal que $i' \circ f \circ i^{-1}$ é seqüencial de *M* em *M'*. ■

Logicamente a definição não depende da escolha de *M, M'*.

Curien estabelece a relação entre as categorias de cds (estáveis) e funções seqüenciais (**CdsSeq**) com a categoria de domínios concretos (distributivos) e funções seqüenciais (**DcSeq**): as mesmas são equivalentes. Outro resultado interessante é que estas duas categorias não são cartesianas fechadas (ver [CUR 86] pag. 158).

3.5 Algoritmos seqüenciais

Nesta seção se apresentam os algoritmos seqüenciais como objetos matemáticos. Para todas dcds *M*, *M'* definimos uma dcds $M \Rightarrow M'$, cujos estados são os algoritmos seqüenciais de *M* em *M'*. Uma função seqüencial que tem, em um ponto dado, mais de um índice de seqüencialidade, pode ser computado intuitivamente, de diferentes formas, de acordo com a ordem em que seus índices são explorados. Por exemplo a função adição sobre $D(\aleph)$ tem, no estado {}, *c*.1 e *c*.2 como índices de seqüencialidade. Assim, a soma esquerda computa o primeiro argumento (i.e. *c*.1) e depois o segundo, entretanto que a soma direita executa as mesmas computações na ordem contrária. Os algoritmos seqüenciais formalizam estas idéias.

3.5.1 As estruturas de dados concretas de algoritmos seqüenciais.

Definem-se os algoritmos seqüenciais como estados de estruturas de dados concretas.

3.5.1.1 Definição (a cds $M \Rightarrow M'$)

Se M e M' são duas cds, a cds $M \Rightarrow M'$ e definida como:

i) Se x é um estado finito de M, c' uma célula de M', então xc' é uma célula de $M \Rightarrow M'$

ii) Os *valores* e *eventos* são de dois tipos:

a) tipo "valof": se c é uma célula de M, então *"valof c"* é um valor de $M \Rightarrow M'$ e (xc', *valof c*) é um evento de $M \Rightarrow M'$ sss c é acessível a partir de x;

b) tipo "output": se v' é uma célula de M', então *"output v'"* é um valor de $M \Rightarrow M'$ e (xc', *output v'*) é um evento de $M \Rightarrow M'$ sss (c', v') é um evento de M';

iii) As relações de habilitação são de dois tipos:

a) (yc', *valof c*) $\vdash xc'$ sss $y \prec_c x$ e x é um estado finito (tipo "valof")

b) $(x^1c'^1$, *output* $v'^1), \ldots, (x^nc'^n$, *output* $v'^n) \vdash xc'$ sss $x = \cup\{x^i \mid i \leq n\}$, x é finito e $(c'^1, v'^1), \ldots, (c'^n, v'^n) \vdash c'$ (tipo "output"). ■

As componentes da exponencial $M \Rightarrow M'$ de duas cds M e M', podem ser interpretadas como:

- Células: pares cuja primeira componente é um estado de entrada x e uma célula de saída c', denotada por xc'. Intuitivamente: uma célula contém uma parte elementar de uma descrição concreta de uma função, que pode ser vista como uma pergunta: que fazer quando a entrada é x e o trabalho é preencher c'?

- Valores: "valof *c*" ou "output *v'*". Os valores são partes elementares de código que podem ser vistas como respostas às perguntas: ou a entrada já realizada não é suficiente e deve ser incrementada em *c*, ou é suficiente (a entrada) e pode ser escrito um valor resultado em *c'*.

- Eventos: (*xc'*, *valof c*), onde *c* está habilitada, mas não está preenchida em *x*; (*xc'*, *output v'*), onde (*c', v'*) é um evento da cds, i.e., *v'* é um valor possível para *c'*. Intuitivamente: se se quer preencher *c'* deve-se ler o valor *c*; se se escreve um valor em *c'*, o mesmo deve ser um valor legal para *c'*.

- Habilitação: há dois tipos de habilitação que estabelecem as condições para que um estado da cds $M \Rightarrow M'$ seja um algoritmo seqüencial. Intuitivamente: o tipo "valof" estabelece que se (*xc'*, *v'*) é um evento de um estado *a* da cds $M \Rightarrow M'$, então também pertence a *a* o evento que gerou *x*: (*yc'*, *valof c*), onde claramente $x = y \cup \{(c, v)\}$; o tipo "output" diz que antes de escrever o valor para uma célula de saída, se devem escrever os valores para as células de saída que habilitam a primeira.

3.5.1.2 Definição (algoritmo seqüencial)

Um estado de $M \Rightarrow M'$ é chamado *algoritmo seqüencial* ou simplesmente *algoritmo*.■

As computações no modelo são, operacionalmente, dirigidas por demanda: por exemplo, a informação de um observador externo acerca do resultado de aplicar um algoritmo a um estado de entrada, pode ser gradualmente incrementada preenchendo as células do estado resultado, onde cada demanda do valor de uma célula resulta em uma nova computação. Uma célula da exponenciação (da cds $M \Rightarrow M'$) consiste de um estado corrente finito *x*, que descreve a informação atual conhecida acerca da entrada, e um requerimento (ou pergunta) da computação de um valor de uma célula *c'* da saída. Os eventos de um algoritmo associam um comando a cada célula *xc'*, ou um comando *output v'*, que termina a computação e determina que o evento (*c', v'*) está na saída, ou um comando *valof c* que tenta incrementar o estado atual *x* preenchendo a célula *c*. Esta célula *c*, naturalmente suficiente, é o índice de seqüencialidade (da função de entrada-saída do algoritmo) no estado *x*, por isso a escolha de *c* entre todos os índices de seqüencialidade em *x* (se não for o único), determina a estratégia de computação. Quando a sub-computação termina (se termina), com um valor v para c na entrada, a computação principal reassume com a célula habilitada $(x \cup (c, v))c'$, até que um valor seja "dado" à célula c'. As sub-computações procedem da mesma forma: por isto é um sistema de co-rotina global.

3.5.1.3 Definição (aplicação de um algoritmo seqüencial)

Se a e x são estados de $M \Rightarrow M'$ e M respectivamente, então

$$a.x = \{(c', v') \mid \exists y \subseteq x \text{ tal que } (yc', \textit{output } v') \in a\}$$

é a *aplicação* de a no estado x. A função que realiza a associação $x \mapsto a.x$ é chamada a *função de entrada-saida* computada por a. ■

Apresenta-se agora um exemplo de algoritmo seqüencial (a soma "esquerda" e "direita"), dado por Curien (pag. 162 de [CUR 86]). Seja $\mathrm{ADD}_{\mathrm{ESQ}}$ o algoritmo que executa a soma de dois argumentos, avaliando primeiro o argumento esquerdo, e $\mathrm{ADD}_{\mathrm{DIR}}$ o algoritmo que executa a soma avaliando primeiro o argumento direito. Seja N a cds definida como a quaterna $(\{c\}, \aleph, \{(c, n) \mid n \in \aleph\}, \{\vdash c\})$. Os algoritmos seqüenciais têm a seguinte forma:

$$\mathrm{ADD}^{\aleph}_{\mathrm{ESQ}} = \{(\{\}c, \textit{valof } c.1), (\{(c.1, i)\}c, \textit{valof } c.2), (\{(c.1, i), (c.2, j)\}c, \textit{output } i{+}j) \mid i, j \in \aleph\}.$$

$$\mathrm{ADD}^{\aleph}_{\mathrm{DIR}} = \{(\{\}c, \textit{valof } c.2), (\{(c.2, j)\}c, \textit{valof } c.1), (\{(c.2, j), (c.1, i)\}c, \textit{output } i{+}j) \mid i, j \in \aleph\}.$$

Curien apresenta alguns resultados acerca das cds bem-fundadas, filiformes e estáveis que não são necessárias aqui. Outro conceito importante no trabalho de Curien é o de *algoritmo abstrato*, que pode ser visto como uma função seqüencial junto com uma estratégia de computação. Estes resultados são importantes para mostrar que a categoria das cds e algoritmos seqüenciais é cartesiana fechada, mas que não é relevante neste trabalho (ver seção 2.5 de [CUR 86]).

Na Figura 3.1 se apresentam as componentes de uma máquina que computa algoritmos seqüenciais: memória de entrada, memória de saída, memória de programa, memória de trabalho, valor de referência, controle e o oráculo.

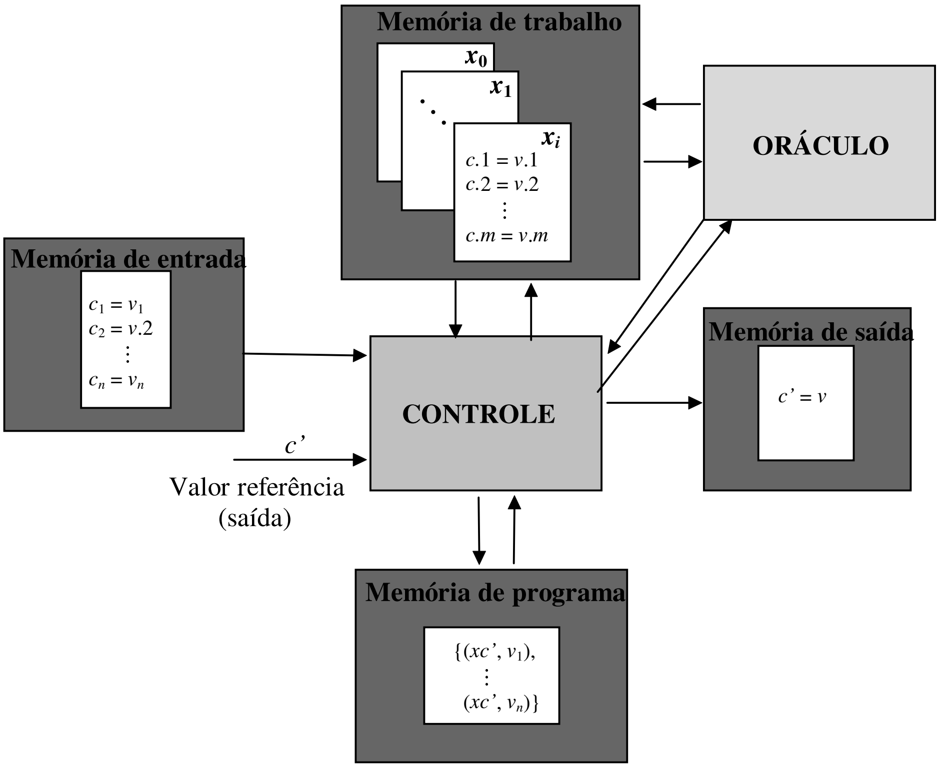


Figura 3.1 - Máquina que computa alg. seqüenciais

O controle recebe o valor de referência (célula a ser preenchida) e a memória de entrada. Na memória de programa se encontra o algoritmo a ser executado. O controle lê e atualiza a memória de trabalho e quando precisar executar alguma operação (como por exemplo a soma de dois valores), interage com o oráculo. Finalmente dá o resultado na memória de saída.

4 ESPAÇOS COERENTES

Neste capítulo se apresentam os *espaços coerentes* (também chamados de *domínios de Girard*). O modelo dos espaços coerentes é um modelo interessante da teoria de tipos da Lógica Linear Clássica (CLL). Girard introduziu os espaços coerentes com o fim de dar semântica à lógica linear.

4.1 Introdução

Na semântica denotacional os objetos do tipo $U \rightarrow V$ são interpretados como funções de U em V, onde o problema consiste em dar um significado razoável à palavra "função". Existem varias aproximações, entre as quais se encontra a idéia de Scott, onde um tipo é interpretado como um espaço topológico, e $U \rightarrow V$ são as funções contínuas de U em V. Por razões de ordem topológica (ver [GIR 89]), Girard considerou outra alternativa à categoria dos espaços topológicos com funções continuas: a dos espaços coerentes e funções lineares.

4.2 Espaços coerentes

4.2.1 Definição (teia)

Uma *teia* $\boldsymbol{A} \equiv (A, \approx_A)$[8] é um par formado por um conjunto A sobre o qual é definida uma relação reflexiva e simétrica $\approx_A$. ■

[8]Segundo Girard ([GIR 89], pag. 56) uma teia é definida como $|\mathcal{A}| \equiv \cup\mathcal{A} = \{e;\ \{e\} \in \mathcal{A}\}$, onde $\mathcal{A}$ é um espaço coerente e $\alpha \approx_A \beta$ sss $\{\alpha, \beta\} \in \mathcal{A}$.

4.2.2 Definição (subconjuntos coerentes)

Um subconjunto x de uma teia $\boldsymbol{A}$ é coerente se e somente se $\forall \alpha, \beta \in x, (\alpha \approx_A \beta)$. O conjunto de todos os *subconjuntos coerentes* de $\boldsymbol{A}$ é denotado por $Coh(\boldsymbol{A})$. Tem-se então:

$$Coh(\boldsymbol{A}) = Coh(A, \approx_A) = \{ x \subseteq A \mid \forall \alpha, \beta \in x, (\alpha \approx_A \beta) \}. \blacksquare$$

4.2.3 Definição (espaços coerentes)

Um *espaço coerente* (ou *domínio de Girard*) $\mathcal{A} = (Coh(A, \approx_A), \subset)$ é a coleção de subconjuntos coerentes de uma teia $\boldsymbol{A}$ parcialmente ordenados por inclusão. Os elementos de A são chamados *tokens* (ou *átomos*) do domínio $\mathcal{A}$. O conjunto dos subconjuntos coerentes finitos de $\boldsymbol{A}$ é denotado por $Fincoh(\boldsymbol{A})$ ou $\mathcal{A}_{\text{fin}}$. ■

Quando não der lugar a confusão, o espaço coerente será denotado indistintamente por $\mathcal{A}$ ou A, ficando claro que A é o conjunto subjacente do espaço coerente $\mathcal{A}$. Da mesma maneira, a teia será escrita como $|\mathcal{A}|$, $\boldsymbol{A}$ ou $|A|$.

A seguinte proposição é dada em [TRO 92] (ver pag. 93), e caracteriza os espaços coerentes. A prova da seguinte proposição, assim como todas as desta seção podem ser encontradas no trabalho de Dimuro em [DIM 94].

4.2.4 Proposição

Se $\mathcal{A}$ é um espaço coerente, então:

i) $\mathcal{A}$ contém todos os conjuntos unitários $\{\alpha\} \subseteq A$,

ii) $a \in \mathcal{A}$, $b \subseteq a \Rightarrow b \in \mathcal{A}$ (fecho inferior, ou propriedade (dc)),

iii) $B \subseteq \mathcal{A}$, $\forall c, c' \in B$, $(c \cup c') \in \mathcal{A} \Rightarrow \bigcup B \in \mathcal{A}$ (completeza binária, ou propriedade (bc)),

iv) $\emptyset \in \mathcal{A}$,

v) $\mathcal{A}$ é fechado em relação a uniões dirigidas (i.e. dirigido com respeito a $\subseteq$), ou seja:

$$a_i \in \mathcal{A} \Rightarrow \bigcup\uparrow \{a_i \mid i \in I\} \in \mathcal{A},$$

ou se X é dirigido com relação à $\subseteq$ em $\mathcal{A}$, então $\bigcup X \in \mathcal{A}$.

vi) $\mathcal{A}$ é fechado em relação a interseções arbitrárias, isto é,

$$\forall i \in I,\ I \text{ arbitrário},\ a_i \in \mathcal{A} \Rightarrow \bigcap\{a_i \mid i \in I\} \in \mathcal{A}.\ \blacksquare$$

Observações:

i) Para mostrar que $\mathcal{A}$ é um espaço coerente, é suficiente mostrar que $\mathcal{A}$ é uma coleção de conjuntos que satisfaz as condições ii) e iii) (propriedades (dc) e (bc)) da proposição anterior. Em particular, tem-se que o objeto indefinido $\emptyset \in \mathcal{A}$.[9]

ii) Um espaço coerente pode ser considerado como um domínio (ordenado parcialmente pela inclusão); e, como tal, um espaço coerente é um domínio, algébrico (todo conjunto é a união dirigida de seus subconjuntos finitos) e que satisfaz a condição de completeza binária (condição ii) da proposição anterior).

iii) Os espaços coerentes podem ser vistos como grafos não dirigidos.

4.2.5 Proposição

Se $X \subseteq \wp(A)$ satisfaz as condições i) a iii) da proposição anterior, então existe uma relação reflexiva e simétrica $\approx$ sobre A tal que $X = Coh((A, \approx))$. ■

Dado um espaço coerente $\mathcal{A}$, é possível recuperar sua teia $\boldsymbol{A}$. Tem-se que

$$\boldsymbol{A} \equiv |\mathcal{A}| = \bigcup \mathcal{A} = \{\alpha;\ \{\alpha\} \in \mathcal{A}\}, \text{ e}$$

$$\alpha \approx_A \beta \text{ sss } \{\alpha, \beta\} \in \mathcal{A}.$$

A construção da teia de um espaço coerente é uma bijeção entre espaços coerentes e grafos (reflexivos e simétricos). A partir da teia pode-se recuperar o espaço coerente:

$$a \in \mathcal{A} \text{ sss } a \subseteq |\mathcal{A}| \wedge \forall \alpha, \beta \in a,\ \alpha \approx \beta.$$

O contrário também é válido, pois $\forall \alpha, \beta \in a$, $\alpha \approx \beta$ se e somente se $\{\alpha, \beta\} \in \mathcal{A}$.

---

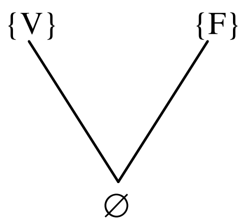
9 Girard ([GIR 89]) define o espaço coerente como um conjunto (de conjuntos) $\mathcal{A}$ que satisfaz justamente estas duas condições ii) e iii): se $e_1 \in \mathcal{A} \wedge e_2 \subset e_1 \Rightarrow e_2 \in \mathcal{A}$ (fecho inferior); e, se $M \subset \mathcal{A}$ e se $\forall\ e_1, e_2 \in M$, $e_1 \cup e_2 \in \mathcal{A} \Rightarrow \cup M \in \mathcal{A}$ (completeza binária)

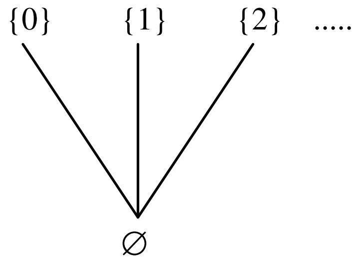
Esta teia constitui um grafo, e na Teoria dos Grafos, um ponto (objeto) é exatamente um clique, isto é, um subgrafo completo. Os átomos de um espaço coerente representam bits atômicos de informação e um conjunto coerente é uma quantidade consistente de informação. A coerência entre tokens significa que estes átomos podem ser vistos como pedaços de informação com relação ao mesmo objeto. A ordem de informação se reflete pela inclusão: $a \subseteq b$ significa que b representa mais informação do que a.

A seguir apresentam-se alguns exemplos de espaços coerentes.

4.2.6 Exemplo

Todos os domínios *flat* são espaços coerentes. Ou seja, os números naturais e os *booleanos*, assim como qualquer outro conjunto de elementos básicos com $\{\emptyset\}$ é um espaço coerente:

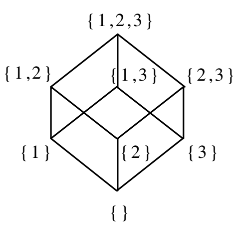


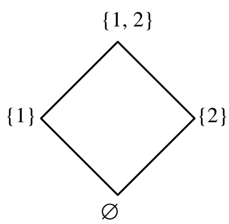


■

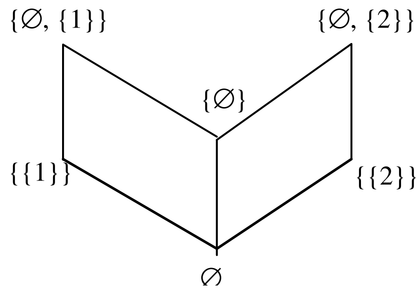
4.2.7 Exemplo

Os seguintes domínios são espaços coerentes:





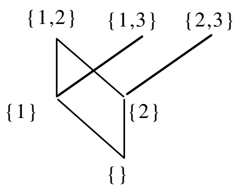
Assim como o seguinte:

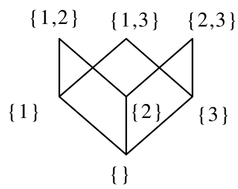


■

4.2.8 Exemplo

Os seguintes não são espaços coerentes:





O primeiro não cumpre a propriedade ii) (fecho inferior) nem a completeza binária, entretanto que o segundo não cumpre a completeza binária. ■

4.2.9 Exemplo

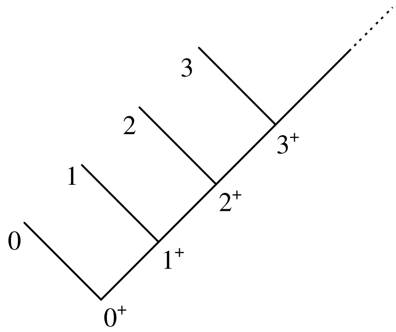
Para conjuntos arbitrários X, Y, considere $(X\times Y, \approx)$ com

$$(x, y) \approx (x', y') \text{ sss } x = x' \Rightarrow y = y'.$$

Então, $Coh(X\times Y, \approx)$ consiste das funções parciais de X em Y. ■

4.2.10 Exemplo

Seja o seguinte domínio dos números naturais parciais (ver [ESC 90], e [GIR 89]):



Os números naturais parciais são úteis em situações onde não só interessam as computações totais, senão também respostas parciais. Por exemplo, se estou em um quarto, e quero saber quantas pessoas têm em outro quarto. Se a "máquina" que está contando trabalha somente com números naturais (não parciais), e não terminou de contar, então não posso saber a quantidade até que ela não termine o processo. Mas, eventualmente, eu poderia querer tomar decisões com resultados parciais, então seria bom que, por exemplo, cada 5 minutos pergunte quantas pessoas têm, e ela me responda "pelo menos 12" (representado pelo número parcial 12^+); 5 minutos depois ela responde "pelo menos 35" (representado por 35^+); etc.

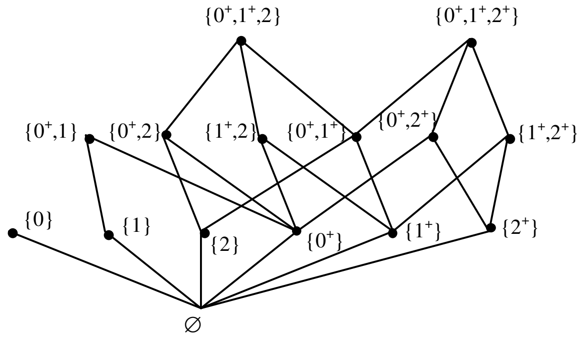
Obviamente, o anterior não é um espaço coerente (os objetos não são conjuntos, e a ordem não é a inclusão), mas podemos obter um espaço coerente definindo Int^+ por $|Int^+| = \{0, 0^+, 1, 1^+, 2, 2^+, ...\}$, com:

$$p \approx q \;\Leftrightarrow\; p = q \quad p^+ \approx q \;\Leftrightarrow\; p \leq q \quad p^+ \approx q^+ \;\; \forall\, p, q.$$

Considerar-se-á somente $|Int^+| = \{0, 0^+, 1, 1^+, 2, 2^+\}$; portanto o espaço coerente é:

$Int^+ = \{\emptyset, \{0\}, \{1\}, \{2\}, \{0^+\}, \{1^+\}, \{2^+\}, \{0^+, 1^+\}, \{0^+, 2^+\}, \{1^+, 2^+\}, \{1^+, 2\}, \{0^+, 1\}, \{0^+, 2\}, \{0^+, 1^+, 2^+\}, \{0^+, 1^+, 2\}\}$

A representação gráfica é a seguinte:



Pode observar-se que para o espaço coerente de todos os números naturais parciais, os objetos maximais são da forma $\{0^+, 1^+, ..., n\text{-}1^+, n\}$. ■

4.3 Interpretação

O objetivo principal de uma semântica baseada em espaços coerentes é interpretar um tipo através de um espaço coerente $\mathcal{A}$, e um termo deste tipo por um ponto de $\mathcal{A}$ (subconjunto coerente de $|\mathcal{A}|$, infinito em geral. Para trabalhar de uma maneira efetiva com pontos de $\mathcal{A}$ é necessário introduzir a noção de aproximação finita.

4.3.1 Definição (aproximação)

Uma *aproximação* de $a \in \mathcal{A}$ é um subconjunto a' de a. Se a' é finito, então se diz que a' é uma *aproximação finita* de a. ■

A condição ii) da proposição 4.2.4 (fecho inferior), garante que as aproximações também estão em $\mathcal{A}$. Tem-se que:

4.3.2 Proposição

i) a é a união de seu conjunto de aproximações finitas;

ii) o conjunto I das aproximações finitas é dirigido, em outras palavras

a) I é não vazio ($\varnothing \in I$)

b) se $a', a'' \in I$, podemos encontrar $a \in I$ tal que $a', a'' \subseteq a$ (considere $a = a' \cup a''$). ■

O anterior vem das seguintes definições:

4.3.3 Definição (objeto total)

Um *objeto total* x em um espaço coerente $\mathcal{A}$ é um objeto maximal de $\mathcal{A}$, isto é, x é um subconjunto coerente de $|\mathcal{A}|$ tal que se existe $\beta \in A$, tal que $\forall \alpha \in x$, $\beta \approx \alpha$, então $\beta \in x$. ■

A definição anterior quer dizer que, se x é um objeto total, então quaisquer dois elementos de x estão relacionados por $\approx$ e não existe nenhum outro elemento de A que se relacione com todos os elementos de x e que não pertença a x. Denota-se o conjunto de objetos totais de $\mathcal{A}$ por $tot(\mathcal{A})$.

4.3.4 Definição (objeto parcial)

Qualquer objeto de um espaço coerente $\mathcal{A}$ é um *objeto parcial*. Os objetos totais são casos particulares dos objetos parciais. ■

Os objetos totais dos inteiros e os booleanos, são todos os conjuntos unitários ("*singletons*"), entretanto que as aproximações destes objetos totais (neste caso unicamente o $\emptyset$) são objetos parciais.

Os objetos parciais tem o mesmo significado que na teoria da recursão, onde se obtém funções parciais a partir de funções totais com a operação de *lifting* (adicionar um elemento "indefinido" $\emptyset$).

4.4 Funções em espaços coerentes

Sejam $\mathcal{A}$ e $\mathcal{B}$ dois espaços coerentes. As funções de $\mathcal{A}$ em $\mathcal{B}$ serão vistas como funções definidas unicamente por suas aproximações, e desta forma "contínuas". Aqui é possível usar uma linguagem topológica, onde os subconjuntos $\{a \mid a_0 \subseteq a\}$ de $\mathcal{A}$, para a_0 finito, são os abertos da topologia. Entretanto, na teoria de domínios de Scott as funções entre domínios são exatamente aquelas que são contínuas para esta topologia, isto não acontecerá nesta teoria.

4.4.1 Funções monotônicas, contínuas, estáveis e lineares

4.4.1.1 Definição (função monotônica)

Uma função $F\colon \mathcal{A} \rightarrow \mathcal{B}$ é *monotônica* se para todo a, $a' \in \mathcal{A}$, tem-se que

$$a \subseteq a' \Rightarrow F(a) \subseteq F(a'). \blacksquare$$

Uma função monotônica é portanto aquela que preserva aproximações: se é fornecida mais informação no início de um processo, então se obtém maior retorno no final. Alternativamente, F somente utiliza informação positiva sobre seus argumentos.

4.4.1.2 Definição (função contínua)

Uma função $F\colon \mathcal{A} \rightarrow \mathcal{B}$ é *contínua* se para todo subconjunto dirigido (com relação à inclusão) X de $\mathcal{A}$ tem-se que

$$F(\bigcup X) = \bigcup \{F(b) \mid b \in X\}. \blacksquare$$

Esta condição também é conhecida como a preservação de uniões dirigidas em relação à inclusão, e pode ser dada como:

$$F(\bigcup_{i \in I}{\uparrow} a_i) = \bigcup_{i \in I}{\uparrow} F(a_i),$$

que é equivalente a

$$F(a) = \bigcup{\uparrow}\{F(a_0) \mid a_0 \subseteq a,\ a_0 \text{ finito}\}.$$

4.4.1.3 Definição (função estável)

Uma função $F\colon \mathcal{A} \rightarrow \mathcal{B}$ é *estável* se é contínua e satisfaz a propriedade de estabilidade dada por:

$$a \cup a' \in \mathcal{A} \Rightarrow F(a \cap a') = F(a) \cap F(a'). \blacksquare$$[10]

[10]A definição 3.4.1 (ver capítulo 3) de estabilidade é diferente. A definição dada aqui corresponde à definição de função condicionalmente multiplicativa. De agora em diante utilizaremos esta definição de função estável, pois trabalharemos com espaços coerentes, e segundo a proposição 3.4.3, neste “framework” ambos conceitos são equivalentes.

4.4.1.4 Definição (função linear)

Uma função F: $\mathcal{A} \rightarrow \mathcal{B}$ é *linear* se é estável e satisfaz a seguinte condição:

se $X \subseteq \mathcal{A}$, e para todo $b,c \in X \Rightarrow b \cup c \in \mathcal{A}$, então $F(\bigcup X) = \bigcup\{F(b) \mid b \in X\}$, isto é, F comuta com uniões arbitrárias. ■

Observações:

1) Continuidade implica em monotonicidade, assim como a estabilidade também implica em monotonicidade.

2) Considerando os espaços coerentes $\mathcal{A}$ e $\mathcal{B}$ como categorias, nas quais os morfismos de a em a' são inclusões $a \subseteq a'$, então a monotonicidade estabelece que a função F constitui um functor. A continuidade e a linearidade estabelecem que F preserva co-limites dirigidos (*filtered colimits*) e co-limites arbitrários, respectivamente. Estas condições são inteiramente familiares com a abordagem topológica. Entretanto, isto não acontece com a propriedade de estabilidade, que por si só não possui significado topológico óbvio. Em termos de categorias, a propriedade de estabilidade afirma que os "pullbacks" devem ser preservados:

$$
\begin{tikzcd}
& a \cup b & \\
b \arrow[ur] & & a \arrow[ul] \\
& a \cap b \arrow[ul] \arrow[ur] &
\end{tikzcd}
$$

O objetivo é que isto seja válido para qualquer conjunto $\{a_1, a_2, ...\}$ limitado superiormente, não somente os finitos, mas no contexto de aproximação fortemente finita não é necessário salientar este fato. Observa-se neste caso, o fato de que os elementos aproximadores possuem somente uma quantidade finita de elementos abaixo deles, o que não é geralmente verdadeiro na teoria de Scott.

A propriedade de estabilidade força a existência de uma menor aproximação em certos casos, simplesmente tomando-se a interseção de um conjunto que é limitado superiormente.

Segundo [TRO 92] (pag. 95) as funções contínuas, estáveis e lineares têm as seguintes caracterizações:

4.4.1.5 Proposição

Seja uma função F: $\mathcal{A} \to \mathcal{B}$, então temos que:

i) F é *contínua* se e somente se, quando $\alpha \in F(a)$, existe um objeto finito $a_0 \subseteq a$ tal que $\alpha \in F(a_0)$;

ii) F é *estável* se e somente se, quando $\alpha \in F(a)$, existe um menor objeto (necessariamente finito) $a_0 \subseteq a$ tal que $\alpha \in F(a_0)$;

iii) F é *linear* se e somente se, quando $\alpha \in F(a)$, existe $\beta \in a$ tal que $\alpha \in F(\{\beta\})$. ■

A proposição anterior dá uma forma de interpretar os conceitos de continuidade, estabilidade e linearidade em termos do *trace* de uma função, como veremos mais adiante.

A prova de todas as proposições até aqui apresentadas podem ser encontradas em [DIM 94].

4.4.2 Interpretação intuitiva das funções

Os conceitos de função estável e linear são mais recentes, e possivelmente menos intuitivos. Mesmo assim se tentará dar uma interpretação das mesmas. A monotonicidade e continuidade foram amplamente usadas e estudadas, pois acompanham a teoria dos domínios desde sua concepção (ver [STO 79], pag. 96).

4.4.2.1 Monotonicidade

O domínio que interpreta os valores de uma linguagem tem sido ordenado por "conteúdo de informação" dos elementos, o que implica em uma restrição das funções que consideramos válidas em computação. É lógico considerar funções que, quanto mais exata é a informação, mais exato é o resultado, ou em outras palavras, se se conhece mais informação sobre os argumentos de entrada, pode-se querer saber mais (ou pelo menos, não menos) acerca do resultado. Isto é precisamente o que estabelece a monotonicidade. Por outro lado, considerando uma teoria de funções parciais (com um objeto indefinido $\emptyset$ ou $\perp$), a monotonicidade obriga às funções que, para um dado argumento x definido dá o valor indefinido $\perp$, para todo valor y, tal que $y \leq x$, a função aplicada a y também seja indefinido.

4.4.2.2 Continuidade

Como os elementos infinitos não podem ser representados no computador, tem-se que trabalhar com aproximações finitas. Isto não significa que somente se trabalhará com objetos finitos, os objetos infinitos são tratados como limites (ou supremos) de um subconjunto finito de aproximações, que tem alguma representação no computador. Por isto, todos os espaços de valores que é possível computar, devem vir "equipados" com um subconjunto particular contável de elementos a partir dos quais os outros são gerados (base contável). Esta noção de computar um valor por enumeração, restringe as funções permissíveis. Uma função deve produzir uma enumeração dos elementos da base de sua resposta a partir de uma enumeração de aqueles da base dos seus argumentos. Qualquer elemento particular da base da resposta deve ser produzido em tempo finito quando a função tenha recebido só uma parte finita da enumeração do argumento.

Sejam D e D' reticulados completos com bases $E \subseteq D$ e $E' \subseteq D'$ e seja $f{:}D \rightarrow D'$. Seja $X \subseteq E$ um conjunto, que define o $\sup(X)$, que f mapeia em $f(X)$. Precisa-se, portanto, pelo menos um subconjunto $Y' \subseteq E'$ tal que $\sup(Y') = f(\sup(X))$, onde todo $y' \in Y'$ deve ser produzido somente a partir de um subconjunto finito de X. Portanto, impor-se-á a seguinte condição:

$\forall X \subseteq E$, $\exists Y' \subseteq E'$ tal que $\sup(Y') = f(\sup(X))$ e $\forall y' \in Y'$, $y' \leq f(\sup(X_f))$, para algum $X_f \subseteq X$.

Que significa a afirmação anterior? Em primeiro lugar, já que todos os elementos básicos da resposta são produzidos por algum subconjunto finito de elementos da base dos argumentos, a resposta total deve estar incluída no supremo do resultado, obtido a partir de todos estes conjuntos finitos, portanto tem-se a seguinte afirmação:

A condição anterior implica que $\forall X \subseteq E$, $f(\sup(X)) \leq \sup(f(\sup(X_f)))$ tal que $X_f \subseteq X$ e X_f finito.

Pode-se dizer que se f é monotônica, então a afirmação anterior é válida se e somente se também é válida para todo $X \subseteq D$. Portanto f é contínua se e somente se é monotônica e $f(\sup(X)) \leq \sup(f(X))$ (ou equivalentemente, $f(\sup(X)) = \sup(f(X))$).

4.4.2.3 Estabilidade

A estabilidade é melhor entendida considerando a caracterização: f é estável se e somente se, quando $\alpha \in F(a)$, existe um menor objeto (necessariamente finito) $a_0 \subseteq a$ tal que $\alpha \in F(a_0)$. É uma restrição da noção de continuidade, onde se garante que, todo átomo do contradomínio é imagem de um objeto (conjunto) finito do domínio.

4.4.2.4 Linearidade

A linearidade restringe ainda mais as funções, estabelecendo que, para todo âtomo do uma imagem, não só existe um menor objeto finito no domínio, senão que este objeto finito é um átomo. Em outras palavras, os átomos do domínio da função caracterizam totalmente a função: a imagem de qualquer objeto $\{a_1, ..., a_n\}$ do domínio da função é igual à imagem de algum dos conjuntos unitários $\{a_1\}, ..., \{a_n\}$. Considerando a definição, e tendo em conta que a união de conjuntos é o supremo, $f(\{a_1, ..., a_n\}) = \sup(f(\{a_1\}), ..., f(\{a_n\})$.

4.5 Produto dirigido de dois espaços coerentes

A função F de dois argumentos, que mapeia $\mathcal{A}$, $\mathcal{B}$ em $\mathcal{C}$ é estável quando:

i) $a \subseteq a' \in \mathcal{A} \wedge b \subseteq b' \in \mathcal{B} \Rightarrow F(a, b) \subseteq F(a', b')$;

ii) $F(\bigcup_{i \in I}\uparrow a_i, \bigcup_{j \in J}\uparrow b_j) = \bigcup_{(i,j) \in I \times J}\uparrow F(a_i, b_j)$

iii) $a \cup a' \in \mathcal{A} \wedge b \cup b' \in \mathcal{B} \Rightarrow F(a \cap a', b \cap b') = F(a, b) \cap F(a', b')$.

Da mesma maneira se pode definir a estabilidade para qualquer número de argumentos. Observe entretanto, que a continuidade separada é suficiente para a continuidade conjunta, a estabilidade em dois argumentos é equivalente à estabilidade em cada um separadamente, junto com a condição adicional que o "pullback"

$$\begin{tikzcd} & (a',b') & \\ (a',b) \arrow[ur] & & (a,b') \arrow[ul] \\ & (a,b) \arrow[ul] \arrow[ur] & \end{tikzcd}$$

seja preservado.

É possível transformar funções de duas (ou mais) variáveis ao caso de funções unárias. Para isto introduziremos o *produto dirigido* $\mathcal{A}$ & $\mathcal{B}$ de dois espaços coerentes (a notação vem da lógica linear).

4.5.1 Definição (produto dirigido)

Se $\mathcal{A}$, $\mathcal{B}$ são dois espaços coerentes, se define o *produto dirigido* $\mathcal{A}$ & $\mathcal{B}$ como:

$$|\mathcal{A} \,\&\, \mathcal{B}| = |\mathcal{A}| + |\mathcal{B}| = \{1\}\times|\mathcal{A}| \cup \{2\}\times|\mathcal{B}|;$$

onde

$$(1, \alpha) \approx_{\mathcal{A}\&\mathcal{B}} (1, \alpha') \text{ sss } \alpha \approx_{\mathcal{A}} \alpha'$$

$$(2, \beta) \approx_{\mathcal{A}\&\mathcal{B}} (2, \beta') \text{ sss } \beta \approx_{\mathcal{B}} \beta'$$

$$(1, \alpha) \approx_{\mathcal{A}\&\mathcal{B}} (2, \beta) \text{ para todo } \alpha\in|\mathcal{A}| \text{ e } \beta\in|\mathcal{B}|. \blacksquare$$

Em particular, os pontos de $\mathcal{A}$ & $\mathcal{B}$ (subconjuntos coerentes de $|\mathcal{A} \,\&\, \mathcal{B}|$) podem ser escritos unicamente como $\{1\}\times a \cup \{2\}\times b$ com $a\in\mathcal{A}$ e $b\in\mathcal{B}$.

Dada uma função estável F de $\mathcal{A}$. $\mathcal{B}$ em $\mathcal{C}$, define-se uma função G de $\mathcal{A}$ & $\mathcal{B}$ em $\mathcal{C}$ como:

$$G(\{1\}\times a \cup \{2\}\times b) = F(a, b).$$

É imediato que G é estável (pois F é estável); por outro lado, a mesma fórmula define, a partir de uma função unária G, uma função estável binária F, e estas duas transformações são inversas.

4.6 Espaço de Funções

A idéia inicial foi que, "tipo = espaço coerente". Na seção anterior se definiu o produto de espaços coerentes, que corresponde ao produto de tipos, mas que acontece com o espaço de funções? Seria possível definir $\mathcal{A} \to \mathcal{B}$ como o conjunto de funções estáveis de $\mathcal{A}$ em $\mathcal{B}$, mas este não é um espaço coerente. Deve-se, portanto, obter uma representação do conjunto de funções estáveis de tal forma que seja um espaço coerente, isto é, na categoria $\mathrm{COH_S}$, representar o conjunto de morfismos $\mathrm{STAB}(\mathcal{A}, \mathcal{B})$ por objetos $\mathcal{A} \to \mathcal{B}$. Mostrar-se-á também que na categoria $\mathrm{COH_L}$, o conjunto de morfismos $\mathrm{COH_L}(\mathcal{A}, \mathcal{B})$ pode ser representado por objetos $\mathcal{A} \multimap \mathcal{B}$.

4.6.1 Grafos e "Trace's"

4.6.1.1 Definição (grafo)

Toda função contínua F: $\mathcal{A} \to \mathcal{B}$ é determinada por um subconjunto $gr(F)$ de $\mathcal{A}_{\mathrm{fin}} \times B$, o *grafo* de F:

$$gr(F) = \{(a, \beta) \mid a \in \mathcal{A} \text{ e } \beta \in F(a)\},$$

e, contrariamente, qualquer conjunto $X \subseteq \mathcal{A}_{\mathrm{fin}} \times B$ tal que

$$(a, \beta) \in X,\ a \subseteq a' \in \mathcal{A} \Rightarrow (a', \beta) \in X$$

$$(a, \beta), (a, \beta') \in X \Rightarrow \beta \approx \beta',$$

determina uma função contínua F_X: $\mathcal{A} \to \mathcal{B}$ dada por:

$$F_X(a) = \{\beta \mid \exists a' \subseteq a, (a', \beta) \in X\}. \blacksquare$$

4.6.1.2 Definição (trace)

Toda função estável F: $\mathcal{A} \to \mathcal{B}$ é determinada por um subconjunto $tr(F)$ de $gr(F)$, o *trace* de F:

$$tr(F) = \{(a, \beta) \mid a \in \mathcal{A} \text{ é o menor conjunto tal que } \beta \in F(a)\}.$$

Por outro lado, $tr(F)$ é o conjunto $X \subseteq \mathcal{A}_{\text{fin}} \times B$ tal que

$$(a, \beta), (a', \beta') \in X, a \cup a' \in \mathcal{A} \Rightarrow \beta \approx \beta'$$

$$(a, \beta), (a', \beta) \in X, a \cup a' \in \mathcal{A} \Rightarrow a = a',$$

e, contrariamente, qualquer conjunto X que cumpre o anterior, define uma função estável $F_{st(X)}$: $\mathcal{A} \rightarrow \mathcal{B}$ dada por:

$$F_{st(X)}(a) = \{\beta \mid \exists a' \subseteq a, (a', \beta) \in X\}. \blacksquare$$

4.6.1.3 Definição (trace linear)

Seja F: $\mathcal{A} \rightarrow \mathcal{B}$ uma função linear. Então todos os elementos (a, β) de $tr(F)$ têm a forma $(\{\alpha\}, \beta)$, portanto define-se o *trace linear* de F como

$$ltr(F) = \{(\alpha, \beta) \mid \beta \in F(\{a\})\},$$

que é um conjunto $X \subseteq A \times B$ tal que

$$(\alpha, \beta), (\alpha', \beta') \in X, \alpha \approx \alpha' \Rightarrow \beta \approx \beta'$$

$$(\alpha, \beta), (\alpha', \beta) \in X, \alpha \approx \alpha' \Rightarrow \alpha = \alpha'.$$

Contrariamente, qualquer conjunto X que satisfaz as condições anteriores, determina uma função linear $F_{lin(X)}$: dada por:

$$F_{lin(X)}(a) = \{\beta \mid \exists \alpha \in a, (\alpha, \beta) \in X\}. \blacksquare$$

Foram definidos o *grafo* e *trace* de funções estáveis e lineares. Os primeiros dão lugar à seguinte proposição (ver [GIR 89], pag. 64):

4.6.1.4 Proposição

Se F tem seus valores nas funções estáveis de $\mathcal{A}$ em $\mathcal{B}$, seus traces dão os pontos de um espaço coerente: que se escreverá $\mathcal{A} \rightarrow \mathcal{B}$. $\blacksquare$

4.6.2 A ordem de Berry (ordem estável)

Sendo um espaço coerente, $\mathcal{A} \rightarrow \mathcal{B}$ é naturalmente ordenado por inclusão. A bijeção entre $\mathcal{A} \rightarrow \mathcal{B}$ e as funções estáveis de $\mathcal{A}$ em $\mathcal{B}$, induzem, então, uma nova relação de ordem:

4.6.2.1 Definição (ordem estável)

Sejam F e G funções estáveis, então define-se a *ordem estável* como

$$F \leq_S G \quad \text{sss} \quad \forall a',a \in \mathcal{A},\ a' \subseteq a \Rightarrow F(a') = F(a) \cap G(a'). \blacksquare$$

A ordem estável dá lugar à seguinte proposição:

4.6.2.2 Proposição

A ordem definida no espaço coerente $\mathcal{A} \rightarrow \mathcal{B}$ (a inclusão), corresponde à ordem estável em $COH_S(\mathcal{A}. \mathcal{B})$, isto é,

$$F \leq_S G \quad \text{sss} \quad tr(F) \subseteq tr(G). \blacksquare$$

4.7 Linearização de uma função estável

Nesta seção se verão alguns exemplos do processo de linearização de uma função estável (ver [GIR 89], pag. 101). Define-se agora o conceito de espaço coerente “of course”.

4.7.1 Definição (espaço coerente “of course”)

Seja D um espaço coerente, definem-se o espaço coerente *of course* de D, denotada por $!D$, e a sua teia como

$$|!D| = D_{\text{fin}} = \{\ x \in D \mid x \text{ é finito}\}$$

$$x_1 \approx x_2 \ (\text{mod } !D) \text{ sss } x_1 \cup x_2 \in D.$$

A função básica associada com $!D$ é

oc: $x \mapsto !x = \{x_0 \mid x_0 \subseteq x \text{ e } x_0 \text{ é finito}\}$

de D em $!D$. ■

A função *oc* é estável, mas não é linear. O ponto interessante desta definição é que, dados dois espaços coerentes D e E, $D \rightarrow E$ é equivalente a $!D \multimap E$. Em outras palavras, mudando o domínio da função pelo "of course", toda função estável tem uma correspondente função linear.

4.7.2 Definição (função estável-linear a partir de uma estável)

Se f é uma função estável de D em D', define-se uma função estável-linear $Lin(f)$ de $!D$ em D como $Trlin(Lin(f)) = Tr(f)$. Temos que

$$Lin(f)(!x) = f(x).$$

Ou seja, se $\beta \in f(x)$, então, para algum $x_0 \subseteq x$, temos que $(x_0, \beta) \in Tr(f) = Trlin(Lin(f))$, mas $x_0 \in !x$, portanto $\beta \in Lin(f)(!x)$. Similarmente, se $\beta \in Lin(f)(!x)$, vê-se que $\beta \in f(x)$. ■

4.7.3 Definição (função estável a partir de uma linear)

Se g é uma função linear de $!D$ em D', se define uma função estável $Delin(g)$ de D em D' como

$$Delin(g)(x) = g(!x).$$ ■

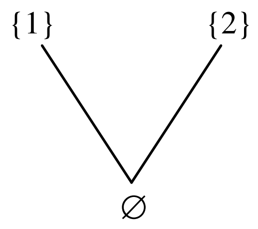
Pode-se mostrar que *Lin* e *Delin* são funções mutuamente inversas, e que $Lin(f)(!x) = f(x)$ caracteriza $Lin(f)$.

4.7.4 Exemplo

Sejam $D = \{\emptyset, \{1\}, \{2\}\}$ um espaço coerente, e a teia dada por

$$|D| = \{1, 2\} \text{ tal que } 1\#2,$$

que dá o seguinte diagrama



Seja oc: $D \to !D$ definida como

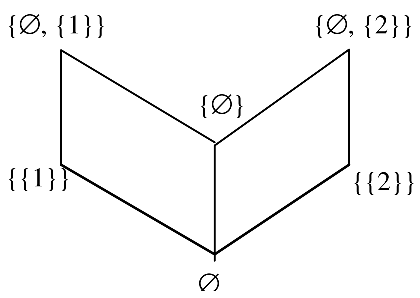
$$oc(x) = !x = \{x_0 \mid x_0 \subseteq x \text{ e } x_0 \text{ é finito}\}$$

que gera a teia e o espaço coerente $!D$ como segue:

$$|!D| = \{\varnothing, \{1\}, \{2\}\} \text{ onde } \varnothing \approx \{1\} \text{ e } \varnothing \approx \{2\},$$

$$!D = \{\varnothing, \{\varnothing\}, \{\{1\}\}, \{\{2\}\}, \{\varnothing, \{1\}\}, \{\varnothing, \{2\}\}\},$$

que tem o seguinte grafo



O trace é o seguinte:

$$tr(oc) = \{(\varnothing, \{\varnothing\}), (\{1\}, \{\varnothing, \{1\}\}), (\{2\}, \{\varnothing, \{2\}\})\}.$$

Seja $D' = \{0, 1\}$ (onde 0 é o "bottom" e 1 é o "top") e seja f: $D \to D'$ tal que

$$tr(f) = \{ (\varnothing, 0), (\{1\}, 1), (\{2\}, 1)\}.$$

Claramente f é estável.

Obtém-se agora $Lin(f)$: $!D \to D$ e se mostra que é linear. Pela definição temos que $Lin(f)(!x) = f(x)$, e no exemplo:

$Lin(f)(\{\emptyset\}) = 0$

$Lin(f)(\{\emptyset, \{1\}\}) = f(\{1\}) = 1$

$Lin(f)(\{\emptyset, \{2\}\}) = f(\{2\}) = 1$

Resta ainda definir $Lin(f)$ para os argumentos $\emptyset$, $\{\{1\}\}$ e $\{\{2\}\}$. Sabe-se que $Lin(f)$ é linear, portanto uma forma construtiva de estender a função para todos os argumentos seria considerar

$$Lin(f)(\{a_1,..., a_n\}) = \bigcup\{f(a_1),..., f(a_n)\}.$$

Assim, no exemplo, se tem que

$Lin(f)(\emptyset) = \bigcup\{f(\emptyset)\} = f(\emptyset) = 0$

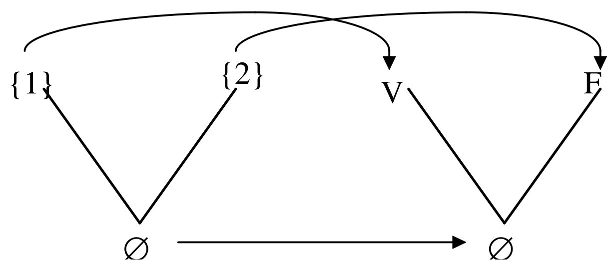
$Lin(f)(\{\{1\}\}) = \bigcup\{f(\{1\})\} = f(\{1\}) = 1$

$Lin(f)(\{\{2\}\}) = \bigcup\{f(\{2\})\} = f(\{2\}) = 1$

Claramente $Lin(f)$ assim definida é linear (basta considerar todos os casos e mostrar que $Lin(f)(x \cup y) = Lin(f)(x) \cup Lin(f)(y)$).

<u>Exemplo 2</u>:

Seja D como no exemplo anterior, e $D' = \{\perp, V, F\}$, o domínio *flat* dos booleanos. Seja f: $D \to D'$ tal que



ou seja,

$tr(f) = \{ (\emptyset, \perp), (\{1\}, V), (\{2\}, F)\}$.

Claramente f é estável.

Da definição de $Lin(f)$ temos que:

$Lin(f)(\{\emptyset\}) = f(\emptyset) = \perp$

$Lin(f)(\{\emptyset, \{1\}\}) = f(\{1\}) = V$

$Lin(f)(\{\emptyset, \{2\}\}) = f(\{2\}) = F$

$Lin(f)(\emptyset) = \bigcup\{f(\emptyset)\} = f(\emptyset) = \perp$

$Lin(f)(\{\{1\}\}) = \bigcup\{f(\{1\})\} = f(\{1\}) = V$

$Lin(f)(\{\{2\}\}) = \bigcup\{f(\{2\})\} = f(\{2\}) = F$

Facilmente pode ver-se que $Lin(f)$ é linear.

4.7.5 Exemplo

Seja $D = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}$ um espaço coerente, e a teia dada por

$|D| = \{1, 2\}$ tal que $1 \approx 2$.

A teia do espaço coerente $!D$ é dado por

$|!D| = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}$

com $\{1\} \approx \{2\}$

$\{1\} \approx \{1, 2\}$

$\{2\} \approx \{1, 2\}$

$\emptyset \approx \{1\}$

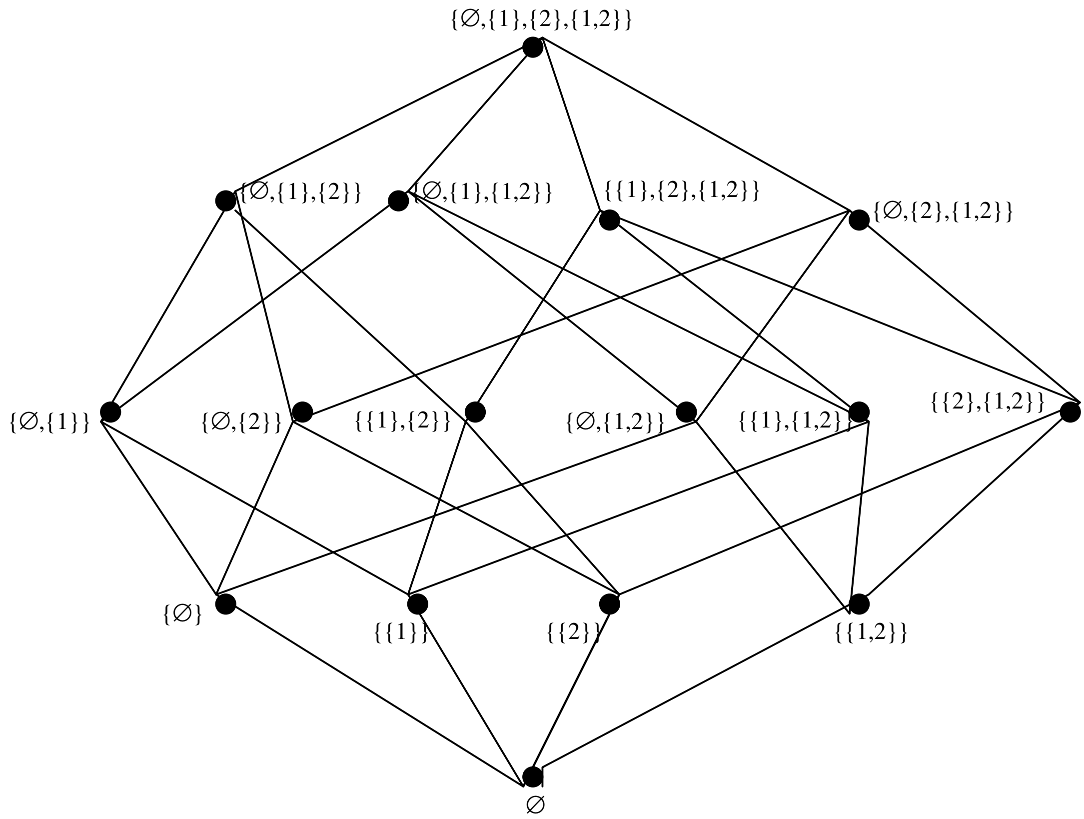
$\emptyset \approx \{2\}$

$\emptyset \approx \{1, 2\}$,

que gera o seguinte espaço coerente:

!D = { ∅, {∅}, {∅, {1}}, {∅, {2}}, {∅, {1, 2}}, {{1}}, {{2}}, {{1, 2}}, {{1}, {2}}, {{1}, {1,2}}, {{2}, {1, 2}}, {{1}, {2}, {1, 2}}, {∅, {1}, {2}}, {∅, {1},{1,2}}, {∅, {2}, {1, 2}}, {∅, {1}, {2}, {1, 2}} }.

O grafo do espaço coerente *!D* definido acima é representado na seguinte figura:



Seja *D'* = {0, 1}, como no exemplo 1. Seja *f*: *D* → *D'* com o seguinte *trace*

$$tr(f) = \{(\emptyset, 0), (\{1\}, 0), (\{2\}, 0), (\{1, 2\}, 1)\}$$

Claramente *f* é estável.

Definindo *Lin*(*f*) como antes, temos que

$Lin(f)(\{\emptyset\}) = f(\emptyset) = 0$

$Lin(f)(\{\varnothing, \{1\}\}) = f(\{1\}) = 0$

$Lin(f)(\{\varnothing, \{2\}\}) = f(\{2\}) = 0$

$Lin(f)(\{\varnothing, \{1\}, \{2\}, \{1, 2\}\}) = f(\{1, 2\}) = 1$

$Lin(f)(\varnothing) = \bigcup\{f(\varnothing)\} = f(\varnothing) = 0$

$Lin(f)(\{\{1\}\}) = \bigcup\{f(\{1\})\} = f(\{1\}) = 0$

$Lin(f)(\{\{2\}\}) = \bigcup\{f(\{2\})\} = f(\{2\}) = 0$

$Lin(f)(\{\{1, 2\}\}) = \bigcup\{f(\{1, 2\})\} = f(\{1, 2\}) = 1$

$Lin(f)(\{\{1\}, \{2\}\}) = \bigcup\{ f(\{1\}), f(\{2\})\} = \bigcup\{0, 0\} = 0$

$Lin(f)(\{\varnothing, \{1, 2\}\}) = \bigcup\{ f(\varnothing), f(\{1, 2\})\} = \bigcup\{0, 1\} = 1$

$Lin(f)(\{\{1\}, \{1, 2\}\}) = \bigcup\{ f(\{1\}), f(\{1, 2\})\} = \bigcup\{0, 1\} = 1$

$Lin(f)(\{\{2\}, \{1, 2\}\}) = \bigcup\{ f(\{2\}), f(\{1, 2\})\} = \bigcup\{0, 1\} = 1$

$Lin(f)(\{\varnothing, \{1\}, \{2\}\}) = \bigcup\{ f(\varnothing), f(\{1\}), f(\{2\})\} = \bigcup\{0, 0, 0\} = 0$

$Lin(f)(\{\varnothing, \{1\}, \{1, 2\}\}) = \bigcup\{ f(\varnothing), f(\{1\}), f(\{1, 2\})\} = \bigcup\{0, 0, 1\} = 1$

$Lin(f)(\{\varnothing, \{2\}, \{1, 2\}\}) = \bigcup\{ f(\varnothing), f(\{2\}), f(\{1, 2\})\} = \bigcup\{0, 0, 1\} = 1$

$Lin(f)(\{\{1\}, \{2\}, \{1, 2\}\}) = \bigcup\{ f(\{1\}), f(\{2\}), f(\{1, 2\})\} = \bigcup\{0, 0, 1\} = 1$

Considere-se a estabilidade e a linearidade para um exemplo:

Sejam $a = \{\varnothing, \{1\}, \{1, 2\}\}$ e $b = \{\{2\}, \{1, 2\}\}$

- estabilidade:

$a \uparrow b \Rightarrow$

$Lin(f)(a \wedge b) = Lin(f)(\{\{1, 2\}\}) = 1$, e

$Lin(f)(a) \wedge Lin(f)(b) = 1 \wedge 1 = 1$, portanto é estável.

- linearidade:

$Lin(f)(a \vee b) = Lin(f)(\{\varnothing, \{1\}, \{2\}, \{1, 2\}\}) = 1$, e

$Lin(f)(a) \vee Lin(f)(b) = 1 \vee 1 = 1$,

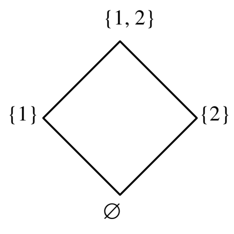
portanto é linear.

É possível testar as condições de estabilidade e linearidade para todos os casos.

4.8 Uma interpretação computacional de *!D*

Os exemplos vistos anteriormente ilustram uma nova interpretação computacional de *!D*, onde *D* é um espaço coerente. Pode-se observar que a imagem da função *oc* é um conjunto que, considerado como uma estrutura com a inclusão como ordem, dá um conjunto com todas as estratégias de computação do argumento. A pergunta é: que significam os outros objetos de *!D* que não são imagem de nenhum objeto de *D*? Para entender melhor esta interpretação considere-se o exemplo a seguir.

Seja *D* como no exemplo 4.7.5; assim $oc(\{1, 2\}) = \{\varnothing, \{1\}, \{2\}, \{1, 2\}\}$, que ordenado por inclusão dá o espaço coerente



Mas, que significa isto? Pode-se interpretar como: para computar o elemento $\{1, 2\}$ se tem que computar o elemento $\varnothing$, depois o $\{1\}$ e paralelamente o $\{2\}$, ou então o $\{1\}$ e depois o $\{2\}$, ou o $\{2\}$ e depois o $\{1\}$. Isto é claro nos objetos totais, mas, que acontece com os objetos parciais? Seja $oc(\{2\}) = \{\varnothing, \{2\}\}$, que é isomorfo ao domínio de Sapiensky (com um elemento "bottom" e um "top"). Como antes, se tem uma estratégia para computar o objeto parcial {2}. Considere-se agora o que acontece com os objetos de *!D* que não são imagem de nenhum objeto de *D*. Considerem-se por exemplo {{1,2}} ou {{1}, {2}}, ou ainda {∅, {2},{1, 2}}. Todos eles são informações parciais para a computação do objeto total {1, 2}: em outras palavras, estão "construindo" a estratégia.

A próxima pergunta é: que significa *Lin*(*f*)? Claramente a informação que *Lin*(*f*) dá é uma informação parcial, quando aplicada a argumentos parciais, sobre o que ela pode inferir de seus argumentos acerca do possível resultado da computação. O resultado pode ser considerado "exato" (não aproximado) unicamente quando o objeto é total.

Seja por exemplo *Lin*(*f*)({{1}, {2}}) = 0 (o elemento bottom do domínio), o que diz que no processo de computação, por enquanto, o resultado é 0 (pode ser que dê 1, mas "ainda" não está definido). Seja por exemplo *Lin*(*f*)({∅, {1}}) = 0. Interpretando como antes, pode-se dizer que, computar ∅ primeiro e depois {1} implica, por enquanto, no resultado 0. Considere-se agora *Lin*(*f*)({{2}, {1, 2}}) = 1, que diz que se pode ter certeza que, qualquer estratégia de computação a partir do estado {{2}, {1, 2}} vai dar 1, (pois 1 é o supremo do contradomínio).

O anterior sugere, de certa forma, a idéia de um "interpretador", que tem como argumento uma função e os argumentos, e dá como valor o resultado da aplicação da função assim como a estratégia de computação para computá-la. No capítulo 6 se verá a definição formal destes conceitos.

Como se trabalhará com cds (seguindo [CUR 86]), se apresenta um exemplo do anterior em cds. É importante lembrar que é uma cds: uma cds é uma quaterna ($C, V, E, \vdash$) de células, valores, eventos e uma relação de habilitação tal que

$E \subseteq C \times V$ e

$\forall c \in C, \exists v \in V$ tal que $(c, v) \in E$ ("toda célula pode ser preenchida")

Ainda mais, $\forall c \in C, \vdash c$, pois se estão considerando unicamente cds que são teias de um espaço coerente.

Intuitivamente, dado um espaço coerente D, os objetos do espaço coerente $!D$ são conjuntos que representam os "passos de computação" para computar os objetos de D. Sejam os seguintes exemplos:

4.8.1 Exemplo

Seja $M = (C, V, E, \vdash)$ uma cds, onde

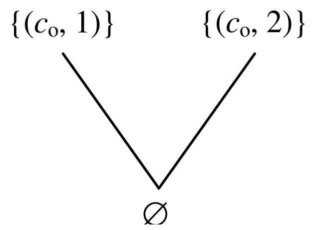
$C = \{c_0\}$

$V = \{1,2\}$

$E = \{(c_0, 1), (c_0, 2)\}$

$\vdash c_0$

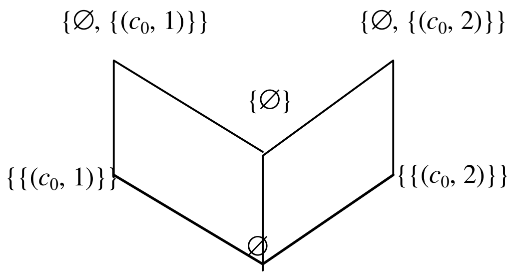
Portanto o conjunto de estados $D \equiv D(M)$ (gerado por M) é definido como o conjunto $\{\varnothing, (c_0,1), (c_0, 2)\}$, que dá o seguinte grafo



O espaço coerente $!D(M)$ é o seguinte:

$!D(M) \equiv\ !D = \{\varnothing, \{\varnothing\}, \{\{(c_0, 1)\}\}, \{\{(c_0, 2)\}\}, \{\varnothing, \{(c_0, 1)\}\}, \{\varnothing, \{(c_0, 2)\}\}\}$,

que é representado como



4.9 Categorias de espaços coerentes

No contexto em que se está trabalhando, existem essencialmente três categorias que têm como objetos espaços coerentes: a que têm como morfismos as funções contínuas; aquela cujos morfismos são funções estáveis; e finalmente a categoria cujos morfismos são as funções lineares. Sabe-se que (ver [TRO 92]):

4.9.1 Proposição

A categoria dos espaços coerentes e funções contínuas (COH) não é cartesiana fechada. ■

4.9.2 Proposição

A categoria dos espaços coerentes e funções estáveis (COH_S) é cartesiana fechada.■

4.9.3 Proposição

A categoria dos espaços coerentes e funções lineares (COH_L) não é cartesiana fechada. ■

5 ESPAÇOS COERENTES COMO ESTRUTURAS DE EVENTOS E CDS

Neste capítulo, se verá a relação dos espaços coerentes com os domínios de eventos e domínios concretos, assim como a relação entre a parte “concreta” destes domínios: teias de espaços coerentes com estruturas de eventos e estruturas de dados concretas (cds). Na primeira seção se mostra a relação de varias categorias. Já que Zhang apresenta somente a relação a nível de categorias, mas não dá uma caracterização das mesmas, na seção 5.2 ver-se-á a relação dos espaços coerentes com as estruturas de eventos e cds (que é a que realmente nos interessa).

Segundo Zhang (ver [ZHA 89], pag. 156), um espaço coerente pode ser visto como caso particular de dI-domínios[11], e de estruturas de eventos. Como dI-domínio, um espaço coerente é um domínio D, que é coerente, no sentido que: tem supremos para todo par de conjuntos compatíveis, e d é primo completo se e somente se $d \neq \bot_D \wedge (x < d \Rightarrow x = \bot_D)$. Como estrutura de eventos estável, um espaço coerente é uma estrutura da forma (E, Con, $\{\varnothing \vdash e \mid e \in E\}$), onde Con é determinada por uma relação de conflito.

No que diz respeito à segunda “caracterização” dos espaços coerentes (relacionados com estruturas de eventos), se mostra neste capítulo que a relação é com os domínios de eventos (a parte abstrata das estruturas de eventos), e não precisamente com as estruturas de eventos como afirma Zhang.

[11] Um dI-domínio é um domínio consistentemente completo D que satisfaz o axioma d (distributividade) e o axioma I:
axioma d) $\forall x,y,z \in D,\ y \uparrow z \Rightarrow x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$;
axioma I) $\forall d \in D^0,\ |\{x \mid x \leq d\}| < \infty$.

5.1 Relação de algumas categorias de domínios

Sejam as seguintes categorias de domínios:

CATEGORIAS	OBJETOS	MORFISMOS
DI	dI-domínios	funções estáveis
DL	dI-domínios	funções lineares
SEV_{AS}	Estruturas de Eventos Estáveis	funções estáveis
SEV_{A}	Estruturas de Eventos Estáveis	funções lineares
SEV_{Syn}*	Estruturas de Eventos Estáveis	morfismos parcialmen. sincrônicos
SEV_{Syn}	Estruturas de Eventos Estáveis	morfismos sincrônicos
SF_{S}	Famílias Estáveis	funções estáveis
SF_{L}	Famílias Estáveis	funções lineares
COH_{S}	Famílias (Espaços) Coerentes	funções estáveis
COH_{L}	Famílias (Espaços) Coerentes	funções lineares
FF	Famílias Finitas	mapeamentos lineares

As categorias DI, SEV_{AS}, SF_{S} e COH_{S} são categorias cartesianas fechadas (CCC), as categorias DL, SEV_{A}, SF_{L}, COH_{L} e FF são categorias cartesianas monoidais[12] (MCC), entretanto que para as categorias SEV_{Syn}* e SEV_{Syn} não se apresenta este resultado.

A relação entre estas categorias é representada na Figura 5.1, onde:

[12]A diferença entre a categoria cartesiana monoidal e a categoria cartesiana fechada é que, a primeira não é fechada com respeito ao produto cartesiano, mas sim com respeito ao produto tensorial (ver capítulo 7 de [Zha 89]).

1: subcategoria completa (full-subcategory)
2: equivalentes
3: adjunção
4: subcategoria
5: co-reflexão
?: não conhecida.

$$
\begin{tikzcd}
COH_S \arrow[r, "1"] \arrow[d, leftrightarrow, "3"'] & SF_S \arrow[r, leftrightarrow, "2"] \arrow[d, leftrightarrow, "3"'] & DI \arrow[r, leftrightarrow, "2"] \arrow[d, leftrightarrow, "3"'] & SEV_S \arrow[d, leftrightarrow, "3"'] \\
COH_L \arrow[r, "1"] \arrow[dr, leftrightarrow, "5?"] & SF_L \arrow[r, leftrightarrow, "2"] & DL \arrow[r, leftrightarrow, "2"] & SEV_L \\
 & FF \arrow[ur, leftrightarrow, "5"'] \arrow[rr, leftrightarrow, "?"] & & SEV_{Syn}^{*} \arrow[u, "4"] \\
 & & & SEV_{Syn} \arrow[u, "4"]
\end{tikzcd}
$$

Figura 5.1 - Relação de algumas categorias de domínios

5.2 Espaços coerentes, domínios de eventos e domínios concretos

Nesta seção se dá uma caracterização dos espaços coerentes como domínios de eventos e domínios concretos. Conseqüentemente, as teias de espaços coerentes serão caracterizadas como estruturas de eventos e cds. Droste (ver [DRO 89]) define os domínios de evento como:

5.2.1 Definição (domínio de eventos de Droste)

Um *domínio de eventos de Droste*[13] é uma cpo algébrica $(D, \leq)$ que satisfaz as seguintes condições para todo $x, x', y, y', z \in D^0$:

(F) $\{y \in D \mid y \leq x\}$ é finito;
(C) se $x \prec y$, $x \prec z$, $z \neq y$ e $y \uparrow z$, então existe $y \vee z$, e $y \prec y \vee z$, $z \prec y \vee z$;
(I) $[x, x'] \succ\!\prec [y, y'] \wedge x \leq y \Rightarrow x' \leq y'$. ■

5.2.2 Proposição

Todo domínio de eventos é um domínio de eventos de Droste. ■

Uma prova desta proposição é apresentada em [DRO 89] (ver pag. 43).

5.2.3 Proposição

Seja $(E, \#, \vdash)$ uma estrutura de eventos tal que $\forall e \in E, \vdash e$. Então, $(E, \approx)$ é a teia de um espaço coerente, onde

$$\forall e_1, e_2 \in E,\ e_1 \approx e_2 \text{ sss } \neg(e_1 \# e_2).$$

Prova)

Uma teia é definida como um conjunto munido de uma relação reflexiva e simétrica, que implica que o espaço coerente correspondente tem como átomos todos os conjuntos unitários. No domínio de eventos canônico associado com a estrutura de eventos $(E, \#, \vdash)$, não contém necessariamente todos os conjuntos unitários da forma $\{e\}$ (para todo evento $e \in E$), pois dependerá da relação de habilitação $\vdash$. Por isto a condição de que $\forall e \in E, \vdash e$. Portanto, dada a estrutura de eventos $(E, \#, \vdash)$, a única coisa que temos que provar, é que "$\approx$" é reflexiva e simétrica.

[13] O que neste trabalho se define como *domínio de eventos de Droste*, Droste ([DRO 89], pag. 40) chama simplesmente de domínio de eventos, entretanto que o que neste trabalho foi definido como *domínio de eventos*, no capítulo 2 (ver também [CUR 86], pag. 133), Droste chama de *domínio de eventos de conflito* ([DRO 89], pag. 43).

Reflexividade: $\forall e_1 \in E$, $\neg(e_1 \# e_1)$, pois se $e_1 \in x$, e $e_1 \# e_1$, então por definição de #, $e_1 \notin x$, o que é uma contradição. Portanto $e_1 \approx e_1$.

Simetria: $\forall e_1, e_2 \in E$, $e_1 \approx e_2$ sss $\neg(e_1 \# e_2)$ sss $\neg(e_2 \# e_1)$ sss $e_2 \approx e_1$.

Dos resultados anteriores, $(E, \approx)$ é uma teia de um espaço coerente. ■

5.2.4 Proposição

Seja $(D, \subseteq)$ um domínio de eventos. Então, $(D, \subseteq)$ é um espaço coerente se e somente se D cumpre a seguinte propriedade (de fecho inferior):

(dc) $x \in D, x' \subseteq x \Rightarrow x' \in D$.

Em outras palavras: (F), (C), (R), (V), (dc) $\Rightarrow$ (bc).

***Prova*)**

Intuitivamente: $\vdash e$ é a condição para que uma uma estrutura de eventos seja uma teia de um espaço coerente, portanto o domínio de eventos é tal que $\forall e$, $\{e\} \in D$. Como (dc) implica em $\vdash e$, então se deveria poder deduzir a propriedade de (bc) a partir de D. ■

Como corolário da proposição anterior, temos que D deve conter todos os conjuntos unitários (átomos do espaço coerente).

5.2.5 Teorema

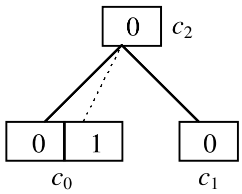
Seja D um domínio de eventos tal que seja um espaço coerente. Então, a estrutura de eventos canônica obtida pelo teorema de Winskel (teorema 3.3.5) é uma teia do espaço coerente.

***Prova*)**

Seja $[x, x']_{\succ\prec}$, portanto $x' = x \cup \{e\}$, e pela propriedade (dc) de espaços coerentes, $\{e\} \in D$. Isto implica $\vdash e$, onde e é qualquer evento. Mas, pela definição de Winskel $\forall e$, $e = [x, x']_{\succ\prec}$, para algum $x, x' \in D^0$, portanto $\forall e$, $\vdash e$. Então, pela proposição 5.2.3, a estrutura de eventos gerada pelo domínio de eventos (espaço coerente) é uma teia. ■

Observação: O resultado é intuitivo, pois os domínios de eventos (em particular, os domínios concretos) que não cumprem $\forall e, \ \vdash e$, falham na propriedade (dc) (fecho inferior) dos espaços coerentes (usada para mostrar a proposição anterior), como se pode ver no seguinte exemplo (em domínios concretos), dado por Curien & Berry em [BER 82] (pag. 271):

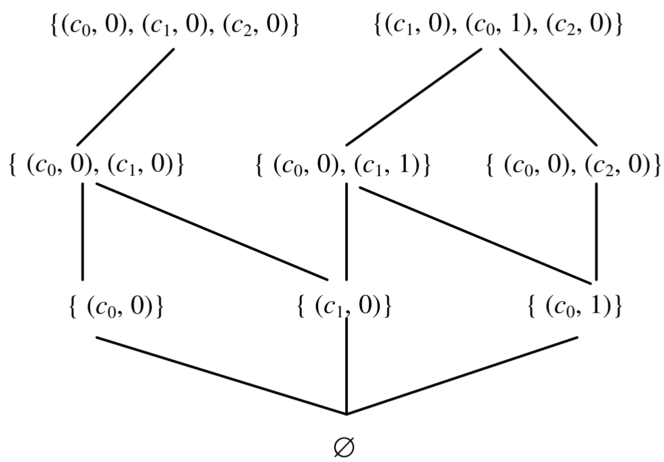
Seja a seguinte cds $\mathbb{Q}$:



onde

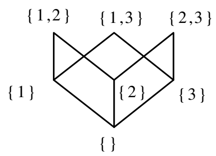
$$\begin{array}{rl} & \vdash c_0 \\ & \vdash c_1 \\ (c_0, 0), (c_1, 0) & \vdash c_2 \\ (c_0\ 1) & \vdash c_2. \end{array}$$

O conjuntos de estados é o seguinte domínio concreto:



que não é um espaço coerente, pois não cumpre a propriedade (dc) (o subconjunto $\{(c_1, 0), (c_2, 0)\}$ deveria ser um estado), claramente devido a que c_2 é habilitado por conjuntos de eventos diferentes de $\varnothing$ ($(c_0, 0), (c_1, 0) \vdash c_2$ e $(c_0\ 1) \vdash c_2$).

Um exemplo de um domínio, que não cumpre a propriedade de completeza binária (bc) é seguinte domínio: $E = \{1,2,3\}$, com $\forall e \in E, \ \vdash e$:



Não é um espaço coerente pois não cumpre a propriedade (bc): se $M = \{1,2,3\} \subseteq E$, $\{1\}\cup\{2\}\in E$, $\{2\}\cup\{3\}\in E$, $\{1\}\cup\{3\}\in E$, mas $\cup M = \{1,2,3\} \notin E$, portanto a proposição 3 não pode ser aplicada. Mas, pela proposição 1, se poderia obter uma teia, pois $\forall e \in E, \ \vdash e$ (o qual seria um absurdo!). Neste domínio temos 3 classes de equivalência de intervalos primos:

$1 \equiv \{[\varnothing,\{1\}], [\{2\},\{1,2\}], [\{3\},\{1,3\}]\} = [\varnothing,\{1\}]_{\succ\prec}$

$2 \equiv \{[\varnothing,\{2\}], [\{1\},\{1,2\}], [\{3\},\{2,3\}]\} = [\varnothing,\{2\}]_{\succ\prec}$

$3 \equiv \{[\varnothing,\{3\}], [\{2\},\{2,3\}], [\{1\},\{1,3\}]\} = [\varnothing,\{3\}]_{\succ\prec}$

Claramente é um domínio de eventos de Droste (cumpre as propriedades (F), (C) e (I)), mas não é um domínio de eventos, pois temos, por exemplo, que $[\varnothing,\{1\}] \succ\prec [\{2\},\{1,2\}]$, $[\varnothing,\{3\}] \succ\prec [\{2\},\{2,3\}]$ e $\{1\}\uparrow\{3\}$; mas isto não implica que $\{1,2\}\uparrow(\{2,3\}$; logo, não cumpre a propriedade (V), portanto não é um domínio de eventos.

O exemplo anterior mostra porque é necessário pedir na proposição 5.2.4 que os domínios de eventos tenham a propriedade (dc) para serem espaços coerentes.

5.2.6 Proposição

(C) $\wedge$ (Q) $\wedge$ (dc) $\Rightarrow$ (bc).

Prova)

Segundo Khan & Plotkin ([KHA 93], pag. 216), o seguinte domínio não satisfaz a propriedade (Q):

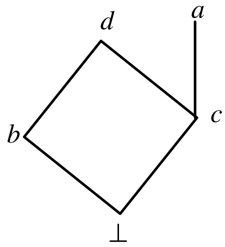
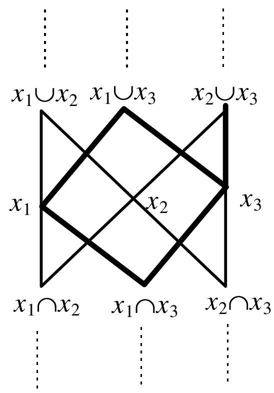


Figura 5.2- Domínio que não tem a prop. (Q)

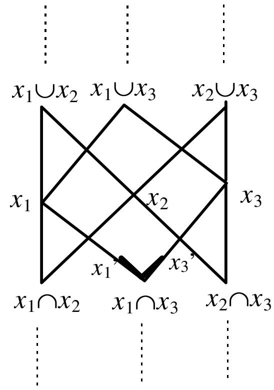
Suponha, por absurdo, que a propriedade (bc) não é satisfeita. Isto é, se $(E, \subseteq)$ é um

domínio tal que $M \subseteq E$, $\forall x_1, x_2 \in M$, $x_1 \cup x_2 \in E$ e $\bigcup M \notin E$.

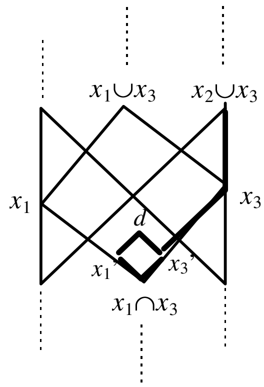
Vejamos que "aspecto" tem tal domínio. Por exemplo, seja $M = \{x_1, x_2, x_3\}$. Pela propriedade (dc), $\exists$ $x_1 \cap x_2$, $x_1 \cap x_3$ e $x_2 \cap x_3$, portanto o domínio tem a seguinte forma:



onde a parte em negrito é o domínio da Figura 5.2. Para que a hipótese da propriedade seja satisfeita, se deve ter um x_1', x_3' tal que $x_1' \cap x_3 \prec x_1'$, onde $x_1 \cap x_3 = x_1' \cap x_3 \prec x_1' \subseteq x_1$. Da mesma forma, se tem que $= x_1 \cap x_3 \prec x_3' \subseteq x_3$. Ou seja,



Pela condição (C), já que $x_1 \cap x_3 \prec x_1$', $x_1 \cap x_3 \prec x_3$', $x_1' \neq x_3$ e $x_1' \uparrow x_3$', então $\exists\ d = x_1' \cup x_3$' e $x_1' \prec x_1' \cup x_3$', $x_3' \prec x_1' \cup x_3$':



Portanto, se tem que x_1' # $x_2 \cup x_3$, e $x_1' \cap (x_2 \cup x_3) = x_1 \cap x_3 \prec x_1$', mas não existe um elemento t tal que t # x_1' e $x_1' \cap (x_2 \cup x_3) \prec t \leq x_2 \cup x_3$, pois $t = x_3$' e $x_1' \uparrow x_3$', o que contradiz a propriedade . Tal situação é possível porque $\bigcup M \notin E$, portanto $x_1 \cup x_2 \cup x_3 = (x_1 \cup x_3) \cup (x_2 \cup x_3) \notin E$, o que implica que $(x_1 \cup x_3)$ deve ser compatível com $(x_2 \cup x_3)$, isto é $x_1 \cup x_2 \cup x_3 \in E$.

Claramente a demonstração pode ser estendida a subconjuntos de E com cardinalidade maior que três. Seguindo a mesma construção anterior, sempre é possível obter uma estrutura como na Figura 5.2. ■

Ver-se-á agora qual é a condição que deve cumprir um domínio concreto para ser um espaço coerente. Os domínios da Figura 5.3 são domínios concretos, mas não são espaços coerentes. A Figura 5.3(a) falha na propriedade (bc), entretanto que o domínio da Figura 5.3(b) falha na propriedade (dc). Na proposição anterior se viu que (C) $\wedge$ (Q) $\wedge$ (dc) $\Rightarrow$ (bc), portanto para que um domínio concreto seja um espaço coerente, seria natural pedir que cumpra somente a propriedade (dc).

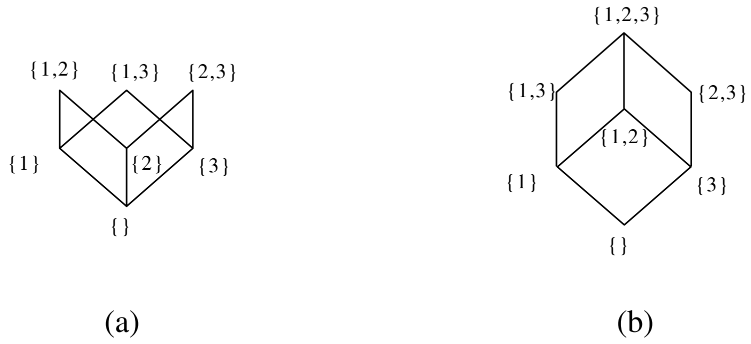


Figura 5.3 - Domínios Concretos que não são Espaços Coerentes

5.2.7 Corolário

Seja D um domínio concreto tal que satisfaz a propriedade

$$\text{(dc) } x \in D,\ x' \subseteq x \Rightarrow x' \in D.$$

Então D é um espaço coerente.

***Prova*)**

Um domínio concreto cumpre as propriedades (F), (C), (R) e (Q), e um espaço coerente deve cumprir (dc) e (bc). D cumpre a hipótese da proposição 5.2.6, portanto D cumpre (bc), isto é D é um espaço coerente. ■

5.2.8 Proposição

Todo espaço coerente E é um domínio de eventos.

***Prova*)**

Tem-se que provar que E é uma cpo algébrica, e que cumpre as propriedades (F), (C), (R) e (V).

<u>E é uma cpo</u>: Tem-se que mostrar que:

1. $\subseteq$ é uma ordem parcial completa: imediato, pois é a inclusão de conjuntos;

2. que existe um elemento "bottom": imediato, considerando $\perp = \varnothing$;

3. que dado um subconjunto $A \subseteq E$ dirigido (isto é $A \neq \varnothing$ e se $x,y \in A$, $\exists z \in A$ tal que $x \subseteq z$ e $y \subseteq z$), então $\exists$ sup(A) em E: Dado um conjunto $A \subseteq E$ dirigido, claramente sup(A) = $\bigcup A$, e pela propriedade (bc) $\bigcup A \in E$, portanto existe sup(A) em E.

De 1., 2. e 3. se tem que E é uma cpo.

<u>E é algébrico</u>: Tem-se que provar que $\forall x \in E$, o conjunto das aproximações $\mathcal{A}(x)$ de x (o conjunto dos elementos isolados menores que x) é dirigido e $x = \bigcup \mathcal{A}(x)$.

Seja $x \in E$, por definição $\mathcal{A}(x) \subseteq x$, e pela propriedade (dc) $\mathcal{A}(x) \in E$. $A \neq \varnothing$, pois $\varnothing \subseteq x$. Sejam y, $z \in \mathcal{A}(x)$, então, por definição, $y \subseteq x$ e $z \subseteq x$, o que implica que $\mathcal{A}(x)$ é dirigido.

Mostra-se agora que $x = \bigcup \mathcal{A}(x)$. Suponha que $x \neq \bigcup \mathcal{A}(x)$, portanto pode ocorrer:

1. $x \supset \bigcup \mathcal{A}(x)$, portanto $\exists e \in x$ tal que $e \notin \bigcup \mathcal{A}(x)$, isto é, e não pertence a nenhuma das aproximações de x ($\forall y \in \bigcup \mathcal{A}(x)$, $e \notin y$), mas $\{e\} \subseteq x$, e pela propriedade (dc), $\{e\} \in E$, então $\{e\} \in \mathcal{A}(x)$. Ou seja, $\exists y \in \mathcal{A}(x)$ tal que $e \in y$, que é uma contradição!

2. $x \subset \bigcup \mathcal{A}(x)$, portanto $\exists e \in \bigcup \mathcal{A}(x)$ tal que $e \notin x$, isto é $\exists y \in \mathcal{A}(x)$ tal que $e \in y$, que implica que $\neg(y \subseteq x)$, ou seja, $y \notin \mathcal{A}(x)$, que é uma contradição!

De 1. e 2., $x = \bigcup \mathcal{A}(x)$.

<u>E cumpre a prop. (F)</u>: Tem-se que mostrar que $\{d \in E \mid d \leq x\}$ é finito, com $x \in E^0$.

Sabe-se que a teia de E é uma estrutura de eventos (ver proposição 5.2.3), e que o conjunto de estados de E ($D(E)$), ordenados por inclusão, formam um domínio de eventos ($D(E)$, $\subseteq$), onde os elementos isolados (compactos) são os estados finitos (conjuntos de eventos finitos), que é um conjunto finito.

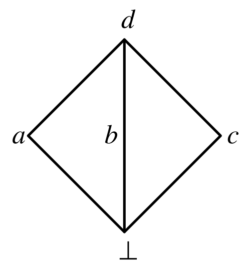
<u>E cumpre a prop. (C)</u>: Tem-se que mostrar que se $x \prec y$, $x \prec z$, $z \neq y$ e $y \uparrow z$, então existe $y \vee z$, e $y \prec y \vee z$, $z \prec y \vee z$, para todo $x, y, z \in E^0$.

Seja $x \prec y$, $x \prec z$, $z \neq y$ e $y \uparrow z$, então $\exists a_1, a_2, a_3 \in E$, tal que $y = x \cup \{a_1\}$, $z = x \cup \{a_2\}$ e $a_1 \neq a_2$. Se $y \uparrow z$, então $\exists w \in E$ tal que $y \subseteq w$ e $z \subseteq w$, portanto $y \cup z \subseteq w$ e pela propriedade (dc) se tem que $y \cup z \in E$.

Por outro lado se sabe que $y \cup z = y \cup (x \cup \{a_2\}) = y \cup \{a_2\}$, ou seja, $y \prec y \cup z$. Da mesma forma: $y \cup z = (x \cup \{a_2\}) \cup z = \{a_1\} \cup z$, ou seja, $z \prec y \cup z$.

<u>E cumpre a prop. (R)</u>: Deve-se mostrar que $[x, y] \succ\!\prec [x, z] \Rightarrow y = z$, para todo $x, y, z \in E^0$.

Considere-se a propriedade equivalente (R_F), que diz que nenhum sub-reticulado é isomorfo ao retículo



É imediato, pois E é um espaço coerente, portanto todos os "singletons" (conjuntos unitários) pertencem a E, portanto, qualquer $\perp = \emptyset$, $a = \{a\}$, $b = \{b\}$, $c = \{c\}$, $d = \{a,b,c\}$, que não é um espaço coerente pela propriedade (dc).

<u>E cumpre a prop. (V)</u>: Tem-se que mostrar que (V) $[x, x'] \succ\!\!\prec [y, y']$, $[x, x''] \succ\!\!\prec [y, y'']$ e $x' \uparrow x'' \Rightarrow y' \uparrow y''$, para todo $x, x', x'', y, y', y'' \in E^0$:

Se $[x, x'] \succ\!\!\prec [y, y']$, então $x' = x \cup \{e_1\}$ e $y' = y \cup \{e_1\}$. Igualmente, como $[x, x''] \succ\!\!\prec [y, y'']$ $x'' = x \cup \{e_2\}$ e $y'' = y \cup \{e_2\}$.

Suponha-se, por absurdo, que y' não é compatível com y'' ($y' \# y''$). Portanto, existem $e' \in y'$, $e'' \in y''$, tal que $\neg(e' \approx e'')$. Se $e' \approx e''$, $y' \cup y'' = y \cup \{e_1\} \cup \{e_2\}$ seria um objeto de E, pois E é um espaço coerente, e por definição contém todos os subconjuntos coerentes da teia. Mas, como $y' = y \cup \{e_1\}$ e $y'' = y \cup \{e_2\}$, se tem que $e' = e_1$ e $e'' = e_2$, pois y é um conjunto consistente de eventos (novamente por ser E um espaço coerente). Portanto, não poderia existir $x' \cup x'' = x \cup \{e_1\} \cup \{e_2\}$, o que implica que x' é incompatível com x'', que contradiz a hipótese. Demostrou-se então, que E cumpre a propriedade (V). ■

5.2.9 Proposição

Todo espaço coerente E é um domínio de eventos de Droste.

Prova)

Direto da proposição 5.2.2 e da proposição 5.2.8. ■

5.2.10 Proposição

Seja E um espaço coerente. E é um domínio concreto se cumpre a propriedade:

(Q) se z, x, y são elementos isolados tais que $z \prec x$, $z \prec y$ e $x \# y$, então existe uma única x' tal que $z \prec x' \leq y$ e $x \# x'$.

Prova)

Imediata. Da proposição 5.2.8, sabe-se que um espaço coerente E cumpre (F), (C) e (R), portanto só falta ter a propriedade (Q), que é satisfeita por hipótese. ■

5.2.11 Teorema

Seja E um espaço coerente, e $|E| = (E, \approx)$ a teia. Então, $(E, \#, \vdash)$ é uma estrutura de eventos, onde

$\forall e_1, e_2 \in E, e_1 \# e_2$ sss $\neg(e_1 \approx e_2)$,

e

$\forall e \in |E|, \vdash e$.

Ainda mais, o domínio de eventos canônico associado com $(E, \#, \vdash)$ é um espaço coerente.

Prova)

Imediato, pois $(E, \#, \vdash)$ assim definida é uma estrutura de eventos. Por outro lado, o domínio de eventos canônico é o conjunto de estados de $(E, \#, \vdash)$, que, pela condição $\forall e \in |E|, \vdash e$, implica que contém todos os conjuntos da forma $\{e\}$, para todo e. Facilmente se pode ver que cumpre as propriedades (dc) e (bc). ■

5.2.12 Teorema

Seja $M = (C, V, E, \vdash)$ uma cds, tal que o conjunto de estados (domínio concreto) seja um espaço coerente. Então $(E', \approx)$ é a teia do espaço coerente, onde

1. $E' = E$

2. $\forall e_1, e_2 \in E', e_1 \approx e_2$ sss $\exists c \in C, \exists v_1, v_2 \in V$, tal que $e_1 = (c, v_1), e_2 = (c, v_2)$ e $v_1 \neq v_2$. ■

5.2.13 Teorema

Seja um espaço coerente $\boldsymbol{E}$ e a teia $|\boldsymbol{E}| = (E, \approx)$. Então é possível construir uma cds M, tal que $(\boldsymbol{E}, \subseteq)$ é isomorfo a $(D(M), \subseteq)$, da seguinte forma:

1. $\forall e \in E, \vdash e$

2. o fecho reflexivo da negação de $\approx$ é transitivo

3. $\forall X, e_1, e_2, X \vdash e_1, \neg(e_1 \approx e_2) \Rightarrow X \vdash e_2$.

***Prova*)**

Dado (E, $\approx$), se pode obter (pelo teorema 5.2.11) uma estrutura de eventos (E, #, $\vdash$), e pela proposição 3.3.6, dada uma estrutura de eventos (E, #, $\vdash$), se pode obter uma cds tal que $D(M)$ e $D(E)$ sejam isomorfos, dadas as seguintes condições:

- o fecho reflexivo da negação de $\approx$ é transitivo

- $\forall X, e_1, e_2, X \vdash e_1,\ e_1 \# e_2 \Rightarrow X \vdash e_2$. ■

Foi estudada neste capítulo, a relação entre espaços coerentes, domínios concretos e domínios de eventos, assim como a relação entre as estruturas concretas destes domínios. O principal resultado para este trabalho é o fato que a teia de um espaço coerente é uma estrutura de dados concretas (cds), tal que toda célula é habilitada pelo conjunto vazio (é uma célula inicial). Os algoritmos lineares definidos no próximo capítulo, como estados de uma cds, podem ser considerados como sendo espaços coerentes (estados de uma teia de espaços coerentes).

6 Algoritmos Lineares

Neste capítulo se definem os *algoritmos lineares* sobre teias de espaços coerentes, seguindo os resultados de [CUR 86] sobre algoritmos seqüenciais e cds. Como se viu antes, dada uma função estável f: $M \to M'$ se pode obter uma função linear $lin(f)$: $!M \multimap M'$. O anterior dá uma idéia que se pode, por analogia com a definição de algoritmo seqüencial de Curien, obter os algoritmos lineares usando o operador "of course" de Girard.

6.1 Os algoritmos lineares

Nesta seção se definem os *algoritmos lineares*, o que leva a outras definições interessantes, como as de *estratégias de computação seqüencial* e *paralela*.

Claramente, dada uma cds M, o espaço coerente $!D(M)$ não é o conjunto de estados de nenhuma cds, já que os objetos de $!D(M)$ não são conjuntos de eventos, senão conjuntos de estados, e portanto não se pode determinar quais são as células. Os objetos de $!D(M)$ serão denotados por X, Y,... ou, eventualmente, por $!x$, $!y$,..., pois são obtidos a partir da aplicação do operador "of course" aos estados de M.

6.1.1 Discussão intuitiva sobre a analogia entre algoritmos seqüenciais e lineares

Ver-se-á, intuitivamente, que seria uma cds $!M \multimap M'$, seguindo a definição de Curien dos algoritmos seqüenciais.

Dada a definição 3.5.1.1, da cds $M \Rightarrow M'$, uma primeira aproximação levaria a definir a cds $!M \multimap M'$, com M e M' sendo cds, como:

1. Se x é um estado finito de M, c' uma célula de M', então as *células* de $!M \multimap M'$ são do tipo Xc', onde $X \in \wp(x)$ (ou equivalentemente, $X \subseteq\, !x$)

2. Os *valores* e *eventos* são de três tipos:
   a) tipo "valof": se $c_1,..., c_n$ são células de M, então $valof\,\{c_1,..., c_n\}$ é um valor de $!M \multimap M'$ e $(Xc', valof\,\{c_1,..., c_n\})$ é um evento de $!M \multimap M'$ se e somente se $c_1,...,$ c_n são acessíveis a partir de X (onde a acessibilidade em X é definida mais embaixo).
   b) tipo "output": se v' é um valor de M', então *output* v' é um valor de $!M \multimap M'$ e $(Xc', output\ v')$ é um evento de $!M \multimap M'$ se e somente se (c', v') é um evento de M'.
   c) tipo "merge": *merge* é um valor de $!M \multimap M'$ e $(Xc, merge)$ é um evento de $!M \multimap M'$ se e somente se não existe o supremo de X em X.

3. As relações de habilitação são de dois tipos:
   a) $(Yc', valof\,\{c_1,..., c_n\}) \vdash Xc'$ sss $Y \prec_{c1,...,cn} X$ e X é finito (tipo "valof")
   b) $(X^1c'^1, output\ v'^1),..., (X^nc'^n, output\ v'^n) \vdash Xc'$ sss $X = \cup\{X^i \mid i \leq n\}$, X é finito e $(c'^1,v'^1),..., (c'^n, v'^n) \vdash c'$ (tipo "output").

Onde a relação de "acessibilidade" poderia ser definida como: Se X é um elemento de $!D(M)$, se diz que uma célula c é *acessível* a partir de X se e somente se c é acessível a partir do sup(X); e um estado de $!M \multimap M'$ seria chamado de *algoritmo linear*.

As componentes da máquina que computa os algoritmos lineares se encontram esquematizadas na Figura 6.1, onde se tem, essencialmente, sete módulos: memória de entrada, memória de saída, memória de trabalho, memória de programa, memória do algoritmo linear, oráculo, e o controle. O algoritmo recebe dados de entrada, que pode ser considerado um estado inicial, onde algumas células (os argumentos de entrada) têm valor, assim como também um valor de referência (a célula de saída a ser preenchida). Na memória do algoritmo linear se encontra o algoritmo linear. O controle lê esta memória e seleciona a estratégia de computação a ser executada (a definição de estratégia será dado mais adiante), de acordo com a capacidade instalada da máquina (quantidade de processadores, etc.). A estratégia de computação selecionada é copiada na memória de programa, que interage com o controle. Para preencher a célula de saída, o algoritmo consulta e atualiza uma memória de trabalho, e eventualmente consulta o *oráculo* que executa as operações necessárias para preencher novas células ou obter o resultado (por exemplo na soma, no momento de executar *output* $i+j$, o oráculo seria o responsável de efetuar $i+j$). Finalmente, o algoritmo preenche a célula de referência em uma memória de saída, terminando assim o processo de execução do algoritmo.

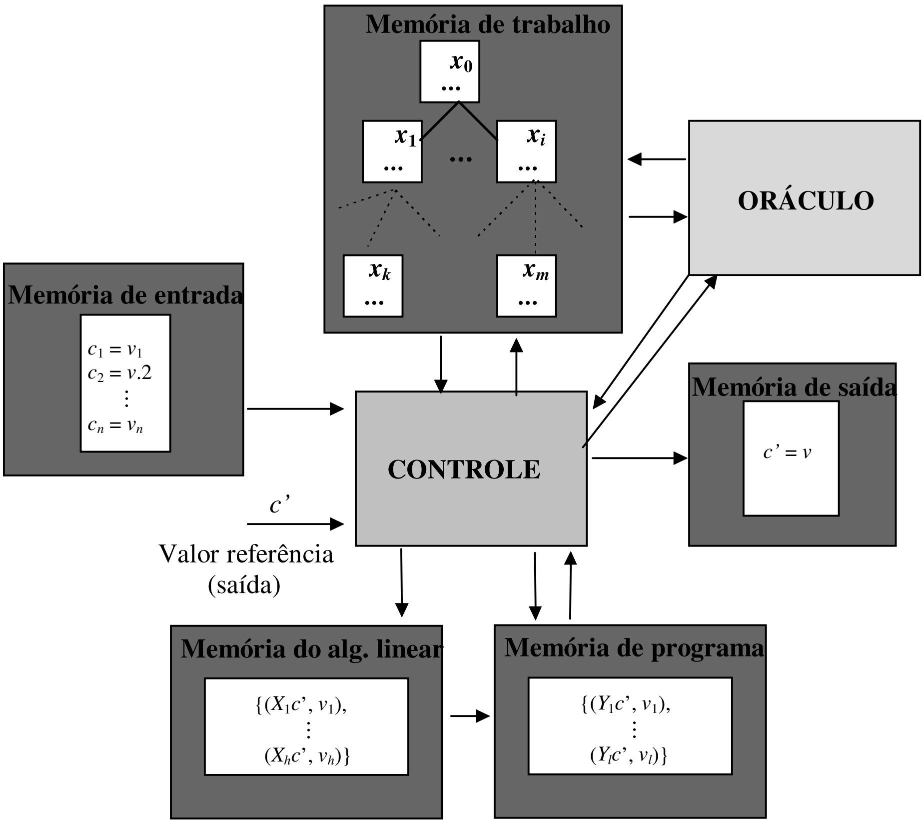


Figura 6.1 - Máquina que computa alg. lineares (1ra aproximação)

Pode-se observar que a memória de trabalho é uma árvore, e não mais uma lista como nos algoritmos seqüenciais. Logicamente, isto é assim por causa da possibilidade do paralelismo.

Considere-se novamente o exemplo do algoritmo seqüencial da soma "esquerda", dado por:

$$\mathrm{ADD}^{\aleph}_{\mathrm{ESQ}} = \{(\{\}c,\ \mathit{valof}\ c.1),\ (\{(c.1,\ i)\}c,\ \mathit{valof}\ c.2),\ (\{(c.1,\ i),\ (c.2,\ j)\}c,\ \mathit{output}\ i+j) \mid i, j \in \aleph\}.$$

Seja $\mathrm{ADD}'^{\aleph}$ definido como segue:

$\mathrm{ADD}^{!\aleph}$ = { ({ }*c*, *valof* {*c*.1}),

({ }*c*, *valof* {*c*.2}),

({ }*c*, *valof* { *c*.1, *c*.2}),

({$\varnothing$}*c*, *valof* {*c*.1}),

({$\varnothing$}*c*, *valof* {*c*.2}),

({$\varnothing$}*c*, *valof* {*c*.1, *c*.2}),

({$\varnothing$, {(*c*.1, *i*)}}*c*, *valof* {*c*.2}),

({$\varnothing$, {(*c*.2, *j*)}}*c*, *valof* {*c*.1}),

({$\varnothing$, {(*c*.1, *i*)}, {(*c*.2, *j*)}}*c*, *merge*),

({$\varnothing$, {(*c*.1, *i*)}, {(*c*.1, *i*), (*c*.2, *j*)}}*c*, *output i+j*),

({$\varnothing$, {(*c*.2, *j*)}, {(*c*.1, *i*), (*c*.2, *j*)}}*c*, *output i+j*),

({{(*c*.1, *i*)}}*c*, *valof* {*c*.2}),

({{(*c*.2, *j*)}}*c*, *valof* {*c*.1}),

({{(*c*.1, *i*)}, {(*c*.2, *j*)}}*c*, *merge*),

({{(*c*.1, *i*), (*c*.2, *j*)}}*c*, *output i+j*),

({$\varnothing$, {(*c*.1, *i*), (*c*.2, *j*)}}*c*, *output i+j*),

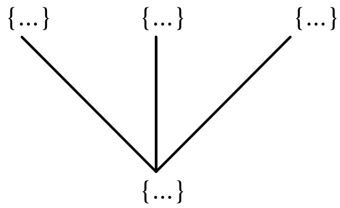
({{(*c*.1, *i*)}, {(*c*.1, *i*), (*c*.2, *j*)}}*c*, *output i+j*),

({{(*c*.2, *j*)}, {(*c*.1, *i*), (*c*.2, *j*)}}*c*, *output i+j*),

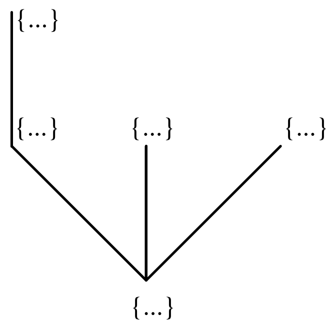
({{(*c*.1, *i*)}, {(*c*.2, *j*)}, {(*c*.1, *i*), (*c*.2, *j*)}}*c*, *output i+j*),

({$\varnothing$, {(*c*.1, *i*)}, {(*c*.2, *j*)}, {(*c*.1, *i*), (*c*.2, *j*)}}*c*, *output i+j*) }.

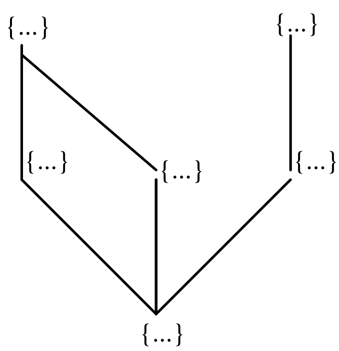
Com a definição anterior, estando no estado $\varnothing$, e executando o comando *valof* { *c*.1, *c*.2}, não fica claro se passo ao conjunto de estados {{(*c*.1, *i*), (*c*.2, *j*)}} ou {{(*c*.1, *i*)}, {(*c*.2, *j*)}}. Portanto é preciso evitar este tipo de ambigüidade. Outro aspecto importante é o fato que com a definição de *valof* anterior, também não é possível expressar a continuidade da computação por um (ou mais) "ramo" só. Por exemplo, se o estado atual é o seguinte:



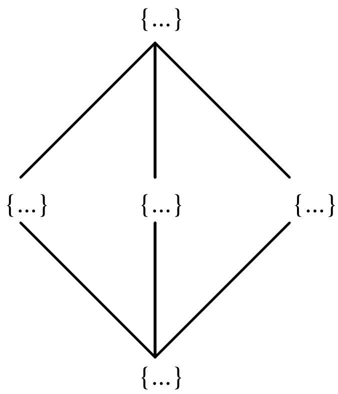
e se quiser passar ao estado:



ou ainda, ao estado



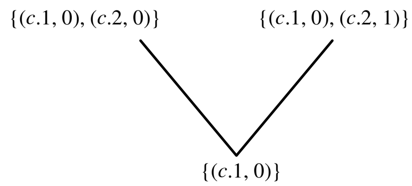
não seria possível, pois a única operação habilitada seria *merge*, a qual produziria o estado



O dito acima implica em uma nova definição dos comandos *valof* e *merge*.

A forma mais natural de fazer esta extensão, seria por como índice de cada célula do *valof*, o estado que habilita a mesma, e como sobrescrito o *processador* utilizado. Por exemplo, a computação $(\{\varnothing\}c$, *valof* $\{c.1^{m_1}_{\{\varnothing\}},\ c.2^{m_2}_{\{\varnothing\}}\}$ gera o conjunto de estados $\{\varnothing, \{(c.1, i)\}, \{(c.2, j)\}\}$; entretanto que, logo após a execução de $(\{\varnothing\}c$, *valof* $\{c.1^{m_1}_{\{\varnothing\}},\ c.2^{m_1}_{\{\varnothing\}}\}$ se teria o conj. de estados $\{\varnothing, \{(c.1, i), (c.2, j)\}\}$, pois o sobrescrito de $c.1$ e $c.2$ indicam o mesmo processador (m_1). Ou seja, dos exemplos anteriores, se pode dizer que $(Xc$, *valof*$\{c.1^{m_1}_{X_1},\ldots,c.n^{m_n}_{X_n}\}$ significa que, tendo computado os estados $x \in X$, para todo $X_i \subseteq$ maximal(X), $1 \le i \le n$, para preencher a célula c, se deve: computar $c.1$ no processador m_1, a partir do conjunto de estados X_1; computar $c.2$ no processador m_2, a partir do conjunto de estados X_2; ...; computar $c.n$ no processador m_n, a partir do conjunto de estados X_n. Isto permite ter computações paralelas simultâneas (ou não).

Com respeito ao *merge*, a definição anterior permitiria gerar estados inconsistentes. Por exemplo, seja a cds M com dois células $c.1$ e $c.2$, onde $c.1$ pode conter somente o valor 0, e $c.2$ pode conter os valores 0 e 1. Dado o estado $\{\{(c.1, 0)\}, \{(c.1, 0), (c.2, 0)\}, \{(c.1, 0), (c.2, 1)\}\}$, ou seja



a única operação possível seria *merge*, o que produziria o seguinte conjunto de estados inconsistente: $\{\{(c.1, 0)\}, \{(c.1, 0), (c.2, 0)\}, \{(c.1, 0), (c.2, 1)\}, \{(c.1, 0), (c.2, 0), (c.2, 1)\}\}$. Isto implica ter uma restrição a mais para a execução do *merge*: a de consistência dos estados argumentos do *merge*. Outra falha do *merge* é que não permite fazer a “união parcial de estados, senão somente de todos os maximais. Isto se soluciona permitindo que o merge tenha como argumentos os estados que se quer juntar. Mas, é realmente necessária a operação *merge*? Pode-se ver que dados $X_1, ..., X_n \subseteq$ maximal(X), consistentes, e se não existe o supremo de X em X, *merge* $\{X_1, ..., X_n\}$é equivalente a um caso particular de *valof* (ver definição de *merge* mais adiante).

A outra observação importante é que nos algoritmos seqüenciais de Curien, e na primeira aproximação da definição dos algoritmos lineares, se tem somente um *output*, i.e., somente uma célula de saída. Seria interessante que o algoritmo linear seja mais geral, ou seja, possa escrever em varias células de saída paralelamente. Isto permitiria dar como resultado vetores e matrizes, caso contrário se teria que computar uma célula por vez, o que seria pouco prático. A solução é, simplesmente, substituir (Xc', v') por $(X\{c_1',..., c_k'\}, v')$, e portanto permitir *output*$\{v_1,..., v_k\}$.

6.1.2 Definição formal de algoritmo linear

Agora se está em condições de definir a cds $!M \multimap M'$, mas antes se definirá a noção de *acessibilidade*:

6.1.2.1 Definição (célula acessível)

Seja M uma cds. Se $X \in !D(M)$, se diz que uma célula c é *acessível* a partir de X se e somente se $\exists x \in$ maximal(X) tal que c é acessível a partir de x na cds M. ■

6.1.2.2 Definição (relação de precedência)

Seja M uma cds. Se $X, Y \in !D(M)$, então se diz que X *precede* (*cobre*) Y *a menos* $c_1,\ldots,c_n$, $Y < _{c1,\ldots,cn} X$ ($Y \prec _{c1,\ldots,cn} X$), se e somente se para toda célula c_i, com $1 \leq i \leq n$, existe $x \in$ maximal(X) e $y \in$ maximal(Y) tal que $c_i \in A(x)$, $c_i \in F(y)$ e $X < Y$ ($X \prec Y$) na cds M. ■

6.1.2.3 Definição (a cds $!M \multimap M'$)

Se M e M' são duas cds, a cds $!M \multimap M'$ e definida como:

1. Se x é um estado finito de M, $c_1',\ldots, c_k'$ células de M', então as *células* de $!M \multimap M'$ são do tipo $X\{c_1',\ldots, c_k'\}$, onde $X \in \wp(x)$ (ou equivalentemente, $X \subseteq !x$)

2. Os *valores* e *eventos* são de dois tipos:

   a) tipo "valof": se $c_1,\ldots, c_n$ são células de M; $m_1,\ldots,m_n \in P$[14] $X_1, \ldots, X_n \subseteq$ maximal(X), são conjuntos de estados; então *valof*$\{c_{1X_1}^{m_1},\ldots,c_{nX_n}^{m_n}\}$ é um valor de $!M \multimap M'$ e $(X\{c_1',\ldots, c_k'\}$, *valof* $\{c_{1X_1}^{m_1},\ldots,c_{nX_n}^{m_n}\})$ é um evento de $!M \multimap M'$ se e somente se $c_1,\ldots,c_n$ são acessíveis a partir de $X_1,\ldots,X_n$ respectivamente.

   b) tipo "output": se $v_1',\ldots, v_k'$ são valores de M', então *output* $\{v_1',\ldots, v_k'\}$ é um valor de $!M \multimap M'$ e $(X\{c_1',\ldots, c_k'\}$, *output* $\{v_1',\ldots, v_k'\})$ é um evento de $!M \multimap M'$ se e somente se $(c_1', v_1'), \ldots, (c_k', v_k')$ são eventos de M'.

3. As relações de habilitação são de dois tipos:

   a) $(Y\{c_1',\ldots, c_k'\}$, *valof* $\{c_{1X_1}^{m_1},\ldots,c_{nX_n}^{m_n}\}) \vdash X\{c_1',\ldots, c_k'\}$ sss $Y \prec _{c1,\ldots,cn} X$ e X é finito (tipo "valof").

   b) $(X^1\{c_1'^1,\ldots, c_k'^1\}$, *output* $\{v_1'^1,\ldots, v_k'^1\}),\ldots, (X^h\{c_1'^h,\ldots, c_k'^h\}$, *output* $\{v_1'^h,\ldots, v_k'^h\}) \vdash X\{c_1',\ldots, c_k'\}$ sss $X = \cup\{X^i \mid i \leq h\}$, X é finito e $(c_1'^1, v_1'^1),\ldots, (c_k'^1, v_k'^1)$, $(c_1'^n, v_1'^n),\ldots, (c_k'^n, v_k'^n) \vdash c_i'$, ou $\varnothing \vdash c_i'$, com $1 \leq i \leq k$ (tipo "output"). ■

A definição anterior dá lugar à definição mais importante desta seção, a de *algoritmo linear*:

[14] O conjunto P é um conjunto enumerável arbitrário de processadores.

6.1.2.4 Definição (algoritmo linear)

Um estado de $!M \multimap M'$ é chamado de *algoritmo linear*. ■

Naturalmente, tendo a noção de algoritmo, é necessária a definição de *aplicação* do algoritmo:

6.1.2.5 Definição (aplicação do algoritmo linear)

Se A e um estado de $!M \multimap M'$ e X pertence a $!D(M)$, então

$$A.X = \{(c', v') \mid \exists y \in M, \{y\} \subseteq X \wedge c' \in \{c_1', \ldots, c_k'\} \wedge v' \in \{v_1', \ldots, v_k'\} \wedge (\{y\}\{c_1', \ldots, c_k'\}, output\ \{v_1', \ldots, v_k'\}) \in A\}$$

é a *aplicação* de A no estado X. A função $X \mapsto A.X$ é chamada a *função de entrada-saída* computada por A. ■

Note-se que o fato que a definição da aplicação do algoritmo A em um conjunto de estados X, é a mesma que a aplicação em um conjunto unitário $\{y\}$, por estar considerando funções lineares.

Pode-se observar as mudanças da nova definição: não se tem somente uma célula de saída a ser preenchida, senão k; assim como também se tem k eventos na memória de saída.

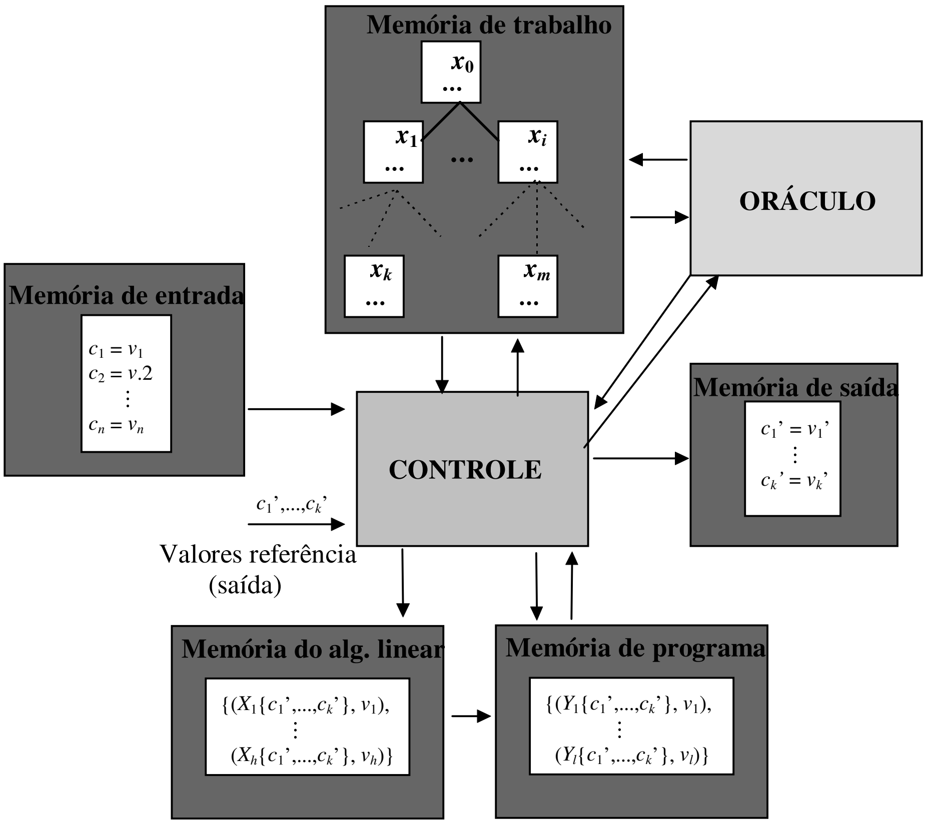


Figura 6.2 - Máquina que computa alg. lineares (2da aproximação)

As operações *fork* e *merge* de linguagens concorrentes são casos particulares do *valof* anteriormente definidos. Assim, se tem as seguintes definições:

6.1.2.6 Definição (merge)

Seja a cds $!M \multimap M'$, X um objeto de $!D(M)$ e $x_1, ..., x_n \in$ maximal(X), então *merge* $\{x_1, ..., x_n\}$= *valof*$\{c_{1Y_1}^{m_1}, \ldots, c_{lY_l}^{m_l}\}$, com $Y_1, ..., Y_l \subseteq \{x_1, ..., x_n\}$, é um valor de $!M \multimap M'$ e $(X\{c_1', ..., c_k'\}$, *merge* $\{x_1, ..., x_n\})$ é um evento de $!M \multimap M'$, se e somente se

1. não existe o supremo de X em X ($n>1$),

2. $x_1, ..., x_n$ são consistentes,

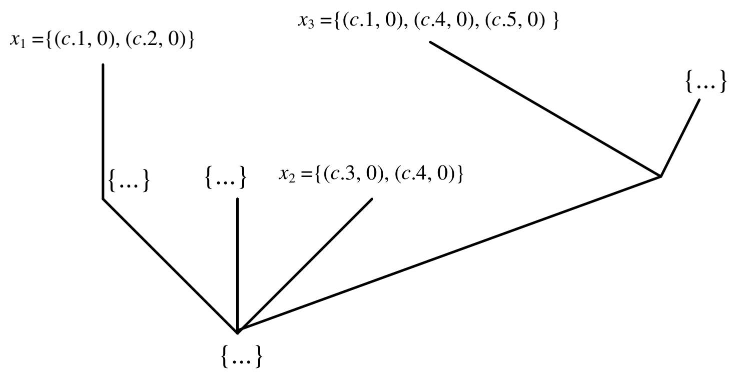
3. $m_1 = ... = m_l$,

4. $c_i \in \bigcup_{j=1}^{n} x_j - \bigcup_{y \in Y_i} y$. ■

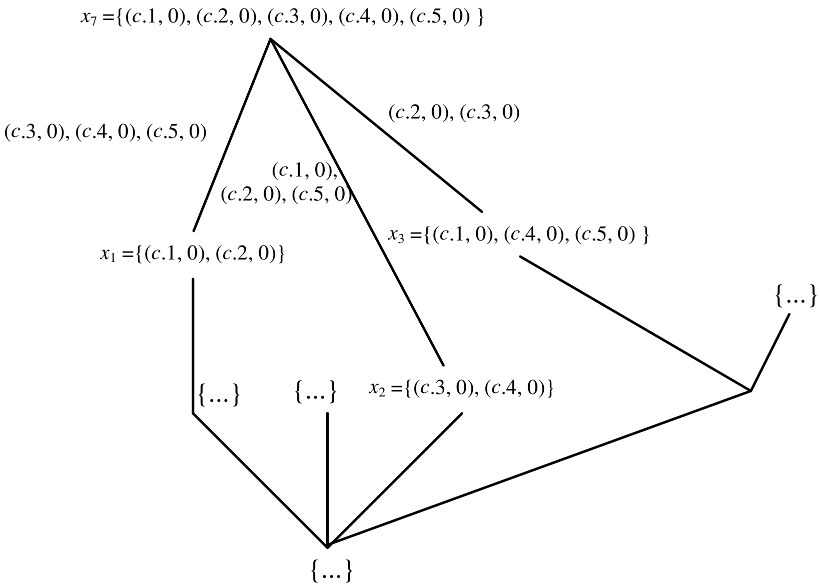
A primeira condição é necessária pois se existe o supremo de X em X a operação *merge* não faz sentido pois a cardinalidade de maximal(X) = 1, por exemplo x_1, e portanto $merge\{x_1\} = x_1$. A condição de consistência é para evitar os casos em que em dois estados distintos, maximais, tenha uma célula com dois valores diferentes, como por exemplo $merge\{\{(c_1, 0),...\}, ...\{(c_1, 1),...\}\} = \{(c_1, 0),..., (c_1, 1), ...\}$ que é um estado inconsistente. A condição 3. é obvia pela definição intuitiva do merge, pois se está "juntando" vários estados, o que implica que o processador deve ser único. A última condição é para garantir que se vão computar os valores de todas as células que não tem valor em todos os estados envolvidos no *merge*.

6.1.2.7 Exemplo

Seja uma cds com 7 células c_1, ..., c_7, que podem ter somente o valor 0. Suponha-se que o conjunto de estados atual X é dado como segue:



onde se tem 5 estados maximais. Suponha que se deseja obter $merge\{x_1, x_2, x_3\}$ (para simplificar se escreverá c_i em vez de $(c.i, 0)$). A noção intuitiva da operação merge diz que se deveria obter um novo estado $x_7 = \{c_1, ..., c_5\}$, computando c_3, c_4, c_5 a partir de x_1; c_1, c_2, c_5 a partir de x_2; e c_2, c_3 a partir de x_3:



Por definição se tem que, *merge*$\{x_1, x_2, x_3\}$= *valof*$\{c1^{m}_{Y_1}, c2^{m}_{Y_2}, c3^{m}_{Y_3}, c4^{m}_{Y_4}, c5^{m}_{Y_5}\}$, onde

$Y_1=\{x_2\} \subseteq \{x_1, x_2, x_3\}$

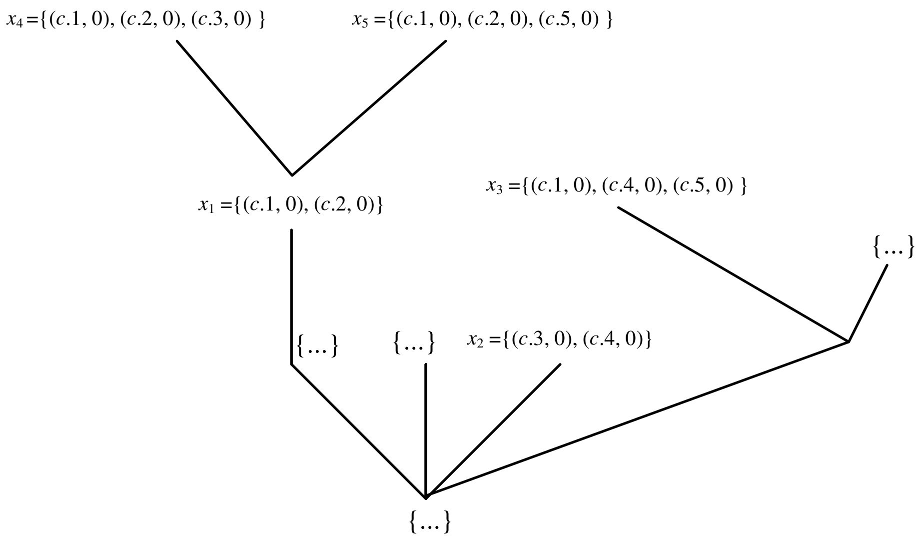
$Y_2=\{x_2, x_3\} \subseteq \{x_1, x_2, x_3\}$

$Y_3=\{x_1, x_3\} \subseteq \{x_1, x_2, x_3\}$

$Y_4=\{x_1\} \subseteq \{x_1, x_2, x_3\}$

$Y_5=\{x_1, x_2\} \subseteq \{x_1, x_2, x_3\}$.

Assim como o *merge*, é possível definir o *fork*, que é a operação que permite obter uma bifurcação a partir de um estado dado $x \in$ maximal(X). Seja X como antes, então , seja $x_1 \in X$ o estado que se deseja bifurcar, computando c_3 e c_5 por exemplo. Obteria-se o seguinte conjunto de estados:



Formalmente:

6.1.2.8 Definição (fork)

Sejam $!M \multimap M'$ uma cds, X um objeto de $!D(M)$ e $y \in$ maximal(X), então $fork_y\ \{c_1^{m_1},\ldots,c_n^{m_n}\} = valof\{c_{1_{\{y\}}}^{m_1},\ldots,c_{l_{\{y\}}}^{m_l}\}$ é um valor de $!M \multimap M'$ e ($X\{c_1',...c_k'\}$, $fork_y\ \{c_1^{m_1},\ldots,c_n^{m_n}\}$) é um evento de $!M \multimap M'$ se e somente se $c_1,..., c_n \notin F(y)$. ■

A condição de que as células a serem computadas não tenham valor em y ($c_1,..., c_n \notin F(y)$), se deve ao fato de que, se a célula já tem valor em y, ela deve permanecer com o mesmo em todo estado que inclua y.

6.2 Como obter o algoritmo linear de uma função estável

Nesta seção se dará uma forma construtiva de obter um estado da cds *!M*—∘*M'* (um algoritmo linear), a partir de uma função estável *f*: *M*→*M'*.[15]

É necessário definir antes a *operação compatível* com um conjunto de estados *X*.

6.2.1 Definição (operação compatível)

Uma operação (valor) *v* de uma cds *!M*—∘*M'*, é *compatível* com um conjunto de estados *X* de *!D*(*M*), se e somente se

1. $v = valof\{c_{1\,X_1}^{m_1},\ldots,c_{n\,X_n}^{m_n}\}$, com m_1, ..., $m_n \in P$, e a execução de *v* no conjunto de maximais de *X* gera um conjunto de estados *X'* que pertence a *!D*(*M*);

2. *v* = *output* {v_1',..., v_k'} e existe $x \in$ maximal(*X*) tal que a função subjacente *f*(*x*) seja definida. ■

A definição anterior pode ser expressa como: se (*X*{c_1',..., c_k'}, *v*) é um evento de *!M*—∘*M'*, que produz *X'*, então (*X'*{c_1',..., c_k'}, *v'*) é um evento de *!M*—∘*M'*, para alguma operação *v'* (ou, equivalentemente, (*X*{c_1',..., c_k'}, *v*) $\vdash$ (*X'*{c_1',..., c_k'}, *v'*))

6.2.2 Construção da cds *!M*—∘*M'* a partir de uma função *f*

Sejam *M* e *M'* duas cds (teias de espaços coerentes) e *f*: *M*→*M'* uma função estável. A cds *!M*—∘*M'* se pode obter da seguinte forma:

1. Obtenha *!D*(*M*) aplicando a função *oc*(*D*(*M*)) (“of course” de *D*(*M*))

[15] Se *f* não é estável, o algoritmo de construção da cds *!M*—∘*M'* ainda é válido. Isto é, se *f*: *M* → *M'* não é estável, é possível aplicar o operador “of course” a *M*. Obteria-se uma função *f'*: *!M* → *M'*, mas que não seria necessariamente linear.

2. A partir do objeto {} de $!D(M)$ escreva todos os eventos $(\{\}\{c_1\text{'},\ldots, c_k\text{'}\}, \mathit{valof}\ C)$, onde C é o conjunto de células a serem preenchidas. Para o átomo $\{\varnothing\}$, escreva $(\{\}\{c_1\text{'},\ldots, c_k\text{'}\}, \mathit{start})$, onde *start* é a operação virtual que habilita o sistema.

3. Para todas as operações dos eventos do passo 2, gere todos os *eventos variáveis* da forma $(A\{c_1\text{'},\ldots, c_k\text{'}\}, \mathit{valof}\ \{ {c_1}^{m_1}_{B_1},\ldots,{c_n}^{m_n}_{B_n} \})$ (onde A e $B_1,\ldots,B_n$ são variáveis).[16]

4. Para todo $X \in !D(M)$, instancie os eventos gerados no passo 3., instanciando A com X, e $B_1,\ldots,B_n$ com subconjuntos $X_1,\ldots,X_n \subseteq \mathit{maximal}(X)$, se e somente se a operação é compatível com X.

5. Para todo X tal que existe $\sup(X)$ em X e $f(\sup(X)) \neq \varnothing$ (isto é, tem um valor definido), escreva o evento $(X\{c_1\text{'},\ldots, c_k\text{'}\}, \mathit{output}\ \{v_1\text{'},\ldots, v_k\text{'}\})$, onde $v_1\text{'},\ldots, v_k\text{'} \in M\text{'}$. ■

Asperti introduz , em [ASP 90], uma noção de computabilidade de funções estáveis entre espaços coerentes, baseado na enumerabilidade recursiva do trace da função. Sem entrar em detalhes, é claro que quando se fala de obter um algoritmo a partir de uma função estável, é necessário ter em conta que a função deve ser computável. Se não fosse assim, se estaria dando uma forma de computar uma função não computável, o que é um absurdo.

6.2.3 Exemplo

Ver-se-á a execução do algoritmo anterior para alguns casos da soma de dois argumentos $\mathrm{ADD}^{!\aleph}$:

Passo 1. Ver exemplo 4.7.5

Passo 2. A partir de {} se podem obter os seguintes eventos:

- $(\{\}\{c\text{'}\}, \mathit{start}) \rightarrow \{\varnothing\}$
- $(\{\}\{c\text{'}\}, \mathit{valof}\ \{ {c_1}^{m_1}_{\{\}} \}) \rightarrow \{\{1\}\}$

---

[16]O evento com valor *start* não se inclui, pois somente é possível esta operação no conjunto vazio ({}).

- $(\{\}\{c'\}, \mathit{valof}\,\{c_{2}{}^{m_2}_{\{\}}\}) \rightarrow \{\{2\}\}$

- $(\{\}\{c'\}, \mathit{valof}\,\{c_{1}{}^{m_1}_{\{\}}, c_{2}{}^{m_1}_{\{\}}\}) \rightarrow \{\{1,2\}\}$

- $(\{\}\{c'\}, \mathit{valof}\,\{c_{1}{}^{m_1}_{\{\}}, c_{2}{}^{m_2}_{\{\}}\}) \rightarrow \{\{1\},\{2\}\}$

Passo 3. Os eventos *variáveis* são:

- $(A\{c'\}, \mathit{valof}\,\{c_{1}{}^{m_1}_{B_1}\})$

- $(A\{c'\}, \mathit{valof}\,\{c_{2}{}^{m_2}_{B_2}\})$

- $(A\{c'\}, \mathit{valof}\,\{c_{1}{}^{m_1}_{B_1}, c_{2}{}^{m_2}_{B_2}\})$

- $(A\{c'\}, \mathit{valof}\,\{c_{1}{}^{m_1}_{B_1}, c_{2}{}^{m_1}_{B_2}\})$

Passo 4. Após a execução dos eventos do passo 2. se obtém os quatro átomos de *!M*. Os eventos criados a partir de $\{\varnothing\}$, e de $\{\{1\}\}$ são:

- $(A\{c'\}, \mathit{valof}\,\{c_{1}{}^{m_1}_{B_1}\})$, com $A = B_1 = \{\varnothing\}$, passa de $\{\varnothing\}$ a $\{\varnothing,\{1\}\}$

- $(A\{c'\}, \mathit{valof}\,\{c_{2}{}^{m_2}_{B_2}\})$, com $A = B_2 = \{\varnothing\}$, passa de $\{\varnothing\}$ a $\{\varnothing,\{2\}\}$

- $(A\{c'\}, \mathit{valof}\,\{c_{1}{}^{m_1}_{B_1}, c_{2}{}^{m_2}_{B_2}\})$, com $A = B_1 = B_2 = \{\varnothing\}$, passa de $\{\varnothing\}$ a $\{\varnothing,\{1\}\{2\}\}$

- $(A\{c'\}, \mathit{valof}\,\{c_{1}{}^{m_1}_{B_1}, c_{2}{}^{m_1}_{B_2}\})$, com $A = B_1 = B_2 = \{\varnothing\}$, passa de $\{\varnothing\}$ a $\{\varnothing,\{1,2\}\}$

- $(A\{c'\}, \mathit{valof}\,\{c_{1}{}^{m_1}_{B_1}\})$, $(A\{c'\}, \mathit{valof}\,\{c_{1}{}^{m_1}_{B_1}, c_{2}{}^{m_1}_{B_2}\})$ e $(A\{c'\}, \mathit{valof}\,\{c_{1}{}^{m_1}_{B_1}, c_{2}{}^{m_2}_{B_2}\})$, com $A = B_1 = B_2 = \{\{1\}\}$ não são eventos de *!M*$\multimap$*M'*, pois $\mathit{valof}\,\{c_{1}{}^{m_1}_{B_1}\}$, $\mathit{valof}\,\{c_{1}{}^{m_1}_{B_1}, c_{2}{}^{m_1}_{B_2}\})$ e $\mathit{valof}\,\{c_{1}{}^{m_1}_{B_1}, c_{2}{}^{m_2}_{B_2}\}$ não são operações compatíveis com o estado $\{1\}$ (a célula c_1 já foi computada)

- $(A\{c'\}, \mathit{valof}\,\{c_{2}{}^{m_2}_{B_2}\})$, com $A = B_2 = \{\{1\}\}$, passa de $\{\{1\}\}$ a $\{\{1\},\{1,2\}\}$

- ...

Passo 5. O único objeto tal que a função está definida é $\{1,2\}$, portanto os eventos com valor *output v'* são:

- $(A\{c'\}, \mathit{output}\ i+j)$, com $A = \{\{1,2,\}\}$

- ($A\{c'\}$, *output* $i+j$), com $A = \{\emptyset,\{1,2,\}\}$
- ($A\{c'\}$, *output* $i+j$), com $A = \{\{1\},\{1,2,\}\}$
- ($A\{c'\}$, *output* $i+j$), com $A = \{\{2\},\{1,2,\}\}$
- ($A\{c'\}$, *output* $i+j$), com $A = \{\{1\},\{2\},\{1,2,\}\}$
- ($A\{c'\}$, *output* $i+j$), com $A = \{\emptyset,\{1\},\{2\},\{1,2,\}\}$
- ($A\{c'\}$, *output* $i+j$), com $A = \{\emptyset,\{1\},\{1,2,\}\}$
- ($A\{c'\}$, *output* $i+j$), com $A = \{\emptyset,\{2\},\{1,2,\}\}$. ■

É importante ressaltar o fato que, no espaço coerente $!D(M)$, a relação de ordem entre os objetos é a inclusão, que não implica necessariamente uma relação computacional. Assim, há alguns casos que não têm sentido (imediatamente evidente), como por exemplo:

- passar de $\{\{1\},\{2\}\}$ a $\{\emptyset,\{1\},\{2\}\}$,
- passar de $\{\{1,2\}\}$ a $\{\{1\},\{1,2\}\}$,
- passar de $\{\{1,2\}\}$ a $\{\emptyset,\{1,2\}\}$,
- ...;

pois as operações somente estão definidas sobre estados maximais dos objetos de $!D(M)$.

6.3 Trace do algoritmo linear

As funções contínuas, estáveis e lineares são caracterizadas pelo *trace*. Define-se nesta seção o trace de um algoritmo linear.

6.3.1 Definição (trace do algoritmo linear)

Seja $!M \multimap M'$ uma cds e $A \in !M \multimap M'$ um algoritmo linear. Então, se define o *trace do algoritmo linear*, tr_{alg}, como

$$tr_{alg}(A) = \{(X, (c_i', v_i')) \mid 1 \leq i \leq k, (X\{c_1',..., c_k'\}, output\ \{v_1',..., v_k'\}) \in A\}. \blacksquare$$

Pode-se observar que, assim definido, o trace de um algoritmo linear computa não somente funções, como também relações, pois para cada evento $(X\{c_1',..., c_k'\}, output\ \{v_1',..., v_k'\}) \in A$, se tem k elementos do trace, o que implica que não cumpre a unicidade da definição de função. Seria possível considerar o trace como o conjunto de todos os pares da forma $(X, \{(c_1', v_1'),..., (c_k', v_k')\})$, e nessa forma seria uma função. Com a definição anterior, restringindo a saída a um valor, o trace computaria então uma função. Com essa restrição, tem-se a seguinte proposição:

6.3.2 Proposição

Seja $!M \multimap M'$ uma cds e $A \in !M \multimap M'$ um algoritmo linear que computa uma função estável $f: M \to M'$. Então, o trace do algoritmo linear $tr_{alg}(A)$ é um trace linear se e somente se para todo evento $(X\{c_1',..., c_k'\}, output\ \{v_1',..., v_k'\}) \in A$, $k = 1$.

***Prova*)**

Claramente, se todos os eventos de A que têm valor *output* V' têm a forma $(X\{c_1'\}, output\ \{v_1'\})$, o trace obtido pela definição anterior é de uma função (cumpre as propriedades de existência e unicidade). Em particular, o trace dessa função é linear, pois por definição, se $x \in D(M)$, $f(x) = \{(c', v')\}$ e $oc(x) = X$, então existe um evento $(Y\{c'\}, output\ \{v'\})$ em A tal que $Y = \{y\} \subseteq X$ (se M é finito, então $y = x$). Portanto, $tr_{alg} = \{(\{y\}, (c', v')) \mid \{y\} \subseteq X \text{ e } (X\{c'\}, output\ \{v'\}) \in A\}$. Por definição de trace linear, tr_{alg} é linear. $\blacksquare$

A proposição anterior permite definir, dada uma função estável f, as relações entre esta, a função linearizada, o *algoritmo* linear, e os seus respectivos traces. A Figura 6.3 (pág. 125) mostra esta relação.

6.4 Estratégias de computação

Definem-se nesta seção as noções de *computação* e de *estratégia de computação*.

Voltando ao exemplo do algoritmo que executa a soma de dois argumentos $\mathrm{ADD}^{/\aleph}$, tem-se:

$\mathrm{ADD}^{/\aleph}$ = { $(\{\}c, \mathit{valof}\,\{c.1^{m_1}_{\{\}}\})$,

$(\{\}c, \mathit{valof}\,\{c.2^{m_1}_{\{\}}\})$,

$(\{\}c, \mathit{valof}\,\{c.1^{m_1}_{\{\}}, c.2^{m_2}_{\{\}}\}$,

$(\{\}c, \mathit{valof}\,\{c.1^{m_1}_{\{\}}, c.2^{m_1}_{\{\}}\})$,

$\vdots$

$(\{\varnothing, \{(c.1, i)\}, \{(c.2, j)\}\}c, \mathit{merge}\,\{\{(c.1, i)\}, \{(c.2, j)\}\})$,

$(\{\varnothing, \{(c.1, i)\}, \{(c.1, i), (c.2, j)\}\}c, \mathit{output}\ i{+}j)$,

$(\{\varnothing, \{(c.2, j)\}, \{(c.1, i), (c.2, j)\}\}c, \mathit{output}\ i{+}j)$,

$(\{\{(c.1, i)\}\}c, \mathit{valof}\,\{c.2^{m_1}_{\{\{(c.1,i)\}\}}\})$,

$\vdots$

$(\{\{(c.1, i)\}, \{(c.2, j)\}, \{(c.1, i), (c.2, j)\}\}c, \mathit{output}\ i{+}j)$,

$(\{\varnothing, \{(c.1, i)\}, \{(c.2, j)\}, \{(c.1, i), (c.2, j)\}\}c, \mathit{output}\ i{+}j)$ }

Em $\mathrm{ADD}^{/\aleph}$ se podem observar alguns fatos interessantes. Alguns eventos têm valor *valof*, *output* e *merge*. Os primeiros são aqueles que dão resultados parciais, assim $(\{\varnothing\}c,\ \mathit{valof}\ \{c.1^{m_1}_{\{\varnothing\}}\})$ significa que, estando no estado $\varnothing$, para preencher a célula c se deve computar primeiro o valor de $c.1$, entretanto que $(\{\}c,\ \mathit{valof}\ \{c.1^{m_1}_{\{\}}, c.2^{m_1}_{\{\}}\})$, por exemplo, diz que se o estado atual é o indefinido, para preencher a célula c, se deve computar paralelamente as células $c.1$ e $c.2$, com o mesmo processador. Os eventos que têm como valor *output v'* são aqueles (Xc, v), tais que $\sup(X) = \{(c.1, i), (c.2, j)\}$, ou seja o supremo da cds $\aleph$, i.e., que dão como resultado o estado final (um estado maximal do domínio "original" do algoritmo estável subjacente). Finalmente se tem os eventos com valor *merge*, que, por definição, são aqueles (Xc, v), tais que não existe o supremo de X, e portanto, depois da execução do *merge*, vão produzir um conjunto de estados Y tal que exista o supremo de um subconjunto de X. Um exemplo deste ultimo tipo é $(\{\{(c.1, i)\}, \{(c.2, j)\}\}c$, *merge* $\{\{(c.1, i)\}, \{(c.2, j)\}\})$, e que, depois da execução do *merge*, o evento atual é $(\{\{(c.1, i)\}, \{(c.2, j)\}, \{(c.1, i), (c.2, j)\}\}c$, *output* $i+j)$.

No algoritmo anterior, se pode também observar que, se tem, em princípio, dois tipos de "células" Xc: aquelas em que X contém o estado $\varnothing$ e $\{(c.1, i), (c.2, j)\}$, e as que não contem um, ou nenhum deles. Esta "classificação", aparentemente sem importância, reflete uma das características mais importantes dos algoritmos lineares: a noção de *estratégia de computação*. Assim, todo X que contem o estado $\varnothing$ e $\{(c.1, i), (c.2, j)\}$, dá uma forma de chegar ao estado final $\{(c.1, i), (c.2, j)\}$, partindo do estado inicial $\varnothing$, e obter assim o resultado final *output* $i+j$. Os outros eventos são simplesmente *computações parciais*, ou "partes" de estratégias. As observações anteriores dão lugar às seguintes definições.

6.4.1 Definição (computação)

Seja $!M \multimap M'$ uma cds. Um evento da cds $!M \multimap M'$ é chamado uma *computação*, e portanto, o domínio de um algoritmo linear e denominado *domínio de computações*[17]. ■

No contexto dos algoritmos lineares se usará indistintamente os termos *evento* e *computação*.

Notação: Eventualmente se denotará com C' o conjunto de células de saída $\{c_1', \ldots, c_k'\}$, (ou com C quando não der lugar a confusão com o conjunto de células de uma cds).

6.4.2 Definição (relação de precedência computacional)

Seja $!M \multimap M'$ uma cds. Seja A um algoritmo linear de $!M \multimap M'$ e (X_1C_1', v_1) e $(X'C', v')$ duas computações de A. Então se diz que (X_1C_1', v_1) *precede*

[17]Não confundir com a noção de *domínio de computação* (ou simplesmente, domínio) da teoria dos domínios. O nome utilizado neste trabalho, se deve a que os objetos do domínio do algoritmo linear refletem os passos de computação (ou *computações*) executados, até o estado atual.

computacionalmente $(X'C', v')$, $(X_1C_1', v_1) \leq_{comp} (X'C', v')$, se e somente se existe uma seqüência de eventos de A, (X_1C_1', v_1), ..., $(X_nC_n', v_n) = (X'C', v')$ tal que $(X_iC_i', v_i) \vdash X_{i+1}C_{i+1}'$, $\forall i$, $1 \leq i \leq n$. Se $n = 1$, então se diz que (X_1C_1', v_1) *cobre computacionalmente* $(X'C', v')$ e é denotado por $(X_1C_1', v_1) \prec_{comp} (X'C', v')$. ■

Claramente, para definir a ordem computacional, não se pode ter em conta a ordem em $!D(M)$ (a inclusão), pois por exemplo $\{\emptyset\}$ precede a $\{\emptyset, \{1\}\}$, que precede $\{\emptyset, \{1\}, \{2\}\}$ em $!D(M)$, mas computacionalmente, não existe nenhuma operação v' tal que $(\{\emptyset, \{1\}\}C', v') \vdash \{\emptyset, \{1\}, \{2\}\}C''$.

Seria interessante achar uma semântica operacional que permita dar significado a segmentos de programas (por exemplo funções e procedimentos dentro de um programa principal), e não somente ao programa inteiro. Para isto, se deveria ter uma noção de estratégia não muito forte, como a seguinte:

6.4.3 Definição (estratégia de computação)

Seja $!M \multimap M' = (C, V, E, \vdash)$ uma cds. Seja $A \subseteq E$ um algoritmo linear. Então, um subconjunto $a \subseteq A$ se diz uma *estratégia de computação* (ou simplesmente *estratégia*) se e somente se, fixado um conjunto de estados X_0 e uma operação v_0,

1. $\exists\, !(X_0C_0', v_0) \in a$ tal que $\forall (X'C', v') \in a$, $(X_0C_0', v_0) \leq_{comp} (X'C', v')$;

2. se $(X'C', v') \in a$ e $(X'C', v'') \in a$, então $v' = v''$;

3. $\forall (X'C', v') \in a$, se $(X''C'', v'') \leq_{comp} (X'C', v')$ em A, então $(X''C'', v'') \in a$. ■

A primeira condição estabelece que a tem um primer elemento (com respeito a $\leq_{comp}$). A condição 2. garante a consistência, e a última condição faz o fecho inferior em a. A definição anterior permite dar semântica de segmentos de programas, o qual possibilita obter a semântica do programa principal como composição das semânticas dos segmentos.

6.4.4 Definição (estratégias totais e parciais)

Seja $!M \multimap M' = (C, V, E, \vdash)$ uma cds. Seja $A \subseteq E$ um algoritmo linear, e a uma estratégia de computação de A. Diz-se que a é uma *estratégia de computação total* se e somente se

1. $(\{\varnothing\}C_0', v_0) \in a$ e $\forall(X'C', v') \in a$, $(\{\varnothing\}C_0', v_0) \leq_{comp} (X'C', v')$;

2. se $(X'C', v') \in a$, então $\forall(XC'', v) \in A$ tal que $(X'C', v') \vdash XC''$, $(XC'', v) \in a$.

Se não se cumpre alguma das condições anteriores, então a estratégia se diz que é *parcial*. ■

Dentro das estratégias de computação, claramente se tem algumas que, em cada computação preenchem unicamente uma célula, ou escrevem somente um valor de saída, entretanto outras aplicam estas operações a conjuntos de células e valores, respectivamente. Isto leva à definição de estratégia de computação seqüencial e paralela:

6.4.5 Definição (estratégias paralelas e seqüenciais)

Seja $!M \multimap M' = (C, V, E, \vdash)$ uma cds. Seja $A \subseteq E$ um algoritmo linear, e a uma estratégia de computação de A. Diz-se que a é uma *estratégia de computação seqüencial*, se e somente se

1. $\forall(X\{c_1',..., c_k'\}, \mathit{output}\ \{v_1',..., v_k'\}) \in a$ e $k = 1$; e

2. $\forall(X\{c_1',..., c_k'\}, \mathit{valof}\ \{c_{1X_1}^{m_1}, \ldots, c_{nX_n}^{m_n}\}) \in a$ e $n = 1$.

Caso contrário a estratégia é dita *paralela*. ■

Intuitivamente, as estratégias de computação são os programas que computam a função subjacente. Claramente, para cada função estável (computável), existem varias estratégias de computação, como se pode ver na Figura 6.3.

6.4.6 Exemplo

No exemplo da soma de dois argumentos, as *estratégias de computação* são:

1. $\{(\{\varnothing\}\{c'\},\ \textit{valof}\ \{c_{1\,B_1}^{\ m_1}\}),\ (\{\varnothing,\{1\}\}\{c'\},\ \textit{valof}\ \{c_{2\,B_2}^{\ m_2}\}),\ (\{\varnothing,\{1\},\{1,2\}\}\{c'\},\ \textit{output}\ i{+}j)\}$ (com $B_1 = \{\varnothing\}$, $B_2 = \{\{1\}\}$)

2. $\{(\{\varnothing\}\{c'\},\ \textit{valof}\ \{c_{2\,B_2}^{\ m_2}\}),\ (\{\varnothing,\{2\}\}\{c'\},\ \textit{valof}\ \{c_{1\,B_1}^{\ m_1}\}),\ (\{\varnothing,\{2\},\{1,2\}\}\{c'\},\ \textit{output}\ i{+}j)\}$ (com $B_2 = \{\varnothing\}$, $B_1 = \{\{2\}\}$)

3. $\{(\{\varnothing\}\{c'\},\ \textit{valof}\ \{c_{1\,B_1}^{\ m_1}, c_{2\,B_2}^{\ m_2}\}),\ (\{\varnothing,\{1\},\{2\}\},\ \textit{merge}\ \{\{1\},\{2\}\}),\ (\{\varnothing,\{1\},\{2\},\{1,2\}\}\{c'\},\ \textit{output}\ i{+}j)\}$ (com $B_1 = B_2 = \{\varnothing\}$)

4. $\{(\{\varnothing\}\{c'\},\ \textit{valof}\ \{c_{1\,B_1}^{\ m_1},\ c_{2\,B_2}^{\ m_1}\}),\ (\{\varnothing,\{1,2\}\}\{c'\},\ \textit{output}\ i{+}j)\}$ (com $B_1 = B_2 = \{\varnothing\}$)

Onde 1. e 2. são estratégias seqüenciais, enquanto que 3. e 4. são paralelas. ■

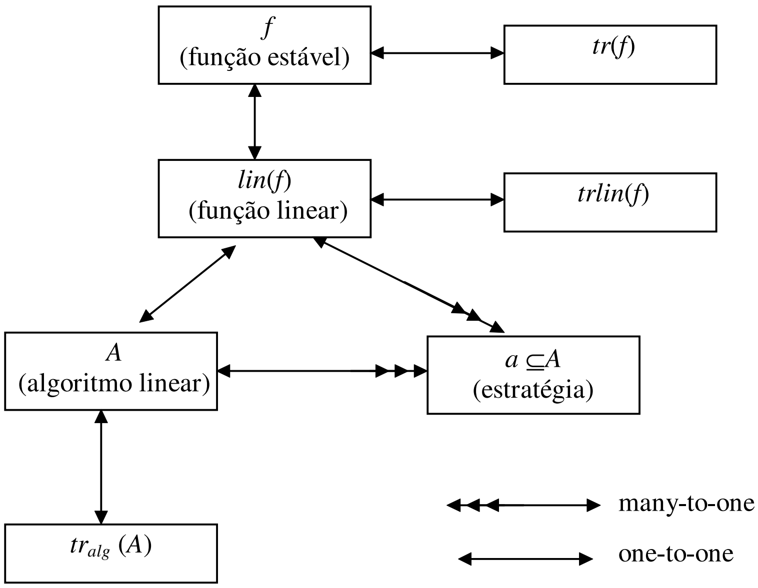


Figura 6.3 - Relação das funções estáveis, lineares, alg. lineares e estratégias

O seguinte é um exemplo de um algoritmo linear obtido a partir de uma função não estável.

6.4.7 Exemplo

Considere-se o domínio S dos strings de comprimento variável sobre o alfabeto $A = \{a, b, \ldots, z\}$. Claramente, para obter um espaço coerente, se tem que considerar conjuntos, e portanto não é possível diferenciar entre o caracter a, por exemplo, quando estiver na primeira posição de um string e na terceira em outro: *adc* e *cda* teriam a mesma representação $\{a, c, d\}$. Existe também o problema de ter mais de uma ocorrência de um caracter em um string: *abcad*, cuja representação como conjunto seria $\{a, b, c, d\}$. Para solucionar este problema, se indexarão os caracteres com a posição que eles ocupam no string. Assim, o string *cda* será representado como $\{c_1, d_2, a_3\}$, enquanto que *adc* será representado como $\{a_1, d_2, c_3\}$ que claramente são conjuntos diferentes; e $\{a_1, b_2, c_3, a_4, d_5\}$ representará *abcad.*

Seja $\boldsymbol{S}$ o espaço coerente, cuja teia é $|\boldsymbol{S}| = \{a_1, a_2, ..., b_1, b_2, ..., ..., z_1, z_2,...\}$, onde a relação de coerência é definida como:

$$\forall x_i, y_i \in |S|, x_i \approx y_i \text{ sss } i \neq j.$$

Para simplificar, serão considerados somente strings de, no máximo, comprimento três, sobre o alfabeto $S = \{a, b, c\}$. Seja $sort{:}\boldsymbol{S} \to \boldsymbol{S}$ a função que recebe um string como entrada, e dá como resultado o string ordenado segundo a ordem lexicográfica. Claramente esta função não é estável nem linear: considere os objetos $\{b_1, a_2\}$ e $\{b_1, c_3\}$ de $\boldsymbol{S}$, então:

- estabilidade: $\{b_1, a_2\}\cup\{b_1, c_3\}= \{b_1, a_2, c_3\} \in \boldsymbol{S}$, mas $sort(\{b_1, a_2\}\cap\{b_1, c_3\}) = sort(\{b_1\}) = \{b_1\}$. Entretanto que $sort(\{b_1, a_2\}) \cap sort(\{b_1, c_3\}) = \{a_1, b_2\}\cap\{b_1, c_3\} = \varnothing$. Portanto não é estável.

- linearidade: não é linear, pois por definição toda função linear deve ser estável.

Obtém-se, seguindo a definição, uma estratégia de computação total. Os sub-índices serão substituídos por células. Assim a_2 será representado como $\{(c.2, a)\}$.

1. O evento $(\{\varnothing\}\{c_1', c_2', c_3'\}, v_0)$ pertence à estratégia por definição. Seja $v_0 = valof$ $\{c.2\}$[18]. Tem-se, portanto, as seguintes computações:

$$(\{\varnothing,\{(c.2, a)\}\}\{c_1', c_2', c_3'\}, v')$$

$$(\{\varnothing,\{(c.2, b)\}\}\{c_1', c_2', c_3'\}, v'')$$

[18]Neste exemplo se omitirão os sub-índices e super-índices das células do valof pois se computará sempre a partir do supremo de X, e sempre usando o mesmo processador.

$(\{\emptyset,\{(c.2, c)\}\}\{c_1', c_2', c_3'\}, v''')$.

Fixando $v' = v'' = v''' = valof\,\{c.1, c.3\}$, tem-se os seguintes eventos:

$(\{\emptyset,\{(c.2, a)\},\{(c.2, a),(c.1, a),(c.3, a)\}\}\{c_1', c_2', c_3'\}, v_1')$

$(\{\emptyset,\{(c.2, a)\},\{(c.2, a),(c.1, a),(c.3, b)\}\}\{c_1', c_2', c_3'\}, v_2')$

$\vdots$

$(\{\emptyset,\{(c.2, a)\},\{(c.2, a),(c.1, c),(c.3, c)\}\}\{c_1', c_2', c_3'\}, v_9')$

$(\{\emptyset,\{(c.2, b)\},\{(c.2, b),(c.1, a),(c.3, a)\}\}\{c_1', c_2', c_3'\}, v_1'')$

$(\{\emptyset,\{(c.2, b)\},\{(c.2, b),(c.1, a),(c.3, b)\}\}\{c_1', c_2', c_3'\}, v_2'')$

$\vdots$

$(\{\emptyset,\{(c.2, b)\},\{(c.2, b),(c.1, c),(c.3, c)\}\}\{c_1', c_2', c_3'\}, v_9'')$

$(\{\emptyset,\{(c.2, c)\},\{(c.2, c),(c.1, a),(c.3, a)\}\}\{c_1', c_2', c_3'\}, v_1''')$

$(\{\emptyset,\{(c.2, c)\},\{(c.2, c),(c.1, a),(c.3, b)\}\}\{c_1', c_2', c_3'\}, v_2''')$

$\vdots$

$(\{\emptyset,\{(c.2, c)\},\{(c.2, c),(c.1, c),(c.3, c)\}\}\{c_1', c_2', c_3'\}, v_9''')$.

Fixando $v_1' = ... = v_9' = v_1'' = ... = v_9'' = v_1''' = ... = v_9''' = output\ \{v_1', v_2'', v_3'''\}$, tem-se os seguintes eventos:

$(\{\emptyset,\{(c.2, a)\},\{(c.2, a),(c.1, a),(c.3, a)\}\}\{c_1', c_2', c_3'\}, output\ \{a, a, a\})$

$(\{\emptyset,\{(c.2, a)\},\{(c.2, a),(c.1, a),(c.3, b)\}\}\{c_1', c_2', c_3'\}, output\ \{a, a, b\})$

$\vdots$

$(\{\emptyset,\{(c.2, a)\},\{(c.2, a),(c.1, c),(c.3, c)\}\}\{c_1', c_2', c_3'\}, output\ \{a, c, c\})$

$(\{\emptyset,\{(c.2, b)\},\{(c.2, b),(c.1, a),(c.3, a)\}\}\{c_1', c_2', c_3'\}, output\ \{a, a, b\})$

$(\{\emptyset,\{(c.2, b)\},\{(c.2, b),(c.1, a),(c.3, b)\}\}\{c_1', c_2', c_3'\}, output\ \{a, b, b\})$

$\vdots$

$(\{\emptyset,\{(c.2, b)\},\{(c.2, b),(c.1, c),(c.3, c)\}\}\{c_1', c_2', c_3'\}, \textit{output}\ \{b, c, c\})$

$(\{\emptyset,\{(c.2, c)\},\{(c.2, c),(c.1, a),(c.3, a)\}\}\{c_1', c_2', c_3'\}, \textit{output}\ \{a, a, c\})$

$(\{\emptyset,\{(c.2, c)\},\{(c.2, c),(c.1, a),(c.3, b)\}\}\{c_1', c_2', c_3'\}, \textit{output}\ \{a, b, c\})$

$\vdots$

$(\{\emptyset,\{(c.2, c)\},\{(c.2, c),(c.1, c),(c.3, c)\}\}\{c_1', c_2', c_3'\}, \textit{output}\ \{c, c, c\})$.

Não se tem mais eventos pois nenhum dos eventos anteriores habilita outro evento

Fixada a estratégia anterior, e com o string *cab* como entrada, seria executada pelo programa a seguinte seqüência de operações (é uma instanciação da estratégia):

$(\{\emptyset\}\{c_1', c_2', c_3'\}, \textit{valof}\ \{c.2\})$,

$(\{\emptyset,\{(c.2, a)\}\}\{c_1', c_2', c_3'\}, \textit{valof}\ \{c.1, c.3\})$,

$(\{\emptyset,\{(c.2, a)\},\{(c.1, c),(c.2, a),(c.3, b)\}\}\{c_1', c_2', c_3'\}, \textit{output}\ \{ a, b, c\})$.

2. A estratégia de computação anterior computa primeiro c_2, seqüencialmente, e depois c_1 e c_3, paralelamente. Uma (das muitas) estratégia total seqüencial para o mesmo exemplo seria:

$Sort_{seq} = \{\ (\{\emptyset\}\{c_1', c_2', c_3'\}, \textit{valof}\ \{c.1\})$,

$(\{\emptyset,\{(c.1, a)\}\}\{c_1', c_2', c_3'\}, \textit{valof}\ \{c.2\})$,

$(\{\emptyset,\{(c.1, b)\}\}\{c_1', c_2', c_3'\}, \textit{valof}\ \{c.2\})$,

$(\{\emptyset,\{(c.1, c)\}\}\{c_1', c_2', c_3'\}, \textit{valof}\ \{c.2\})$,

$(\{\emptyset,\{(c.1, a)\},\{(c.1, a),(c.2, a)\}\}\{c_1', c_2', c_3'\}, \textit{valof}\ \{c.3\})$,

$(\{\emptyset,\{(c.1, a)\},\{(c.1, a),(c.2, b)\}\}\{c_1', c_2', c_3'\}, \textit{valof}\ \{c.3\})$,

$\vdots$

$(\{\emptyset,\{(c.1, c)\},\{(c.1, c),(c.2, b)\}\}\{c_1', c_2', c_3'\}, \textit{valof}\ \{c.3\})$,

$(\{\varnothing,\{(c.1, c)\},\{(c.1, c),(c.2, c)\}\}\{c_1', c_2', c_3'\}, \textit{valof}\,\{c.3\})$,

$(\{\varnothing,\{(c.1, a)\},\{(c.1, a),(c.2, a),(c.3, a)\}\}\{c_1', c_2', c_3'\}, \textit{output}\ \{a, a, a\})$,

$(\{\varnothing,\{(c.1, a)\},\{(c.1, a),(c.2, b),(c.3, a)\}\}\{c_1', c_2', c_3'\}, \textit{output}\ \{a, a, b\})$,

$\vdots$

$(\{\varnothing,\{(c.1, c)\},\{(c.1, c),(c.2, b),(c.3, c)\}\}\{c_1', c_2', c_3'\}, \textit{output}\ \{b, b, c\})$,

$(\{\varnothing,\{(c.1, c)\},\{(c.1, c),(c.2, c),(c.3, c)\}\}\{c_1', c_2', c_3'\}, \textit{output}\ \{c, c, c\})\ \}$

Com esta estratégia de computação total (seqüencial), se teria, para a mesma entrada *cab*, a seguinte instanciação:

$(\{\varnothing\}\{c_1', c_2', c_3'\}, \textit{valof}\,\{c.1\})$,

$(\{\varnothing,\{(c.1, c)\}\}\{c_1', c_2', c_3'\}, \textit{valof}\,\{c.2\})$,

$(\{\varnothing,\{(c.1, c)\},\{(c.1, c),(c.2, a)\}\}\{c_1', c_2', c_3'\}, \textit{valof}\,\{c.3\})$,

$(\{\varnothing,\{(c.1, c)\},\{(c.1, c),(c.2, a)\},\{(c.1, c),(c.2, a),(c.3, b)\}\}\{c_1', c_2', c_3'\}$,
$\textit{output}\ \{a, b, c\})$. ■

6.4.8 Composição de Estratégias

Suponha que se tem um programa P que chama duas funções F_1 e F_2. P é representado na Figura 6.4, onde se pode observar que P pode ser considerado como composto por cinco "segmentos": S_1, F_1, S_2, F_2, S_3. Claramente, com a definição de estratégia de computação (definição 6.4.3), se tem que, para cada um dos segmentos existe uma estratégia. Intuitivamente, a estratégia de computação total para P seria a composição das estratégias dos cinco segmentos, que corresponderia a uma semântica composicional. É obvio que, no contexto de P, qualquer estratégia de computação que computa cada um dos segmentos anteriores, é parcial. No caso de F_1 e F_2, se elas não dependem do ambiente de P (não têm variáveis livres), então as estratégias de F_1 e F_2 serão totais.

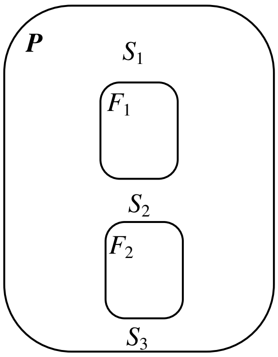


Figura 6.4 - Programa com subrotinas

Quais são as condições que devem cumprir, no exemplo, F_1 e S_2, para serem considerados partes do programa? Isto é, qual é a relação entre ambos? Claramente F_1 deve ser computado antes que S_2, e o estado final de F_1 deve ser o estado inicial de S_2. Em outras palavras, a composição de F_1 e S_2 também deve ser uma estratégia de computação. Formalmente:

6.4.8.1 Definição (composição de estratégias)

Seja $!M \multimap M' = (C, V, E, \vdash)$ uma cds. Seja $A \subseteq E$ um algoritmo linear, e a_1, a_2 estratégias de computação de A. Então $a_1 \circ a_2$ é a *composição* de a_1 e a_2 se e somente se

1. $a_1 \cup a_2$ é uma estratégia de computação, e

2. Não existe nenhuma computação de a_2 que preceda computacionalmente a alguma computação de a_1. ■

6.5 Relação entre algoritmos lineares e os seqüenciais

Mostra-se nesta seção como se relacionam as estratégias seqüenciais dos algoritmos lineares com os algoritmos seqüenciais de Curien.

6.5.1 Proposição

Seja *!M—∘M'* uma cds, *A* um estado de *!M—∘M'* (algoritmo linear). Então, $\{(X_1\{c_1'\},\ v_1'),\ldots,\ (X_i\{c_i'\},\ v_i')\} \subseteq\ !M \multimap M'$, é uma estratégia de computação seqüencial que computa a função subjacente f: $M \to M'$, se e somente se existe um algoritmo seqüencial (de Curien) que computa a mesma função f dada (i.e., o processo de computação tem a mesma seqüência de estados).

Prova)

A condição de que o conjunto das células de saída seja unitário, é justamente porque os algoritmos de Curien somente computam uma célula de saída. Para facilitar a notação, se escreverá $(X_ic_i',\ v_i')$ ao invés de $(X_i\{c_i'\},\ v_i')$.

⇒) Seja $A = \{(X_0c_o',\ v_o'),\ldots,\ (X_ic_i',\ v_i'),\ldots,\ (X_nc_n',\ v_n')\}$ e $a = \{(X_0\{c_i'\},\ v_0'),\ldots,\ (X_i\{c_i'\},\ v_i')\}$ a estratégia de computação seqüencial dada. Define-se um estado de uma cds $M \Rightarrow M'$ como um algoritmo seqüencial que computa f.

1. Para todo evento da forma $(X_ic_i',\ valof\ \{c_i'\})$[19], escreva um evento $(x_ic_i',\ valof\ c_i')$, onde $x_i = \sup(X_i)$, onde c_i é um índice de seqüencialidade para c_i';

2. Para todo evento da forma $(X_ic_i',\ output\ \{v_i'\})$, escreva um evento $(x_ic_i',\ output\ v_i')$, onde $x_i = \sup(X_i)$.

Claramente o conjunto de eventos assim definido é um algoritmo seqüencial de Curien.

⇐) Se $a = \{(x_i^0c_i,\ valof\ c_0'),\ldots,\ (x_i^mc_i,\ output\ u')\}$ é um algoritmo seqüencial que computa a função seqüencial f: $M \to M'$, então se pode construir um algoritmo linear A (ver seção 6.2.2) que computa a mesma função. Claramente uma das estratégias de A, será uma estratégia de computação seqüencial total (direto pela definição do algoritmo linear). ■

A proposição anterior fecha o diagrama da Figura 6.5, onde se tem que, as funções estáveis incluem as seqüenciais, e a partir das primeiras se podem obter

---

[19] Se omitirão os sub-índices e super-índices das células do valof, pois como é uma estratégia seqüencial, o sub-índice será sempre o supremo de X, e o super-índice será qualquer processador.

funções lineares aplicando o operador de linearização de Girard. A partir das funções seqüenciais é possível obter os algoritmos seqüências (seguindo o processo de definição de Curien); e finalmente, dos algoritmos seqüenciais e funções lineares, como foi feito neste trabalho, é possível definir os algoritmos lineares.

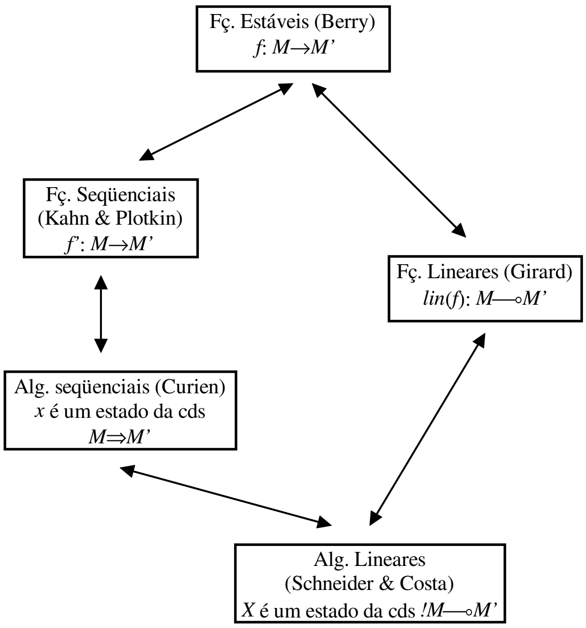


Figura 6.5 - Relação das funções estáveis e lineares com os alg. lineares

7 Conclusões e Trabalhos Futuros

Apresentam-se algumas conclusões sobre os conceitos definidos neste trabalho, assim como também algumas sugestões de continuidade em pesquisas futuras.

7.1 Os Espaços Coerentes

No capítulo 4 foram apresentados os espaços coerentes de Girard ([GIR 89]), assim como também os operadores de linearização e de "of course". É importante ressaltar que Girard, quando define o espaço coerente "of course", e no processo de linearização de uma função estável f: $M \to M'$, não estabelece qual é a imagem (de $lin(f)$) dos objetos X que não são imagem de nenhum objeto x de M. Isto é, Girard define $lin(f)(!x) = f(x)$, e deixa implícita que, a imagem dos outros objetos de $!D(M)$ que não têm pré-imagem em M, é estabelecida considerando que $lin(f)$ tem que ser linear. Neste trabalho se explicita essa condição, como sendo $lin(f)(\{a_1, ..., a_n\}) = \cup \{f(a_1), ..., f(a_n)\}$.

O operador "of course" pode ser utilizado não somente com o objetivo de linearizar uma função estável. De fato, o exemplo 6.4.7 mostra como obter estratégias de computação do algoritmo linear que computam a função *sort*, que não é estável.

No capítulo 5 foi estudada a relação entre os espaços coerentes, domínios concretos e domínios de eventos, assim como também a relação entre as partes "concretas" destes domínios: teias de espaços coerentes, estruturas de eventos e estruturas de dados concretas.

7.2 A importância dos Algoritmos Lineares

Pela definição dos *algoritmos lineares*, esses podem ser considerados como uma extensão dos algoritmos seqüenciais do Curien ([CUR 86]), no sentido que, são definidos também como cds, e que a partir das estratégias de computação seqüenciais dos algoritmos lineares é possível obter um algoritmo seqüencial do Curien. Os algoritmos lineares contém ainda estratégias de computação paralelas, o que permitiria aproveitar ao máximo os recursos de uma máquina paralela. As estratégias de computação de um algoritmo linear foram definidas de forma tal a ser possível obter uma semântica composicional, permitindo dar semântica a segmentos de programas. Esta é uma característica muito importante das estratégias de computação definidas neste trabalho.

Com a representação da Matemática Intervalar nos espaços coerentes ([DIM 95]), é possível obter agora um algoritmo linear que opere sobre intervalos, e desta forma estabelecer que uma função intervalar seria computável se existe um algoritmo linear que contém estratégias que a computam.

7.3 Sugestões para continuidade da pesquisa

Este trabalho deve ser considerado apenas como uma introdução, devendo servir como base para estudos mais avançados. A seguir se apresentam algumas sugestões:

1. Formalizar a máquina que computa os algoritmos lineares (Figura 6.2).

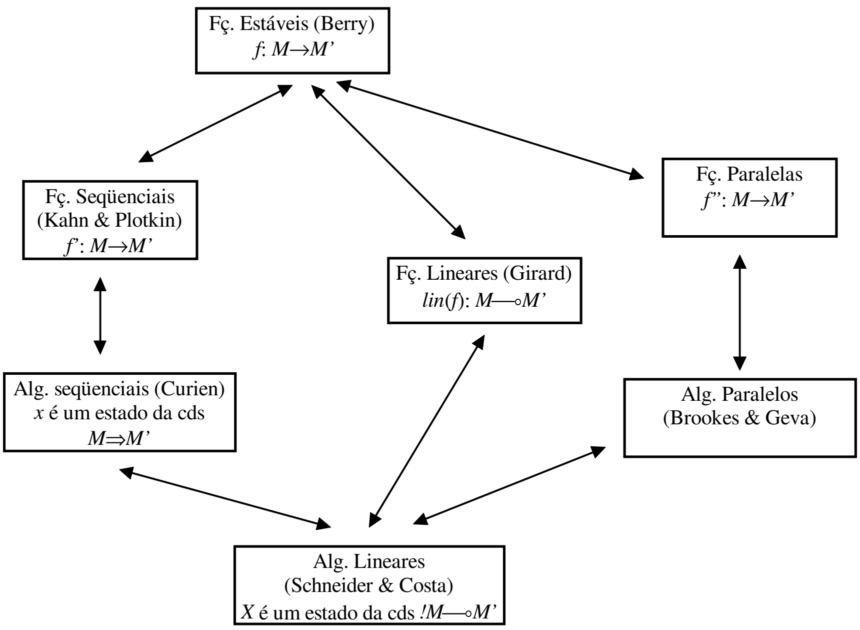
2. Seria interessante dar uma semântica operacional do tipo da aproximação estrutural do Plotkin ([PLO 81]), onde as regras têm a forma das regras de dedução natural de Gentzen.

3. Também seria possível dar uma semântica axiomática baseada na lógica linear, já que os algoritmos lineares podem ser considerados como sendo uma fórmula $!A \multimap B$. Desta forma seria possível determinar se uma estratégia de computação, por exemplo a, pertence a um algoritmo linear que computa uma função f: $A \rightarrow B$, se a é uma prova de $!A \multimap B$, isto é a: $!A \multimap B$.

4. Assim como Curien definiu a linguagem CDS0, a partir dos algoritmos seqüenciais ([CUR 86]), seria possível definir uma linguagem a partir dos algoritmos lineares seguindo o mesmo processo. Para isto, novamente seguindo [CUR 86], seria necessário definir a composição de algoritmos lineares e mostrar que a categoria das cds e os algoritmos lineares é cartesiana fechada.

5. Poderia considerar-se as estratégias de computação de segmentos de programas como especificando a corretude dos mesmos, e desta forma considerar que um programa é correto se cada uma das partes é, dado que a composição das estratégias parciais de um programa é uma estratégia total que o computa.

6. Brookes e Geva definiram em [BRO 90] algoritmos paralelos sobre estruturas de dados concretas, inspirados também em Berry e Curien. A idéia chave, no processo de generalização dos conceitos de Berry e Curien, é substituir o comando "valof" da exponenciação seqüencial com um comando "query", que dá como resultado sub-computações paralelas. Considera-se que, para todo algoritmo paralelo de Brookes & Geva que computa uma função *f* determinada, existe uma estratégia paralela do algoritmo linear gerado a partir de *f*. Seria interessante formalizar esta relação da mesma forma como se fez com os algoritmos seqüenciais. Possivelmente o seguinte quadro poderia ser encontrado:



8 ANEXO

Este anexo contém algumas das categorias de interesse na Computação Teórica.

CATEGORIA	OBJETOS	MORFISMOS	CCC
SET	Conjuntos	Funções Contínuas	CCC
CPO_C	Cpo's	Funções Contínuas	CCC
	Cpo's Consistentemente Completos	Funções Contínuas	CCC
SFP	Cpo's Fortemente Completos	Funções Contínuas	CCC
	Cpo's ω-algébricos e Consistentemente Compl.	Funções Contínuas	CCC
	Cpo's Coerentes e Primo-algébricos	Funções Contínuas	CCC
DF_S	dF-domínios	Funções Estáveis	CCC
DF_L	dF-domínios	Funções Lineares	
CD_C	Domínios Concretos	Funções Contínuas	
DCD_S	Domínios Concretos Distributivos	Funções Estáveis	

DCD_{Seq}	Domínios Concretos Distributivos	Funções Seqüenciais	
DCD_{SA}	Estruturas de Dados Concretas Distributivas	Algoritmos Seqüenciais	CCC
$QDCS_{SS}$	Domínios Qualitativos com Estrutura de Coerência	Funções Fortemente Estáveis	CCC
SEV_S	Estruturas de Eventos Estáveis	Funções Estáveis	CCC
SEV_L	Estruturas de Eventos Estáveis	Funções Lineares	
SEV_{Syn}*	Estruturas de Eventos Estáveis	Morfismos Parcialmente Sincrônicos	
SEV_{Syn}	Estruturas de Eventos Estáveis	Morfismos Sincrônicos	
SF_S	Famílias Estáveis	Funções Estáveis	CCC
SF_L	Famílias Estáveis	Funções Lineares	
COH	Famílias (Espaços) Coerentes	Funções Contínuas	
COH_S	Famílias (Espaços) Coerentes	Funções Estáveis	CCC
COH_L	Famílias (Espaços) Coerentes	Funções Lineares	
FF	Famílias Finitas	Mapeamentos Lineares	

9 BIBLIOGRAFIA

[ABR 93] ABRAMSKY, S. Computational interpretations of linear logic. **Theoretical Computer Science**, Amsterdam, v.111, n.1-2, p. 3-57, 1993.

[ACI 89] ACIÓLY, Benedito Melo. **Fundamentação Computacional da Matemática Intervalar.** Porto Alegre: CPGCC da UFRGS, 1989. 263 p.

[ACI 93] ACIÓLY, Benedito Melo; CLAUDIO, Dalcidio Moraes. **A Domain Approach to Interval Mathematics.** 1993. Texto não publicado.

[ARB 75] ARBIB, Michael A.; MANES, Ernest G. **Arrows, Structures and Functors:** The Categorical Imperative**.** New York: Academic Press, 1975. 185 p.

[ASP 90] ASPERTI, A. Stability and Computability in Coherent Domains. **Information and Computation**, Orlando, FL, v.86, p.115-139, 1990.

[BER 82] BERRY, G.; CURIEN, P.L. Sequential Algorithms on Concrete Data Structures. **Theoretical Computer Science**, Amsterdam, v.20, n.3, p. 265-321, 1982.

[BRO 90] BROOKES, Stephen; GEVA, S. **Towards a theory of parallel algorithms on concrete data structures.** Pittsburgh: Carnegie Mellon, 1990. 54 p.

[BRO 93] BROOKES, Stephen. Historical Introduction to "Concrete Domains". **Theoretical Computer Science**, Amsterdam, v.121, p. 179-186, 1993.

[BUC 93] BUCCIARELLI, Antonio. **Sequential models of PCF: some contributions to the Domain-theoretic approach to full abstraction.** Pisa: Università di Pisa-Genova-Udine, 1993. 153 p. Ph.D. Thesis.

[BUC 91] BUCCIARELLI, Antonio; EHRHARD, T. Sequentiality and strong stability. Annu. IEEE Symp. on Logic in Computer Science. **Proceedings...** [S.l.: s.n.], 1991

[CLA 94] CLAUDIO, Dalcidio Moraes; ESCARDÓ, Martin Hötzel. **Scott Domain Theory as a Foundation for Interval Analysis.** 1994. Texto não publicado.

[CLA 94a] CLAUDIO, Dalcidio Moraes; ACIÓLY, Benedito Melo. **Interval Orders Approximation.** 1994. Texto não publicado.

[CON 93] CONSTABLE, Robert L.; SMITH, Scott F. Computational Foundations of Basic Recursive Function Theory. **Theoretical Computer Science**, Amsterdam, v.121, p.89-112, 1993.

[COS 94] COSTA, A. C. da Rocha; MOREIRA, A. Freitas. **Resource-Flow Interpretation of Sequent Calculi.** Submetido no WOLLIC'94, Recife, Brasil, 1994. 18 p.

[CUR 86] CURIEN, P.L. **Categorical Combinators, Sequential Algorithms and Functional Programming.** [S.l.:s.n.], 1986. 300 p. (Research Notes in Theoretical Computer Science).

[DAV 90] DAVEY, B. A.; PRIESTLEY, H. A.. **Introduction to lattices and order.** Cambridge: University Press, c1990. 248 p.

[DIM 91] DIMURO, Graçaliz P. **Dominios intervalares da matemática computacional.** Porto Alegre: CPGCC da UFRGS, 1991. 321 p.

[DIM 94] DIMURO, Graçaliz P. **A teoria dos espaços coerentes.** 1994. 21 p. Texto não publicado.

[DIM 95] DIMURO, Graçaliz P. **Espaços coerentes de intervalos.** 1995. 65 p. Texto não publicado.

[DRO 89] DROSTE, M. Event structures and domains. **Theoretical Computer Science, Amsterdam**, v.68, p. 37-48, 1989.

[DRO 93] DROSTE, M. On stable domains. **Theoretical Computer Science, Amsterdam**, v.111, p. 89-102, 1993.

[ESC 90] ESCARDÓ, Martin Hötzel. **Números naturais parciais.** Porto Alegre: CPGCC da UFRGS, 1990.

[GIR 87] GIRARD, Jean Yves. Linear Logic. **Theoretical Computer Science**, Amsterdam, v.50, p.1-102, 1987.

[GIR 89] GIRARD, Jean Yves; TAYLOR, P.; LAFONT, Y.. **Proofs and types.** Cambridge: University Press, 1989. 175 p.

[HAN 92] HANNAN, John; MILLER, Dale. From Operational Semantics to Abstract Machines. **Mathematical Structures in Computer Science**, v.2, n.4, p.415-459. 1992.

[HAR 95] HARLAND, James; PYM, David; WINIKOFF, Michael. **Programming in Lygon: an overview.** (Disponível por e-mail jah@cs.rmit.edu.au, 1995. 15 p.).

[JUN 88] JUNG, A. **Cartesian closed categories of domains.** [S.l.:s.n.], 1988. Dissertation, FB Mathematik der TH Darmstadt.

[KAH 93] KAHN, G.; PLOTKIN, G. Concrete domains. **Theoretical Computer Science**, Amsterdam, v.121, p. 187-277, 1993.

[LAF 88] LAFONT, Y. The linear abstract machine. **Theoretical Computer Science**, Amsterdam, v.59, p. 157-180, 1988.

[LAM 94] LAMB, Luís da Cunha. **Introdução à teoria das categorias.** Porto Alegre: CPGCC da UFRGS, 1994. 45 p.

[LIM 69] LIMA, Elon Lages. **Elementos de Topología Geral.** Rio de Janeiro: IMPA, 1969. 463 p.

[LIN 93] LINCOLN, Patrick; SCEDROV, Andre; SHANKAR, Natarajan. Linearizing intuitionistic implication. **Annals of Pure and Applied Logic**, v.60, p.151-177. 1993.

[LOE 87] LOECKX, Jacques; SIEBER, Kurt. **The foundation of Program Verification.** Stuttgart: B.G. Teubner, c1987. 230 p. (Wiley - Teubner series in Computer Science).

[MAN 74] MANNA, Zohar. **Mathematical Theory of Computation.** New York: Mc Graw Hill, c1974. 448 p.

[MIL 77] MILNER, Robin. Fully abstract models of typed lambda-calculi. **Theoretical Computer Science**, Amsterdam, v.4, n.1, p.1-22, 1977.

[NIE 81] NIELSEN, M.; PLOTKIN, G.; WINSKEL, G. Petri nets, event structures and domains. **Theoretical Computer Science**, Amsterdam, v.13, p. 85-108, 1981.

[PIN 95] PINA, Michele; POIGNÉ, Axel. On the nature of events: another perspective in concurrency. **Theoretical Computer Science**, Amsterdam, v.138, p. 425-454, 1995.

[PLO 77] PLOTKIN, G. LCF considered as a programming language. **Theoretical Computer Science**, Amsterdam, v.5, n.3, p. 223-256, 1977.

[PLO 81] PLOTKIN, G. **A structural approach to operational semantics.** Denmark: University of Aarhus, 1981. 172 p.

[SCE 90] SCEDROV, Andre. A brief guide to Linear Logic. **Bulletin of the European Association for Theoretical Computer Science**, [S.l], v.41, p.154-165, 1990.

[SMY 77] SMYTH, M.B. Effectively given domains. **Theoretical Computer Science**, Amsterdam, v.5, n.3, p. 257-274, 1977.

[SMY 83] SMYTH, M.B. The largest cartesian closed category of domains. **Theoretical Computer Science**, Amsterdam, v.27, p.109-119, 1983.

[STO 69] STOLL, Robert R. **Set Theory and Logic.** San Francisco: W. H. Freeman, 1963. 474 p.

[STO 94] STOLTENBERG-HANSEN, Viggo; LINDSTRÖM, Ingrid; GRIFFOR, Edward R. **Mathematics Theory of Domains.** Cambridge: University Press, 1994. 349 p.

[STO 79] STOY, Joseph E. **Denotational Semantics:** the Scott-Strachey approach to programming language theory**.** Cambridge: MIT press, 1979. 414 p.

[TRO 92] TROELSTRA, A. S. **Lectures Notes on Linear Logic.** [S.l.:s.n.], 1992. (Lectures Notes, 29).

[TRO 92a] TROELSTRA, A. S. Tutorial on Linear Logic. In: Joint International Conference and Symposium on Logic Programming, 1992, Washington. **Proceedings...** [S.l.: s.n.], p.327-355, 1992.

[WAD 91] WADLER, Philip. Is there a use for linear logic? **SIGPLAN Notices**, [S.l.], p.255-273, June 1991. Trabalho apresentado no Symposium on Partial Evaluation and Semantics based program manipulation, 1991, New Harven.

[WAT 91] WATT, David A.; MUFFY, Thomas. **Programming Language, Syntax and Semantics.** New York: Prentice Hall, c1991. 389 p.

[WIN 81] WINSKEL, G. **Events in Computation.** Edinburgh: Edinburgh University, 1981. Ph.D. Thesis.

[ZHA 89] ZHANG, Guo-Qiang. **Logic of Domains.** Cambridge: University of Cambridge, 1989. 250 p. (Technical Report, n.185).

[ZHA 95] ZHANG, Guo-Qiang. **The largest cartesian closed category of stable domains.** Athens, Georgia: Department of Computer Science, The University of Georgia, 1995. 21 p.