Computabilidade no Espaço dos Intervalos Reais:

Um Modelo BSS Intervalar

AARÃO LYRA

Orientador: Prof. Dr. BENJAMÍN RENÉ CALLEJAS BEDREGAL

Dissertação apresentada ao Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte, para obtenção do título de Mestre em Sistemas e Computação.

NATAL
Rio Grande do Norte - Brasil
Fevereiro, 1999

Computabilidade no Espaço dos Intervalos Reais:

Um Modelo BSS Intervalar

Dissertação de Mestrado

Aarão Lyra

Orientado por Benjamín René Callejas Bedregal

Universidade Federal do Rio Grande do Norte
Departamento de Informática e Matemática Aplicada
Mestrado em Sistemas e Computação
Campus Universitário s/n, Lagoa Nova, Natal-RN
59072-970, Fax: (084)215-3780
aarao@ufrnet.ufrn.br

NATAL
Rio Grande do Norte - Brasil
Fevereiro, 1999

Banca Examinadora

....

Benjamin René Callejas Bedregal

....

Benedito Melo Acióly

....

David Paul Boris Déharbe

Dedico esta dissertação a minha esposa *Edna Melo dos Reis Lyra*, minhas filhas *Amanda Lyra Melo dos Reis* e *Aline Lyra Melo dos Reis*, e meus pais *Abrahão Lyra* e *Antonia Freitas Lyra*.

Agradecimentos

Após essa difícil jornada, resta-nos agradecer a todos aqueles que, em contextos diferentes, contribuíram para que chegássemos até aqui.

Na impossibilidade de citar todos os nomes que fizeram a travessia conosco nesta construção, permitimo-nos destacar os seguintes nomes, na certeza de que somos gratos a todos que direta ou indiretamente contribuíram para o sucesso deste trabalho.

Assim, apresentamos a profundidade dos reconhecimentos plenos:

A Deus, Artífice Maior do conhecimento.

Aos meus pais, *Abrahão Lyra e Antonia Freitas Lyra*, que depositaram em mim toda confiança me proporcionando a abertura deste caminho.

Aos meus irmãos *Abrahão Lyra Filho (in memorian), Aliete Lyra de Almeida, Arlete Lyra* e *Allyson Lyra*, pelo eterno companheirismo e amizade.

A minha esposa e filhas, *Edna Melo dos Reis Lyra, Amanda Lyra Melo dos Reis e Aline Lyra Melo dos Reis*, pelo estímulo, carinho, amor e abnegação necessários a concretização deste trabalho.

Ao professor orientador, *Benjamin René Callejas Bedregal*, por sua tão competente orientação, a minha admiração e agradecimento pelo apoio constante, pela confiança, amizade e, principalmente, pelo incentivo dedicado durante todo o decorrer do curso.

Aos professores do DIMAp, pelas significativas contribuições que enriqueceram este estudo. Em especial a professora *Márcia de Paiva Bastos Gottgtroy*, que muito me incentivou durante este curso, o seu apoio foi indispensável à realização deste trabalho. Ao professor *Benedito Melo Acióly*, nome que sempre me servirá de referência, pela sua presença amiga, por seus diversos artigos publicados que muito serviu neste trabalho e pelas diversas contribuições a mim ofertada. Aos professores *Dario José Aloise*, *David Paul Boris Déharbe* e *Anamaria Martins Moreira* cujo ensinamentos foi de grande valia.

Aos funcionários do DIMAp, em especial a *José Francisco, Ana Cláudia,* e *Aguiar*, que sempre contribuiram comigo, e sou eternamente grato.

Aos colegas, pela amizade e apoio ao longo do curso. Em especial a *Italo Rosendo* e *Selma*, pelas noites de estudo indispensáveis que passamos juntos.

Aos amigos, *Bruno José Santana* e *Sérgio Ramiro*, que me apoiaram e incentivaram nos momentos mais difíceis, os meus sinceros agradecimentos. A *José Cesário Filho*, pela paciência a mim dispensada. A *José Airton*, *Maria de Fátima Quinderé* e *Ubiratan (Bira)* pela amizade, compreensão e apoio ofertada no meu trabalho. A *Marcos José do Nascimento* pelo apoio nas correções ortográficas e pela eterna amizade.

Resumo

A Matemática Intervalar é uma teoria matemática originada na década do 60 com o objetivo de responder questões de exatidão e eficiência que surgem na prática da computação científica e na resolução de problemas numéricos. As abordagens clássicas para teoria da computabilidade tratam com problemas discretos (por exemplo, sobre os números naturais, números inteiros, strings sobre um alfabeto finito, grafos, etc.). No entanto, campos da matemática pura e aplicada tratam com problemas envolvendo números reais e números complexos. Isto acontece, por exemplo, em análise numérica, sistemas dinâmicos, geometria computacional e teoria da otimização. Assim, uma abordagem computacional para problemas contínuos é desejável, ou ainda necessária, para tratar formalmente com computações analógicas e computações científicas em geral. Na literatura existem diferentes abordagens para a computabilidade nos números reais, mas, uma importante diferença entre estas abordagens está na maneira como é representado o número real. Existem basicamente duas linhas de estudo da computabilidade no contínuo. Na primeira delas uma aproximação da saída com precisão arbitrária é computada a partir de uma aproximação razoável da entrada [Bra95]. A outra linha de pesquisa para computabilidade real foi desenvolvida por Blum, Shub e Smale [BSS89]. Nesta aproximação, as chamadas máquinas BSS, um número real é visto como uma entidade acabada e as funções computáveis são geradas a partir de uma classe de funções básicas (numa maneira similar às funções parciais recursivas). Nesta dissertação estudaremos o modelo BSS, usado para se caracterizar uma teoria da computabilidade sobre os números reais e estenderemos este para se modelar a computabilidade no espaço dos intervalos reais. Assim, aqui veremos uma aproximação para computabilidade intervalar epistemologicamente diferente da estudada por Bedregal e Acióly [Bed96,BA97a,BA97b], na qual um intervalo real é visto como o limite de intervalos racionais, e a computabilidade de uma função intervalar real depende da computabilidade de uma função sobre os intervalos racionais.

Palavras-chave: Computabilidade, Intervalos Reais, Modelo BSS e Espaços Contínuos.

Abstract

Interval Mathematics is a mathematical theory originated in the sixties and aims at solving accuracy and efficency issues that arise in scientific computation and in the resolution of numerical problems. The classic approach for computability theory deals with discrete problems (for example, on the natural numbers, integer numbers, strings on a finite alphabet, graphs, etc.). However, fields of pure and applied mathematics deal with problems involving real numbers and complex numbers. This happens, for example, in numerical analysis, dynamic systems, computational geometry and optimization theory. Thus, a computational approach for continuous problems is desirable, or even necessary, to deal formally with analogical computations and scientific computations in general. In the literature different approaches exist for the computability in the real numbers, and, an important difference among these approaches is the way the real number is represented. Basically, there are two lines of research in continuous computability. In the first estimation of, an output arbitrary accuracy is computed from an input aproximation [Bra95]. The second approach to real computability was developed by Blum, Shub and Smale [BSS89]. Where, the BSS machines, a real number is seen as a finished entity and the computable functions are generated starting from a class of basic functions (in a similar way to the partial recursive functions). In this dissertation we study this BSS model, used to characterize a computability theory on the real numbers and we extend it to model computability in the real intervals space. Thus, here we have developed a theoretical framework to study interval computability epistemologically different from the approach of Bedregal and Acióly [Bed96, BA97a, BA97b], where a real interval is seen as the limit of rational intervals, and the computability of an interval function depends on the computability of a function on the rational intervals.

key-words: Computability, Real Intervals, BSS Model and Continuous Spaces.

Índice

Capítulo 1

Introdução

A computação científica possui três tipos de erros. Ao primeiro chamamos *erro quanto a falta de precisão de equipamentos*, existe basicamente, devido à falta de precisão de dados nos equipamentos de medida que, aplicado nos computadores e sob operações repetitivas, se multiplica ocasionando vários problemas; esse tipo de erro só poderá ser sanado melhorando a precisão dos equipamentos de medida. Outro erro é o de *arredondamento*, acontece devido ao número finito de dígitos que uma máquina pode trabalhar; em computação científica diversas vezes precisamos computar operações com números que possuem grande quantidades de dígitos ou até mesmo infinitos (como uma dízima periódica[1] ou o valor de π, por exemplo), neste caso, a máquina arredonda este número a um outro número com quantidade de dígitos finita e capaz de computar, porém isto pode ocasionar erros futuros ao arrendondar e computar valores arrendondados várias vezes. O último erro existente é o de *truncamento*, uma máquina ao resolver um problema levará um determinado tempo para encontrar seu resultado, este tempo pode ser inviável, uma solução para este caso é o truncamento, ou seja, em algumas situações o processo da repetição da operação é truncada e o resultado é aproximado, esse resultado aproximado poderá ser aplicado em outras operações que multiplicarão o erro, e que resultará em um resultado incerto e impreciso.

A qualidade do resultado em computação científica depende do conhecimento e controle dos erros na computação. Algoritmos convencionais, chamados algoritmos pontuais, computam uma resposta e às vezes uma estimativa do erro. No entanto, o usuário não pode conseguir uma resposta exata sem o auxílio de um análise rigorosa dos erros, o qual é extensa, dispendiosa e nem sempre viável. Por outro lado, técnicas intervalares podem ser programadas em computadores de tal modo que a computação possua uma rigorosa e completa análise do erro no resultado.

A Matemática Intervalar é uma teoria matemática originada na década do 60

[1] Mesmo sabendo que uma dízima periódica pode ser reduzida a um número racional, na prática da computação científica ela será infinita, pois haverá necessidade de ser calculada

com o objetivo de responder questões de exatidão e eficiência que surgem na prática da computação científica e na resolução de problemas numéricos. A matemática intervalar se assenta em dois conceitos fundamentais: a propriedade da inclusão-monotonicidade de sua aritmética e uma topologia de Hausdorff que veremos no capítulo 2 desta dissertação, definida no conjunto de intervalos. A propriedade de inclusão-monotonicidade tem se revelado uma ferramenta útil na elaboração de algoritmos intervalares enquanto a topologia de Hausdorff não consegue refletir, consistentemente, as carasterísticas lógicas e topológicas daquela propriedade, porém, é a utilizada na prática em nossos dias. Em [Aci91] é introduzido o conceito de espaço dos reais parciais (intervalos de extremos reais) munidas da sua topologia de Scott cujos elementos são vistos como intervalos de informação. Cada elemento do espaço informa sobre um objeto real ou intervalo. Uma conseqüência desta mudança é a compatibilidade entre as propriedades da inclusão-monotonicidade e sua topologia de Scott.

As abordagens clássicas para a teoria da computabilidade tratam com problemas discretos (por exemplo, sobre os números naturais, números inteiros, strings sobre um alfabeto finito, ou sobre grafos, etc.). No entanto, os campos da matemática pura e aplicada tratam com problemas envolvendo números reais, números complexos, etc. Isto acontece, por exemplo, em análise numérica, sistemas dinâmicos, geometria computacional e teoria da otimização. Assim, uma abordagem computacional para problemas contínuos é desejável, ou ainda necessária, para tratar formalmente com computações analógicas e com computações científicas em geral.

A computabilidade sobre conjuntos contáveis é obtida a partir do desenvolvimento de uma teoria da computabilidade sobre os números naturais [BL74] de tal modo que a computabilidade nos outros conjuntos se reduz à computabilidade no conjunto dos números naturais. Como cada número real parcial pode ser visto como limite de uma seqüência convergente de intervalos reais e inversamente, podemos esperar que os números reais parciais tenham um papel numa teoria da computabilidade para conjuntos com a cardinalidade do contínuo semelhante a dos números naturais na teoria da computabilidade para conjuntos contáveis. Na literatura existem diferentes abordagens para a computabilidade nos números reais, mas, uma importante diferença entre estas abordagens está na maneira como é representado o número real [Gia93]. Num dos primeiros artigos sobre teoria dos domínios, D.Scott [Sco70] sugeriu que o cpo (conjunto completo parcialmente ordenado) definido no capítulo 5.2.3 consistindo de intervalos da reta real poderia ser usado para estudar computabilidade sobre os números reais. Os espaços em teoria dos domínios para estudar computabilidade são chamados de domínios (contínuos) efetivamente dados. Previamente Martin-Löf [ML70] construiu um espaço similar de aproximações. Em ambos os casos a reta real é mergulhada no espaço de aproximações onde a noção de computabilidade pode ser definida de uma maneira natural. Muitos resultados concernentes à teoria da computabilidade sobre os números reais podem ser obtidas neste contexto. Nesta abordagem uma aproximação da saída com precisão arbitrária é computada a partir de uma aproximação razoável da entrada [Bra95]. Uma outra linha de pesquisa para computabilidade real foi desenvolvida por Blum, Shub e Smale [BSS89]. Nesta aproximação, as chamadas máquinas BSS, um número real é visto como uma entidade acabada e as funções computáveis são geradas a partir de uma classe de funções básicas (numa maneira similar às funções parciais recursivas). Mesmo sabendo que cada uma destas aproximações tem seus méritos, nenhuma tem sido aceita pela maioria dos matemáticos e cientistas da computação.

Na primeira abordagem (de limite) as construções são baseadas na representação dos números reais como seqüências convergentes de intervalos racionais que apesar de não serem uma representação apropriada para implementar computações sobre números reais [Gia93] são bons para uma teoria da computabilidade sobre números reais (parciais).

Esta visão nos permite estender a conhecida máquina de Turing, desenvolvida para computações em mundos contáveis, para uma máquina chamada de **C**-Turing [BA97a] na qual esta nova noção de computabilidade foi fortemente sustentada na bem consolidada e aceita teoria dos domínios.

Nesta dissertação estudaremos o modelo BSS, desenvolvido por Blum, Shub e Smale [BSS89], usado para se caracterizar uma teoria da computabilidade sobre os números reais e estenderemos este para se modelar a computabilidade no espaço dos intervalos. Assim, aqui veremos uma aproximação para computabilidade intervalar epistemologicamente diferente da estudada por Bedregal e Acióly [Bed96,BA97a,BA97b], na qual um intervalo real é visto como o limite de intervalos racionais, e a computabilidade de uma função intervalar real depende da computabilidade de uma função sobre os intervalos racionais, isto é, ela se baseia no princípio de que podemos computar uma aproximação da saída com precisão arbitrária a partir de uma aproximação razoável da entrada.

Esta dissertação consta de 7 capítulos dispostos da seguinte maneira:

Capítulo 1- Esta Introdução.

Capítulo 2- *Fundamentação matemática.* Veremos neste capítulo o estudo das estruturas algébricas essenciais para o desenrolar da dissertação, as demonstrações fundamentais para a extensão do modelo BSS e conceitos básicos de topologia para podermos enquadrar o nosso novo modelo.

Capítulo 3- *Teoria da Computabilidade Clássica.* Veremos aqui, os seguintes fundamentos da teoria da computabilidade: definições básicas (relações, funções, recursividade, Algoritmos, Procedimentos Efetivos, etc), modelos de máquinas (URM, MAR e Turing) e a Tese de Church. Servirão como suporte teórico para o modelo

BSS Intervalar.

Capítulo 4- *O modelo BSS.* Neste capítulo apresentaremos o modelo BSS e sua teoria. Veremos a definição de fractais e sua aplicabilidade usando a máquina BSS como também alguns exemplos de modelos BSS e sua aplicação.

Capítulo 5- *Matemática Intervalar.* Faremos neste capítulo uma breve introdução à matemática intervalar, conheceremos as funções intervalares básicas, estudaremos a topologia relacionando sempre com o modelo BSS e com a estrutura matemática proposta, também veremos que todo anel é um quasi-anel ordenado.

Capítulo 6- *O Modelo BSS-Intervalar.* Neste momento, caracterizaremos o Modelo BSS para ser uma extensão intervalar, onde as operações acontecerão sob intervalos reais. Veremos alguns exemplos tradicionais importantes, como também a garantia de continuidade no modelo proposto.

Capítulo 7- *Conclusão.* Exporemos neste capítulo os resultados obtidos, faremos uma análise sob eles, levando em conta que estamos trabalhando sob uma nova estrutura, bem como, proporemos trabalhos que poderão dar continuidade aos nossos estudos.

Capítulo 2

Fundamentação Matemática

2.1 Introdução

Em nossa dissertação usaremos diversas vezes modelos matemático-computacionais que trabalha sobre determinados conjuntos algébricos. Para um melhor entendimento faremos neste primeiro momento uma abordagem matemática.

Este capítulo se divide em três seções .

Seção 2.1. Introdução . Esta Introdução .

.

Seção 2.2. Estruturas Algébricas. Esta seção esta dividida em 9 subseções , que servirão como introdução ao Matemática Algébrica, aqui, apresentaremos as estruturas algébricas básicas, dentre elas os quasi-anéis que servirá de elo entre o modelo BSS e o modelo BSS intervalar, e, faremos algumas demonstrações importantes.

Seção 2.3. Conceitos Básicos de Topologia. Esta seção possui apenas uma subseção . Nela veremos o ponto de vista clássico de topologia e da teoria da informação , isto é, espaços métricos, quasi-métricos, ultra-métricos, pseudo-quasi-métrico, etc.

2.2 Estruturas Algébricas

Definição 1 *Uma* **Estrutura Algébrica** *é um conjunto munido de operações com características comuns, seja quanto a transitividade, associatividade, comutatividade, etc.*

Veremos agora algumas Estruturas Algébricas:

2.2.1 Operação Binária

Definição 2 *Seja R um conjunto não vazio. Uma* **operação binária** *sobre R, $*$, é uma função que associa a cada par (a, b), constituído de elementos de R, um elemento $a * b$ em R.*

A estrutura algébrica mais simples que se pode pensar é um conjunto não vazio, R, com uma única operação definida sobre R. Esta operação pode ou não satisfazer algumas propriedades. Tal estrutura, quando a operação for binária, é chamada **grupóide**.

Exemplo 1 *Seja $\mathbb{R}$ o conjunto dos números reais. $(\mathbb{R}, +)$ é um grupóide com elemento neutro 0 e $(\mathbb{R}, \cdot)$ é um grupóide com elemento neutro 1. $(\mathbb{R}, -)$ é um grupóide, mas o elemento neutro 0 é apenas à direita ($a - 0 = a$, para todo $a \in R$). Fato similar ocorre para o grupóide $(\mathbb{R}, /)$, onde obviamente o divisor não pode ser nulo.*

2.2.2 Grupo

Definição 3 *Seja R um conjunto não vazio e $*$ uma operação binária definida sobre R. $(R, *)$ é um* **grupo** *se:*

i. Se $a, b, c \in R$, então $(a * b) * c = a * (b * c) \Rightarrow$ lei da associatividade.

ii. Existe $e \in R$ tal que $e * a = a * e = a$, $\forall a \in A \Rightarrow$ existência de elemento neutro.

iii. Existe $x \in R$ tal que $a * x = x * a = e$, $\forall a \in A \Rightarrow$ existência de inversos.

2.2.3 Anel

Definição 4 *Seja A um conjunto não vazio sobre o qual estão definidas duas operações binárias + e $\cdot$, denominadas, respectivamente, adição e multiplicação. Dizemos que $(A, +, \cdot)$ é um* **anel** *se $(A, +)$ for um grupo e as seguintes propriedades sejam satisfeitas para qualquer $a, b, c \in A$:*

i. $a + b = b + a \Rightarrow$ *lei da comutatividade da adição .*

ii. $(a \cdot b) \cdot c = a \cdot (b \cdot c) \Rightarrow$ *lei da associatividade da multiplicação .*

iii. $a \cdot (b + c) = a \cdot b + a \cdot c$, $(b + c) \cdot a = b \cdot a + c \cdot a \Rightarrow$ *leis distributivas.*

iv. Existe $1 \in A$ *tal que* $1a = a1 = a$, $\Rightarrow$ *existência de uma unidade.*

Observação: Será adotado neste trabalho a convenção de se escrever ab em lugar de $a \cdot b$.

2.2.4 Corpo

Definição 5 *Uma estrutura algébrica* $(A, +, \cdot)$ *é um* **corpo** *se* $(A, +, \cdot)$ *é um anel e* $\forall a, b, c \in A$ *valem as seguintes propriedades:*

i. $ab = ba \Rightarrow$ *lei da comutatividade da multiplicação*

ii. Se $a \in A$ *e* $a \neq \mathbb{O}$, *então existe* $a^{-1} \in A$ *tal que* $aa^{-1} = a^{-1}a = 1 \Rightarrow$ *existência de um inverso multiplicativo.*

2.2.5 Quasi-Anéis

Definição 6 *Seja* A *um conjunto não vazio sobre o qual estão definidas duas operações binárias* $+$ *e* $\cdot$, *denominadas, respectivamente, adição e multiplicação. Estas estruturas são* **quasi-anéis** *se para quaisquer* $a, b, c \in A$. $(A, +, \cdot)$ *As seguintes propriedades sejam satisfeitas:*

i. $a + b = b + a \Rightarrow$ *lei da comutatividade da adição .*

ii. Existe $\mathbb{O} \in A$ *tal que* $a + \mathbb{O} = \mathbb{O} + a = a$, $\Rightarrow$ *existência de um zero (elemento neutro).*

iii. Existe $1 \in A$ *tal que* $1a = a1 = a$, $\Rightarrow$ *existência de uma unidade.*

iv. Existe um elemento $k \in A$ *tal que:*

iv_1. $k + 1 = \mathbb{O}$

iv_2. $k(ab) = (ka)b = a(kb)$

iv_3. $k(a + b) = (ka) + (kb)$

Seja $N \subseteq A$, $\mathbb{O} \in N$ *e* $/ : A \times (A - N) \longrightarrow A$. *A estrutura* $(A, N, +, \cdot, /)$ *é um* **anel divisão** *se a divisão* $(/)$ *é uma operação binária bem definida e também valem:*

i. $a/1 = a$, $\forall a \in A$

ii. $\mathbb{O}/a = \mathbb{O}$, $\forall a \in (A - N)$

iii. $k(a/b) = (ka)/b = a/(kb)$, $\forall b \in (A - N)$

Em um quasi-anel ou quasi-anel divisão tem-se três elementos destacáveis: $\mathbb{O}, 1$ e k. Em se tratando de $\mathbb{R}$ ou $\mathbb{C}$, o conjunto dos números complexos, $k = -1$. São corpos $(\mathbb{R}, +, \cdot)$ e $(\mathbb{C}, +, \cdot)$ e, fazendo $N = \{\mathbb{O}\}, (\mathbb{R}, N, +, \cdot, /)$ são anéis divisão. Comparado com outras estruturas algébricas, nota-se que em um quasi-anel, além da inexistência das propriedades associativa e distributiva, não se assume a existência de elemento inverso, nem aditivo, nem multiplicativo.

2.2.6 Ordem

Definição 7 *Seja A um conjunto. Uma* **ordem** *(ou ordem parcial) sobre A é uma relação binária, $\leq$, sobre A tal que para todo $x, y \in A$,*

i. $x \leq x$, reflexiva;

ii. $x \leq y$ e $y \leq x$ implica $x = y$, anti-simétrica;

iii. $x \leq y$ e $y \leq z$ implica $x \leq z$, transitiva.

Estas condições são referentes, respectivamente, a reflexividade, anti-simetria e transitividade. Um conjunto com uma relação de ordem, $\leq$, é dito um **conjunto ordenado** (ou um conjunto parcialmente ordenado). Alguns autores chamam de *poset*, *partially ordered set*. Quando é necessário destacar a ordem escrevemos $(A, \leq)$.

Uma relação de ordem $\leq$ dá origem a uma relação de **desigualdade estrita**: $x < y$ em A se e somente se $x \leq y$ e $x \neq y$. Também $x \geq y$ é usado em lugar de $y \leq x$. O símbolo $\parallel$ é usado para denotar **não comparabilidade**, isto é, $x \parallel y$ se $x \nleq y$ e $y \nleq x$. Neste caso se diz que x e y são elementos não comparáveis.

Seja A um conjunto ordenado. Então A é uma **cadeia** se, para todo $x, y \in R$, ou $x \leq y$ ou $y \leq x$, isto é, quaisquer dois elementos de A são comparáveis. Nomes alternativos para cadeias são **conjunto linearmente ordenado** e **conjunto totalmente ordenado**.

Exemplo 2 *Seja R qualquer conjunto. O conjunto das partes (ou powerset) $\mathcal{P}(R)$, consistindo de todos os subconjuntos de R, é ordenado com respeito à inclusão de conjuntos. Para $A, B \in \mathcal{P}(R)$, $A \leq B$ se e somente se $A \subseteq B$. Porém $[\mathcal{P}(R), \subseteq]$ não é uma cadeia pois seja $R = \{1, 2\}$, então $\mathcal{P}(R) = \{0, \{1\}, \{2\}, \{1, 2\}\}$. Em $\mathcal{P}(R), \{1\} \nsubseteq \{2\}$ e $\{2\} \nsubseteq \{1\}$. Assim $\{1\}$ e $\{2\}$ não são comparáveis segundo a relação de ordem $\subseteq$.*

Exemplo 3 *A declaração de que o valor computado de $\pi = 3.14$ está correto até a segunda casa decimal, pode ser reescrita dizendo-se que π pertence a um particular intervalo fechado de $\mathbb{R}$. Considerando-se a coleção de todos os intervalos fechados da forma $X = [a, b]$ e $Y = [c, d]$; $X \leq Y$ se e somente se $Y \subseteq X$. Então, se $\pi \in X$, $\pi \in Y$ e $X \leq Y$ significa que, para π, Y contém pelo menos tanta informação quanto X.*

2.2.7 Anel Ordenado

Definição 8 *Um anel $[A, +, \cdot]$ é dito* **ordenado** *se A possui um subconjunto P que admite o seguinte:*

1. *e [1] $\notin P$.*
2. *se $a, b \in P$ então $a + b$, $a \cdot b \in P$.*
3. *Seja $\mathbb{O}$ o elemento neutro e $-a$ o inverso de $(A, +)$. Para todo $a \in A, a|\mathbb{O}$ e $a \in P$ ou $-a \in P$, nunca ambos.*

Se A é um anel ordenado e $a, b \in P$ ($P \subseteq A$, tal que, as condições 1 e 2 acima são satisfeitas), se $a > b$ então $a - b \in P$[2], isto é, $a \leq b$ se $a = b$ ou $b - a \in P$. Podemos definir a seguinte ordem sobre A:

Proposição 1 *Seja A um anel ordenado, então a relação $\leq$ definida por "$a \leq b$ é verdadeira, se e somente se $a = b$ ou $b - a \in P$" é uma relação de ordem sobre A.*

DEMONSTRAÇÃO:

$\Longrightarrow$ Quanto a ser uma ordem:

i. $x \leq x$

   Sendo $x, y \in A$ sabemos que $x \leq y$ se $x = y$ ou $y - x \in P$, onde P está contido em A. Desta forma podemos concluir que quando $x = y$, então $y \leq x$. Logo, como $x = x$, $x \leq x$.

ii. $x \leq y$ e $y \leq$ x $\Rightarrow x = y$

   $x \leq$ y $\rightarrow x = y$ ou $y - x \in P$

   $y \leq$ x $\rightarrow y = x$ ou $x - y \in P$

   se $x \leq y$ e $y \leq x$ então temos as seguintes possibilidades:

   1. $x = y$ e portanto satisfaz a afirmativa;
   2. $y = x$ e portanto $x = y$;
   3. $y - x \in P$ e $x - y \in P$

      Precisamos inicialmente provar que $y - x = -(x - y)$, porém

[1] onde e é o elemento neutro do grupo $[A, +]$

[2] Em nossa trabalho, consideraremos que $a - b = a + (-b)$

$$\begin{aligned}(y-x)+(x-y) &= (y+(-x))+(x+(-y)) \\ &= y+((-x)+(x+(-y))) \\ &= y+(((-x)+x)+(-y)) \\ &= y+(\mathbb{O}+(-y)) \\ &= y+(-y) \\ &= \mathbb{O} \qquad\qquad \to y-x=-(x-y)\end{aligned}$$

Como $y-x=-(x-y)$ e $(y-x) \in$ P $\to -(x-y) \in P$, porém $-(x-y)$ é a negação de $(x-y)$ que já pertencia a P, então, contradiz a propriedade de P na qual só um pertence, nunca ambos. Logo:

$y-x=x-y=\mathbb{O} \in P$, e portanto:

$x=y$ e $x \leq x$

iii. $x \leq y$ e $y \leq z \Rightarrow$ x $\leq z$

Como $x \leq y$ podemos afirmar que:

$(y-x) \in P$

e $y \leq z$ podemos afirmar que:

$(z-y) \in P$

Como ambos pertencem ao subconjunto P, concluimos que

$(y-x)+(z-y) \in P$.

Como: $(y-x)+(z-y)=z-x+(y-y)=z-x$

então $z-x \in P$ e portanto $x \leq z$ ∎

2.2.8 Quasi-Anel Fracamente Ordenado

Definição 9 *Um quasi-anel* $[A,+,\leq]$ *é* **fracamente ordenado** *se* $\leq$ *é uma ordem sobre* A *e* $\forall a,b,c \in A$, *valem:*

i. $a \leq b \Longrightarrow a+c \leq b+c$

ii. $a \leq b \Longrightarrow kb \leq ka$

2.2.9 Quasi-Anel Ordenado

Definição 10 *Um quasi-anel fracamente ordenado* $[A,+,\leq]$ *é* **ordenado** *se* $\forall a,b,c \in A$:

i. $\mathbb{O} \leq a \leq b$ *e* $c > \mathbb{O} \Longrightarrow ac \leq bc$ *e* $ca \leq cb$

Existem definições similares para quasi-anéis divisão.

Proposição 2 *Todo anel é um quasi-anel.*

Demonstração: Seja $a, b, c \in A$ onde $[A, +, \cdot]$ é um anel. Precisamos demonstrar os itens abaixo:

i. $a + b = b + a$. Esta afirmativa é verdadeira, pois, por $[A, +, \cdot]$ ser um anel, esta hipótese já é admitida (lei da comutatividade da adição nos anéis).

ii. Existe $\mathbb{O} \in A$ tal que $a + \mathbb{O} = \mathbb{O} + a \ \forall \ a \in A$.

   Como $[A, +, \cdot]$ é um anel, podemos afirmar que $[A, +]$ é um grupo admitindo a existência do elemento neutro "e", onde: $a + e = e + a = a$. Desta forma, é admitido a existência do $\mathbb{O}$ onde para adição $e = \mathbb{O}$.

iii. Existe $1 \in A$ tal que $1a = a1 = a$. Esta afirmativa é verdadeira pois devido a A ser um anel esta hipótese já é admitida.

iv. Dado φ inverso aditivo do elemento 1, necessitamos demonstrar os sub-itens abaixo:

   iv$_1$. $\varphi + 1 = \mathbb{O}$.
   Como sabemos que $[A, +]$ é um grupo sabemos da existência de inversos aditivos nestes casos, logo existe o inverso aditivo de A que satisfaz a hipótese acima.

   iv$_2$. $\varphi(ab) = (\varphi a)b = (a\varphi)b = a(\varphi b)$.
   A afirmativa é verdadeira pois, por $[A, +, \cdot]$ ser um anel, esta hipótese já é admitida (lei da associatividade nos anéis).

   iv$_3$. $\varphi(a + b) = (\varphi a) + (\varphi b)$
   A afirmativa é verdadeira pois, por $[A, +, \cdot]$ ser um anel, esta hipótese já é admitida (lei das distributivas nos anéis) ∎

Proposição 3 *Todo anel ordenado é um quasi-anel ordenado.*

Demonstração: Já provamos que todo anel é quasi-anel, sendo o anel ordenado uma classe de anel, podemos afirmar que todo anel ordenado também é um quasi-anel. Demonstraremos agora que é quasi-anel ordenado.

Ressaltamos que um quasi-anel ordenado é primeiramente, uma ordem e é fracamente ordenado, só após demonstrarmos essas características provaremos que é ordenado. Seja $x, y, z \in A$, onde A é um anel ordenado.

$\Longrightarrow$ Quanto a ser uma ordem:

Demonstrado na Proposição 1.

$\Longrightarrow$ quanto a ser fracamente ordenado

i. $a \leq b \rightarrow a + c \leq b + c$

$a \leq b \rightarrow b - a \in P$. Como

$b + c - (a + c) = b - a + c - c = b - a$,

$b + c - (a + c) \in P$. Portanto

$a + c \leq b + c$

ii. $a \leq b \rightarrow kb \leq ka$

$k \notin P$ por definição de P

$\rightarrow -k$ e $(b - a) \in P$ e:

$-k(b - a) \in P$

$$\begin{aligned} -k(b-a) &= -k(-(a-b)) \\ &= -(-(k(a-b)) \\ &= k(a-b) \\ &= ka - kb \in \mathrm{P} \\ &\rightarrow kb \leq ka \end{aligned}$$

$\Longrightarrow$ quanto a ser ordenado

iii. se $\mathbb{O} \leq a \leq b$ e $c > \mathbb{O}$ então $ac \leq bc$ e $ca \leq cb$

$a - 0 = a \in P$,

$b - a \in P$,

$b - 0 = b \in P$,

$c - 0 = c \in P$. Logo, pela propriedade de P,

$ac \in P$ e $bc \in P$.

Precisamos provar que $bc - ac \in P$ e $cb - ca \in P$

ora,

$$\begin{aligned} bc - ac &= \{b + (-a)\}c & \\ &= \{b - a\}c, & \text{como} \\ &b - a \in \mathrm{P} & \\ &c \in P, & \text{portanto} \\ &\rightarrow (b-a)c \in P. & \text{Portanto} \\ &bc - ac \in \mathrm{P} & \end{aligned}$$

analogamente demonstraremos para $cb - ca \in P$

$$\begin{aligned} cb - ca &= c\{b + (-a)\} \\ &= c\{b - a\} \\ &c \in \mathrm{P} \\ &b - a \in \mathrm{P} \\ &\rightarrow c(b - a) \in \mathrm{P} \rightarrow cb - ca \in \mathrm{P} \blacksquare \end{aligned}$$

2.3 Conceitos Básicos de Topologia

Uma **topologia** sob um conjunto S é uma coleção de subconjuntos de S que é fechado sob interseção finita e união arbitrária. Um conjunto S, com uma topologia T em S, é um **espaço topológico** (S, T). Os elementos de T são conjuntos abertos do espaço. Formalmente:

A seguir veremos alguns exemplos de espaços topológicos conhecidos.

1. Seja $\mathbb{R}$ o conjunto de números reais e $T = \{x \subseteq \mathbb{R} : x$ é a união de intervalos abertos em $\mathbb{R}\}$. Claremente, $\langle \mathbb{R}, T \rangle$ é um espaço topológico. Esta topologia é conhecida como a topologia usual da reta, topologia canônica sobre $\mathbb{R}$ ou topologia Euclidiana.

2. Seja X qualquer conjunto. $P(X)$ é uma topologia sobre X, chamada a topologia mais fina sobre X, pois esta topologia contém o maior número de conjuntos abertos sobre X. $\langle X, P(X) \rangle$ é um espaço topológico conhecido como discreto.

3. Seja X um conjunto não vazio. A coleção $T = \{X, \emptyset\}$ também, é uma topologia sobre X, chamada trivial, ou topologia indiscreta, pois ela contém o menor número de conjuntos abertos.

Uma **base de uma topologia** T é um subconjunto β de T tal que todo conjunto aberto é a união de elementos de β. Um espaço topológico tendo uma *base contável* diz-se $2^{\underline{o}}$ **contável**. Se $A \subseteq P(S)$, e T é a menor topologia que contém A, dizemos que A é uma **sub-base** de T.

Toda base de T é também uma sub-base de T; por outro lado, A é uma sub-base de T se e somente se a coleção de interseções finitas de elementos de A é uma base de T.

2.3.1 Espaços Métricos e Quasi-Métricos

Uma classe importante de espaços topológicos são os espaços métricos e quasi-métricos.

Seja X um conjunto, $d : X \times X \rightarrow \mathbb{R}^{0+}$ uma função sobre a reta real não-negativa. Consideraremos um números de condições, ou axiomas, a respeito de (X, d).

1. $d(x,x) = 0$ (reflexividade)
2. i. $d(x,z) \leq d(x,y) + d(y,z)$ (desigualdade triangular)
   ii. $d(x,z) \leq max\{d(x,y), d(y,z)\}$ (desigualdade triangular)
3. $d(x,y) = d(y,x)$ (simetria)
4. i. $d(x,y) = d(y,x) = 0 \to x = y$ (identidade simétrica dos indicerníveis)
   ii. $(d(x,y) = 0) \to x = y$. (identidade dos indicerníveis)

Definição 11 *Seja $\mathbb{R}_0^+$ o conjunto dos números reais positivos incluindo o zero, X um conjunto e $d : X \times X \to \mathbb{R}_0^+$ uma função. O par $\langle X, d\rangle$ é um* **espaço métrico** *se d satisfaz os axiomas de simetria, reflexidade, desigualdade triangular (item i) e identidade dos indicerníveis.*

As métricas que acontecem tipicamente em semântica satisfazendo a desigualdade forte $2(ii)$. Neste caso falamos de um **espaço ultra-métrico** . Devemos mostrar que a noção de um espaço quasi-métrico é definida, frequentemente, de modo mais restritivo da que foi adotada aqui, isto pela combinação $1 + 2(i) + 4(i)$. Finalmente, se o axioma 4 é desprezado de quaisquer das formulações , o enfraquecimento resultante é expressado por meio do prefixo "pseudo-"; assim:

Definição 12 *Seja $\mathbb{R}_0^+$ o conjunto dos números reais positivos incluindo o zero, X um conjunto e $d : X \times X \to \mathbb{R}_0^+$ uma função. O par $\langle X, d\rangle$ é um* **espaço pseudo-métrico** *se d satisfaz os axiomas de reflexidade, desigualdade triangular (itens i e ii) e simetria.*

Definição 13 *Seja $\mathbb{R}_0^+$ o conjunto dos números reais positivos incluindo o zero, X um conjunto e $d : X \times X \to \mathbb{R}_0^+$ uma função. O par $\langle X, d\rangle$ é um* **espaço pseudo-quasi-métrico** *se d satisfaz os axiomas de reflexidade e desigualdade triangular (itens i e ii).*

Como dos axiomas de simetria e identidade dos indicerníveis podemos deduzir o axioma dos indicerníveis simétrica, cada espaço métrico é, também, um espaço quasi-métrico. Cada espaço quasi-métrico induz um espaço topológico.

Seja $\langle X, q\rangle$ um espaço quasi-métrico. A **topologia usual induzida** sobre X é definida tomando o subconjunto, $O \subseteq X$, como sendo aberto básico, se somente se, para qualquer $x \in O$ implica que $B_\epsilon(x) \subseteq O$ para algum $\epsilon > \emptyset$, onde $B_\epsilon(x) = \{y : q(x,y) < \epsilon\}$ é denominada de bola aberta de raio ϵ e centro x. No caso, os abertos $B_\epsilon x$, evidentemente, constituem uma base de conjuntos abertos para a topologia.

Seja $\langle X, T\rangle$ um espaço topológico. Podemos olhar para os conjuntos abertos em T como propriedade dos pontos em X, isto é, a topologia T classifica os elementos em X. Assim, a topologia nos permite distinguir ou mostrar a equivalência entre dois pontos pelas suas propriedades descritas na topologia. Portanto, podemos classificar os espaços topológicos por condições de separação sobre seus pontos.

Definição 14 *Seja $\langle X, T\rangle$ um espaço topológico. Se para qualquer $x \neq y \in X$ existe um conjunto aberto que contém um, mas não ambos (x e y), o espaço X diz-se T_0. Se existe um conjunto aberto que contém x mas não y o espaço X diz-se T_1. Finalmente, se existem conjuntos abertos disjuntos contendo x e y, respectivamente, o espaço X diz-se T_2 ou* **Hausdorff**.

É claro que cada espaço T_2 é T_1 e cada espaço T_1 é T_0. Algumas das condições de separação usuais (as quais incluem certas condições não definidas aqui, tais como T_3, $T_{3\frac{1}{3}}$, etc.) são de pouca significância no contexto de ciência da computação. Uma observação importante, é que para os espaços 2^o-contáveis (que incluem todos os espaços de significância computacional) as condições de separação de T_3 para cima são todas equivalentes. A hierarquia de condições de separação é, portanto, de pouco interesse em teoria da computação.

A topologia induzida por uma métrica é T_2, isto é, um espaço de Hausdorff, por outro lado a topologia induzida por uma quasi-métrica é um espaço T_0. Podemos obter um espaço métrico $\langle X, q^*\rangle$, de um espaço quasi-métrico $\langle X, q\rangle$, tomando:

$$q^*(x, y) = Max\{q(x, y), q(y, x)\}$$

Definição 15 *Um Espaço Topológico (X, T) é* **quasi-metrizável (metrizável)** *se podermos definir uma quasi-métrica (métrica) sobre X tal que T é a topologia quasi-métrica (métrica).*

Similarmente, podemos definir ultra-metrizável, pseudo-metrizável, etc.

Capítulo 3

Teoria de Computabilidade Clássica

3.1 Introdução

Vimos em todo o capítulo 2 uma abordagem matemática fundamental ao bom entendimento desta dissertação. Porém, em nosso trabalho, essas definições matemáticas estão associadas a uma abordagem computacional se tornando de igual importância uma nova fundamentação que tratará o assunto sob uma nova visão.

Começaremos esta seção com paradigmas computacionais existentes, resumindo-os até chegar a versões simplificadas de cada paradigma que contém todas as características fundamentalmente importantes. As versões simplificadas serão mostradas para serem equivalentes em um sentido forte. Isto vai sugerir uma visão de modelo-independente da computação em termos de "programas" que computam funções.

O primeiro caso de computação que apresentaremos é a noção que argumentos podem ser de tipos diferentes. Todas as nossas entradas, saídas, variáveis temporárias, etc. serão números naturais (membros de $\mathbb{N}$). Discutiremos informalmente que todos os outros tipos de argumentos, comumente usados, são incluídos somente para conveniência de programação e não são essenciais à computação. O argumento é baseado em como computadores codificam tudo em sucessões de bits. Argumentos Booleanos podem ser representados usando os dois primeiros números de $\mathbb{N}$: 0 e 1. Números de ponto flutuantes, como conseqüência de sua representação finita, podem ser vistos como números racionais. Os Números racionais são pares de números naturais que podem ser codificados como números naturais através da bijeção [LB74]:

Bijeção de $N^2 \to N$

$$< n, m >= n + \sum_{i=0}^{n+1} i = m + \frac{1}{2}(n+m)(n+m+1)$$

Por conseguinte, $\mathbb{N}$ pode representar os números racionais, e consequentemente, os números de ponto flutuantes. Os Números naturais também podem representar cadeias de caracteres por um índice, ou posição, em alguma lista padrão de todas as cadeias. Por exemplo, uma lista padrão de todas as cadeias que usam o alfabeto $\{a, \cdots, z\}$ começaria associando 0 com a cadeia vazia e enumerando as cadeias em ordem lexicográfica: $a, b, \cdots, z, aa, ab, \cdots, az, ba, \cdots$.

A computação analógica também pode ser vista computando com números naturais. Isto segue da observação de que qualquer nível de voltagem só pode ser medido em incrementos determinados pelo dispositivo medidor. Embora as voltagens sejam teoricamente contínuas, todos nossos dispositivos para medir voltagens o fazem com valores discretos. Para exemplos comuns de torneamento de informação essencialmente analógica em uma representação numérica, não precisamos olhar nenhum caso em que se codificou música em tecnologia de disco compacto ou o que codificou imagens de televisão de alta definição e gravações de discos compactos. Estamos agora prontos para apresentar nosso primeiro modelo de computação.

Desta forma veremos algumas definições computacionais básicas e imprescindíveis para o desenvolvimento desta dissertação.

Este Capítulo foi dividido em 8 seções sendo as principais fontes de pesquisas: [BL74], [Bra95] e [Cho90].

Seção 3.1. Introdução . Esta introdução .

Seção 3.2. Conceitos Básicos. Definiremos aqui os conceitos de funções , relações , definições recursivas, onde apresentaremos um exemplo, relações e predicados e relações de equivalência.

Seção 3.3. Automatos. Apresentaremos nesta seção a definição de Automato Finito.

Seção 3.4. Algoritmos e Procedimentos Efetivos. Diferenciaremos os conceitos de Algoritmos e Procedimentos Efetivos e mostraremos alguns exemplos.

Seção 3.5. A Máquina Sem Limites de Registros (URM). Apresentaremos aqui este tradicional modelo computacional, alguns exemplos, figuras, computações neste modelo, algumas funções computáveis sob o modelo URM, fluxogramas e observaremos a computabilidade em outros domínios.

Seção 3.6. Funções Parciais Recursivas. Apresentaremos as funções parciais recursivas, definiremos união ou concatenação , apresentaremos as operações de Substituição , Recursão, Minimalização , bem como diversos exemplos, figuras e modelos.

Seção 3.7. Outras Definições Sobre Computabilidade. Nesta seção apresentaremos as Máquinas de Acesso Randômico, as Máquinas de Turing, veremos a representação esquemática da Máquina de Turing, estudaremos as funções Turing-computáveis, observaremos vários exemplos e figuras.

Seção 3.8. A Tese de Church. Finalmente estudaremos a Tese de Church, veremos que existem dois métodos para provar que uma função f é URM-computável e veremos alguns exemplos importantes.

3.2 Conceitos Básicos

3.2.1 Funções

Assumimos que o leitor esteja familiarizado com a idéia básica de função e a distinção entre uma função f e um valor particular $f(x)$ a um determinado x, onde f é definida (normalmente em textos matemáticos uma função f é definida para ser um conjunto de pares ordenado que se $(x, y) \in f$ e $(x, z) \in f$, então $y = z$, e $f(x)$ é definido para ser este y).

Não insistimos nesta definição de função, mas nossa exposição é consistente com ela.

Uma **relação** $f : X \to Y$ de X em Y é uma coleção de pares ordenados $(x, y) \in X \times Y$.

Uma relação $f : X \to Y$ é uma **função** se $f(x)$ corresponde a, no máximo, um único valor para cada $x \in X$, ou seja, cada entrada produz uma única saída.

Se f é uma função, o **domínio** de f é o conjunto $\{x : f(x)$ é definida $\}$, denotado por $Dom(f)$ e dizemos que $f(x)$ é **indefinida** se $x \notin Dom(f)$. O conjunto $\{f(x)/x \in Dom(f)\}$ é chamado de **contra-domínio** ou **imagem** de f, denotado por $Im(f)$. Se A e B são conjuntos, dizemos que f é uma função de A para B se $Dom(f) \subseteq A$ e $Im(f) \subseteq B$. Usamos a anotação f : A $\to B$ para significar que f é uma função de A para B com $Dom(f) = A$. Uma função f é dita **injetiva** sempre que $x, y \in Dom(f)$ e $x \neq y$, então $f(x) \neq f(y)$. Se é injetiva, então f^{-1} denota o inverso de f, a única função g tal que $Dom(g) = Im(f)$ e $g(f(x)) = x$ para todo $x \in Dom(f)$. Uma função f de A para B é **sobrejetora** se $Im(f) = B$. É **bijetora** se é injetiva e sobrejetora.

Se $domf = X$, então f é chamada **função total**. Se f é definida parcialmente, e não para todo $x \in X$, então dizemos que f é uma **função parcial**.

Suponha que f é uma função e X um conjunto. *A* restrição de f para X, denotado por $f|X$, é a função com domínio $X \cap Dom(f)$ cujo valor de $X \cap Dom(f)$ é $f(x)$, isto é, $f|X(x) = f(x)$ para todo $x \in X \cap Dom(f)$. Escrevemos $f(X)$ para $Im(f|X)$. Se Y é um conjunto, então a imagem inversa de Y sob f é o conjunto $f^{(-1)}(Y) = \{x : f(x) \in Y\}$. (Note que isto é definido até mesmo quando f não é injetiva).

Denotamos por **f0** a função que não é definida em nenhuma parte; $f0$ tem a propriedade $Dom(f0) = Im(f0) = \emptyset$. Claramente, $f0 = g|\emptyset$ para qualquer função g.

Frequentemente em computabilidade encontraremos funções, ou expressões que envolvem funções que não são definidas para todos os valores. Em tais situações a notação seguinte é muito útil. Suponha que $\alpha(x)$ e $\beta(x)$ são expressões que envolvem as variáveis $x = (x_1, ..., x_n)$. Então escrevemos $\alpha(x) \cong \beta(x)$ para significar que para qualquer x, a expressão $\alpha(x)$ e $\beta(x)$ são definidas, ou ambos indefinidas, e se definidas, elas são iguais, Assim, por exemplo, se f, g são funções. $f(x) \cong g(x)$ é outro modo de dizer que $f = g$; e para qualquer número y, $f(x) \cong y$ que $f(x)$ é definido e $f(x) = y$ (desde que y sempre seja definido).

3.2.2 Definições Recursivas

Uma das ferramentas matemáticas que conhecemos, porém pouco utilizada em matemática, é um método de definir conjuntos chamado **definição recursiva**. Uma definição recursiva é caracteristicamente um processo com três etapas. Primeiro, especificamos alguns objetos básicos do conjunto. Segundo, damos regras para construir mais objetos no conjunto a partir dos que conhecemos. Terceiro, declaramos que nenhum objeto diferente dos construídos são permitidos no conjunto.

Observe o seguinte exemplo.

Exemplo 4 *Suponha que estamos tentando definir o conjunto de números pares para alguém que reconhece a aritmética mas nunca ouviu falar destes. Um modo simples de definir este conjunto é: É o conjunto de todos os números inteiros positivos divisível por 2. Outro modo que poderíamos tentar seria:*

É o conjunto de todos os 2n, onde n = 1 2 3 4...

O terceiro método que apresentamos se refere à definição recursiva:

O conjunto é definido por estas três regras:

i. regra 1: 0 é par.

ii. regra 2: se x é par então é $x + 2$ também é.

iii. regra 3: os únicos elementos no conjunto são os que podem ser produzidos através das duas regras anteriores.

Existe uma razão que explica o motivo da terceira definição ser a menos popular: é mais difícil de usar na maioria das aplicações práticas. Por exemplo, supondo que queremos provar que 14 é um elemento do conjunto. Usando a primeira definição, dividimos 14 por 2 e não há nenhum resto. Então é par. Para provar que 14 é par pela segunda definição temos que propor o número 7 de alguma maneira, então 14=(2)(7), sabemos que é par. Provar que 14 é par usando a definição recursiva é um processo mais longo. Poderíamos proceder como abaixo:

i. *Através da regra 1, sabemos que 2 é par.*

ii. *Então através da regra 2 sabemos que 2+2=4 também é par.*

iii. *Novamente através da regra 2 sabemos que se 4 é par, então 4+2=6 também é par.*

iv. *Novamente através de regra 2 sabemos que se 6 é par, então 6+2=8 também é par.*

v. *Agora aplicando a regra 2 a 8 nós concluímos que 8+2=10 é par. Aplicando novamente a regra 2, agora para 10, deduzimos que 10+2=12 é par.*

vi. *E, afinal, aplicando a regra 2 mais uma vez, para o número 12, concluímos que 12+2=14 é, realmente, par.*

Bem extensa. Porém, esta não é a única forma de definir o conjunto dos números pares por definição recursiva. Ainda poderíamos usar:
O conjunto dos números pares pode ser definido por estas três regras:

i. *Regra 1: 2 é par*

ii. *Regra 2: se x e y são pares, então $x + y$ é par.*

iii. *Regra 3: Nenhum número é par sem que possa ser produzido através de regras 1 e 2.*

Deve ser entendido que também podemos aplicar a Regra 2 para o caso onde x e y representem o mesmo número.
Podemos provar agora que 14 é par em menos passos:

1. *Através de regra 1: 2 é par*

2. *Através de regra 2: x=2, y=2, 4 é par*

3. *Através de regra 2: x=2, y=4, 6 é par*

4. *Através de regra 2: x=4, y=4, 8 é par*

5. *Através de regra 2: x=6, y=8, 14 é par.*

Esta é a definição recursiva que define melhor o conjunto de números pares, porque produz o conjunto em menor número de passos.

Notamos que a segunda definição recursiva ainda é mais complexa (provando que aqueles determinados números são pares) que as duas funções não recursivas. Por exemplo, suponha que queremos provar que a soma de dois números pares também é par. Esta é uma conclusão trivial da segunda função recursiva, mas provar isto é decididamente mais complexo. Para termos uma função recursiva dependemos de duas coisas: como será suficientemente comprensível as outras possíveis definições de entender; e que tipos de teoremas que poderíamos desejar provar sobre o conjunto a ser formado.

3.2.3 Relações e Predicados

Seja A um conjunto. Uma propriedade P que é verdadeira para algumas n-tuplas de A^n e falsa para as outras é chamada uma **relação** ou **predicado** n-ário em A.

Por exemplo, a propriedade $x < y$ é uma relação binária (ou predicado) em $\mathbb{N}$; $2 < 3$ é verdadeiro e $9 < 5$ é falso. Como outro exemplo, qualquer função n-ária f de $\mathbb{N}^n$ em $\mathbb{N}$ dá lugar a um predicado $(n+1)$-ário definido por $P(x_1, \cdots, x_n, y)$ se e somente se $f(x_1, \cdots, x_n) = y$.

3.2.4 Relações de Equivalência

Uma relação binária R sobre um conjunto A é chamado uma **relação de equivalência** se satisfaz as seguintes propriedades para todo $x, y, z \in A$:

i. (reflexividade) $R(x, x)$;

ii. (simetria) se $R(x, y)$ então $R(y, x)$;

iii. (transitividade) se $R(x, y)$ e $R(y, z)$ então $R(x, z)$.

Pensamos em $R(x, y)$ como dizendo que x, y são equivalentes (em algum sentido). Então definimos a classe de equivalência de x como o conjunto $\{y : R(x, y)\}$, consistindo em todos os objetos equivalente a x.

3.3 Autômatos

3.3.1 Autômatos Finitos

Por computador determinístico, queremos dizer que, ao ler uma entrada particular de instrução, a máquina passa de um determinado estado para algum outro estado particular (ou se mantém no mesmo estado) onde o estado resultante é completamente determinado pelo estado anterior e a entrada da instrução. Nenhum conhecimento é requerido do estado no que a máquina estava algumas instruções atrás. Algumas sucessões de entrada de instruções podem conduzir ao sucesso e outras não. O sucesso é completamente determinado pela seqüência de entrada.

Veremos agora um modelo chamado de autômatos finitos, finito porque os números de possíveis estados e número de símbolos no alfabeto são ambos finitos, e autômato porque a mudança de estados é totalmente governada pela entrada. É automático (involuntário e mecânico) não voluntarioso.

Um **autômato finito** é uma coleção de três objetos:

1. Um conjunto finito de estados um dos quais é designado como o estado inicial, e alguns (talvez nenhuma) são designados como estados finais ou de aceitação.

2. Um alfabeto $\sum$ de possíveis entradas de símbolos do qual são formados seqüências que serão lidos símbolo a símbolo.

3. Um conjunto finito de transições que para cada estado e para cada símbolo de entrada do alfabeto, indica qual será o próximo estado do autômato.

Esta definição está incompleta no sentido de que descreve o que é um automato finito mas não como funciona. Apresenta-se como sendo uma entrada seqüencial de símbolos o qual é lido a partir do símbolo mais a esquerda. Começando no estado inicial, os símbolos determinam uma sucessão de estados. A computação termina quando o último símbolo introduzido for lido. Se o último estado é final, então dizemos que o automata reconhece ou aceita a cadeia de símbolos, caso contrário, a entrada é rejeitada.

Alguns estudiosos preferem chamar os autômatos finitos por "aceitadores finitos" porque seu trabalho consiste exclusivamente em concordar com certas seqüências de entradas e executá-las. Não faz nada como produção de impressão. Mesmo assim, aderiremos à terminologia autômato finito.

Começaremos detalhando um exemplo particular

Suponha que o alfabeto tenha só duas letras a e b. Também assumiremos que só existem três estados: x, y e z. Sejam as seguintes regras de transição:

1. Do estado x e entrada a passa para o estado y.

2. Do estado x e entrada b passa para o estado z.
3. Do estado y e entrada a passa para o estado x.
4. Do estado y e entrada b passa para o estado z.
5. Do estado z e qualquer entrada permanece no estado z.

Designaremos x como o estado inicial e z como o único estado final.

Temos, agora, um autômato finito perfeitamente definido, desde que cumpre todas as três exigências acima: estados, alfabeto, transições.

Examinaremos o que acontece para várias seqüências de entradas. Começaremos com a seqüência aaa. Como sempre, começamos no estado x. O primeiro símbolo da seqüência é a, e nos diz que vamos para y (através de regra 1). A próxima entrada também é a, o autômata nos indica, através da Regra 3, que devemos voltar a x. A terceira entrada é outro a, e novamente, através de regra 1, vamos para y. Não há mais nenhuma entrada de símbolos na seqüência. Assim, nosso trabalho terminou. Não terminamos no estado final (estado z). Não obtivemos sucesso na terminação de execução.

A seqüência aaa não está na linguagem de todas as entradas que deixam este autômato finito no estado z. O conjunto de todos as seqüências que nos deixam em um estado final é chamado de linguagem aceita pelo autômato finito. A entrada da seqüência aaa não está na linguagem aceita por este Autômato Finito. Em outras palavras, podemos dizer que a seqüência aaa é rejeitada por este autômato finito porque não conduz a um estado final. Aceitar a linguagem L significa que L é o idioma do autômato finito. Seja linguagem L_1 é um certo autômato finito que aceita L_2 (todas as palavras em L_2 são aceitas e todas as possíveis entradas aceitas são palavras de L_2), então este autômato finito também tem que aceitar todas as palavras da linguagem L_1. Porém, não podemos dizer que L_1 é aceito por este autômato finito desde que isso signifique que todas as palavras aceitadas pelo autômato finito estejam em L_1.

Consideramos que um autômato finito é uma máquina, ou seja, entendemos que este autômato finito tem capacidades dinâmicas. Mover ou introduzir componentes do alfabeto. Algo que pode modificar um estado para outro estado de entrada, ler alfabetos e executar operações . Podemos imaginar que o estado que estamos em algum determinado momento é iluminado e os outros são escuros. Um autômato finito executa uma seqüência de entradas. Sendo assim o vimos como sendo uma máquina.

Um **autômato finito não determinístico** (AFN) é uma coleção de três objetos:

1. Um conjunto finito de estados, com um estado inicial (-) e alguns estados finais (+).

2. Um alfabeto, $\sum$, de possíveis entradas de símbolos.

3. Um conjunto finito de transições que indicam como proceder de cada estado a outros estados no decorrer do alfabeto, de alguma forma, marcado.

Esta teoria pode ser aplicada, hoje em dia, em Redes de Petri. Basicamente esta rede executa operações em um grafo que representa um Automato Finito. Neste caso, cada arco do grafo é quem habilita a execução de uma determinada instrução e ocasiona a mudança entre os nós do grafo. Esta mudança representa a permanência ou a alteração do estado.

3.4 Algoritmos e Procedimentos Efetivos

Quando estudamos aritmética aprendemos a somar e multiplicar dois números. Porém, na maioria das vezes, não fomos ensinados sobre qual o sentido que tem uma soma e um produto de dois números. Simplesmente, executamos determinados métodos ou regras para achar somas e produtos. Tais métodos ou regras são exemplos de algoritmos ou procedimentos efetivos. A implementação deles não requer inteligência acima do normal, apenas a que precisa para obedecer as instruções do professor.

Um *algoritmo* ou *procedimento efetivo* é um conjunto de regras mecânicas, um método automático, ou programa para executar algumas operações matemáticas.

Um **procedimento efetivo** é uma descrição finita e não ambígua de um conjunto finito de operações. As operações devem ser efetivas no sentido de que há um procedimento estritamente mecânico para executá-las. Também deve ser efetivo, o processo de decidir a ordem na qual executar as operações. Um procedimento efetivo pode executar uma tarefa, pode computar uma função ou pode executar uma sucessão de operações. Programas de computador são exemplos de procedimentos efetivos. Quantidade finita de descrição e efetividade de operação são as características mais importantes de procedimentos efetivos.

Um procedimento efetivo que especifica uma sucessão de operações é chamado **algoritmo**. Programas de computador que sempre páram para qualquer entrada, isto é, nunca entram em um ciclo infinito, são algoritmos. O instante em que o algoritmo pára não é, com antecedência, necessariamente calculável.

Alguns exemplos de problemas para os quais os algoritmos podem ser, facilmente, programados são:

1. Dado um determinado n, achar o n-ésimo $n$úmero primo;

2. Diferenciar um polinômio;

3. Achar o maior fator comum de dois números (o algoritmo de Euclides);

4. Dado dois numerais x, y decidir se x é um múltiplo de y.

A saída é produzida mecanicamente de modo que poderia ser pensado como o resultado de uma seqüência de operações realizadas por uma máquina de calcular, ou por um computador ou, mais genericamente, por uma "caixa preta". Assim, o algoritmo é um procedimento ou método que é levado em consideração pela caixa preta para obter a saída referente a uma entrada dada.

Quando um algoritmo ou procedimento efetivo é usado para calcular os valores de uma função numérica a função em questão é descrita através de frases como efetivamente calculável, ou algoritmicamente computável, ou efetivamente computável, ou simplesmente, computável. Por exemplo, o $MDC(x, y)$, o maior fator comum de x e y, e $f(n)$ o n-ésimo número primo, são computáveis neste sentido informal, como já indicamos.

Por outro lado, considere a seguinte função :

$$g(n) = \begin{cases} 1 & \text{, se a expansão decimal de } \pi \text{ tem } n \text{ setes consecutivos} \\ 0 & \text{, caso contrário} \end{cases}$$

A maioria dos matemáticos aceitaria que g é uma função perfeitamente legítima. Mas g é computável? Há um procedimento mecânico para gerar os dígitos sucessivos na expansão decimal de π.

De posse deste procedimento e dado um n, o procedimento começará a geração da expansão decimal de π, um dígito de cada vez, e verificará. Se em alguma fase de uma execução gerar exatamente n setes consecutivos, então o processo pára e $g(n) = 1$. Se nenhuma sucessão de n setes aparecer, então $g(n) = 0$.

O problema com este procedimento é que, se para um n particular não há nenhuma sucessão de n setes sucessivos, então não há nenhuma fase no processo onde podemos parar e concluir que este é o caso. Sabemos que em qualquer fase particular, tal sucessão de setes poderia aparecer na parte da expansão de π que não foi examinada. Assim o procedimento irá gerando dígitos para uma entrada n tal que $g(n) = 0$. Portanto, este procedimento não é efetivo.

Este exemplo contém duas características implícitas na idéia de um procedimento efetivo:

1. que um procedimento efetivo leva em consideração uma seqüência de fases ou passos (ambos concluídos em tempo finito)

2. que qualquer saída deve emergir depois de um número finito de passos.

Logo, descrevemos as idéias de algoritmo e procedimento efetivo informalmente, assim como, a noção associada de função computável. Estas idéias devem ser colocadas, precisamente, para que elas possam se tornar a base para uma teoria matemática de computabilidade.

Faremos estas definições em termos de um "computador idealizado" simples, que executa programas. Evidentemente, os procedimentos que podem ser levados em consideração por um computador real são exemplos de procedimentos efetivos.

Qualquer computador particular real, porém, está limitado ao tamanho dos números que podem receber como entrada, e na quantia de espaço de funcionamento disponível. Está nestes moldes o nosso computador que idealizará conforme a idéia informal de um algoritmo. Os programas para nossa máquina serão finitos, e necessitarão de uma computação completa que levará a um número finito de passos. Entradas e saídas serão restritas aos números naturais. Esta não é uma restrição significante, desde que tenhamos codificados outros tipos de objetos como números naturais.

3.5 A Máquina sem Limite de Registros (URM)

Nossa primeira idealização matemática de um computador é chamada **máquina sem limite de registros**, denotado por URM (oriundo do inglês Unlimited Record Machine); corresponde a uma ligeira variação de uma máquina concebida por Shepherdson & Sturgis [SS63]. Descrevemos aqui a URM e como funciona.

A URM tem um número infinito de registros denominados $R_1, R_2, R_3, \cdots$ cada um dos quais, em qualquer momento, contém um número natural. Denotamos o número contido em R_n por r_n. Isto pode ser representado como segue:

R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**•••**
r1	**r2**	**r3**	**r4**	**r5**	**r6**	**r7**	**•••**

Figura 3.1: Configuração abstrata de uma Máquina URM

Os conteúdos dos registros podem ser alterados pela URM em resposta a certas instruções que pode reconhecer. Estas instruções correspondem a operações muito simples, usadas para executar cálculos com números. Uma lista finita de instruções constitui um programa URM. As instruções são de quatro tipos, como segue:

1. Instrução Zero: Para cada $n = 1, 2, 3, \cdots$ existe uma instrução zero $Z(n)$. *A* resposta da URM para a instrução $Z(n)$ é alterar o conteúdo de R_n para 0 e deixar todos os outros registros inalterados.

   Suponha que a URM tem a seguinte configuração :

   e obedece a instrução $Z(3)$. Então a configuração resultante é:

   A resposta da URM para uma instrução $Z(n)$ é denotado por $0 \to R_n$, ou $r_n := 0$ (isto é lido como "r_n se torna 0").

R1	R2	R3	R4	R5	R6	R7	•••
9	6	5	23	7	0	0	•••

Figura 3.2: Exemplo de uma configuração de uma Máquina URM

R1	R2	R3	R4	R5	R6	R7	•••
9	6	0	23	7	0	0	•••

Figura 3.3: Configuração após a URM aplicar a instrução $Z(3)$

2. Instrução Sucessor: Para cada $n = 1, 2, 3, \cdots$ há uma instrução sucessor $S(n)$. A resposta da URM para a instrução $S(n)$ é aumentar o valor contido em R_n em 1 e deixar todos os outros registros inalterados.

   Exemplo: Suponha que a URM está sob a configuração da Figura 3.3 e vai executar a instrução $S(5)$. Então a nova configuração será:

R1	R2	R3	R4	R5	R6	R7	•••
9	6	0	23	8	0	0	•••

Figura 3.4: Configuração após a URM aplicar a instrução $S(5)$

   O efeito de uma instrução sucessor $S(n)$ é denotado por $R_n + 1 \to R_n$, ou $r_n := r_n + 1$ "r_n se torna $r_n + 1$"

3. Instrução Transferência: Para cada $m = 1, 2, 3, \cdots$ e $n = 1, 2, 3, \cdots$ há uma instrução transferência $T(m, n)$. A resposta da URM para a instrução $T(m, n)$ é substituir o conteúdo de R_n por r_m (o conteúdo de R_m), isto é, transfere r_m para R_n, e todos os outros registros (inclusive R_m) continuam inalterados.

   Exemplo: Suponha que a URM está com a configuração da figura 3.4 e obedece a instrução transferência $T(5, 1)$. Então a configuração resultante é

   A resposta da URM para uma instrução de transferência $T(m, n)$ é denotado através de $r_m \to R_n$ ou $r_n := r_m$, lida como "r_n se torna r_m"

4. Instruções de salto (jump): Na operação de um algoritmo informal pode haver uma fase onde são prescritos cursos alternativos de ação que dependem do progresso da operação até aquela fase. Em outras situações pode ser necessário

R1	R2	R3	R4	R5	R6	R7	•••
8	6	0	23	8	0	0	•••

Figura 3.5: Configuração após a URM executar a instrução $T(5,1)$

repetir uma determinada rotina várias vezes. A URM pode refletir tais procedimentos como estes usando instruções de salto (jumps). Estes permitirão saltos para trás ou adiante na lista de instruções. Por exemplo, poderemos usar uma instrução de salto para produzir a resposta seguinte:

Se $r_2 = r_6$, vá para a décima instrução do programa; caso contrário, vá para a próxima instrução no programa.

A instrução que extrai esta resposta será escrita $J(2,6,10)$.

Para cada $m = 1,2,3,\cdots$, $n = 1,2,3,\cdots$ e $q = 1,2,3,\cdots$ há uma instrução de salto $J(m,n,q)$. A resposta da URM para a instrução $J(m,n,q)$ é como segue. Suponha que esta instrução é encontrada em um programa P. Os conteúdos de R_m e R_n são comparados, mas todos os registros permanecem inalterado. Então

Se $r_m = r_n$, a URM procede à instrução da q-ésima instrução de P;

Se r_n é diferente de r_m, a URM procede à próxima instrução em P.

Zero, sucessor e transferência são chamadas de **instruções aritméticas**.

3.5.1 Computações

O ato de executar uma computação para a URM deve ser proporcionado por um programa P e uma configuração inicial, uma seqüência $a_1, a_2, a_3, \cdots$ de números naturais nos registros $R_1, R_2, R_3, \cdots$, respectivamente. Suponha que P consiste em s instruções $I_1, I_2, \cdots I_s$. A URM começa a computação obedecendo I_1, depois I_2, I_3 e assim por diante, a menos que uma instrução de salto, digamos $J(m,n,q)$, seja encontrada. E neste caso a URM procede à instrução prescrita por I_q caso os conteúdos atuais dos registros R_m e R_n sejam os mesmos. Ilustramos isto com um exemplo.

Exemplo 5 *Considere o programa seguinte:*

$I_1 : J(1,2,6)$
$I_2 : S(2)$
$I_3 : S(3)$

$I_4 : J(1, 2, 6)$
$I_5 : J(1, 1, 2)$
$I_6 : T(3, 1)$

Consideraremos a computação pela URM deste programa sob a seguinte configuração inicial:

R1	R2	R3	R4	R5	R6	R7	• • •
9	7	0	0	0	0	0	• • •

Figura 3.6: Configuração inicial para uma máquina URM do Exemplo 5

Não nos preocuparemos no momento com a função que este programa computa; desejamos ilustrar que o modo no qual a URM opera programas é puramente mecânico sem precisar entender o algoritmo que está sendo executado.

Podemos representar o progresso da computação escrevendo as configurações sucessivas que acontecem, junto com a próxima instrução a ser obedecida à conclusão de cada fase.

	R1	R2	R3	R4	R5	R6	R7	• • •
I1	9	7	0	0	0	0	0	• • •
I2	9	7	0	0	0	0	0	• • •
I3	9	8	0	0	0	0	0	• • •
I4	9	8	1	0	0	0	0	• • •
I5	9	8	1	0	0	0	0	• • •
I2	9	8	1	0	0	0	0	• • •

⋮

Figura 3.7: Execução do programa do Exemplo 5 - O processo continua infinitamente

Podemos descrever a operação da URM sob um programa $P = I_1, I_2, \cdots, I_s$ em geral como segue:

A URM começa obedecendo instrução I_1. Em qualquer fase futura na computação, suponha que o URM está obedecendo a instrução I_k. Então procede à próxima instrução na computação, definida a seguir:

Se I_k não é uma instrução de salto, a próxima instrução é $I_{(k+1)}$;

Se $I_k = J(m, n, q)$ a próxima instrução é I_q;

Caso $r_m = r_n$, senão a próxima instrução é $I_{(k+1)}$.

Onde r_m, r_n são os conteúdos atuais de R_m e R_n.

A URM procede assim sempre que possível; a computação pára quando, e só quando, não há nenhuma próxima instrução; isto é, se a URM obedeceu a instrução I_k e a próxima instrução na computação de acordo com a definição acima é I_v, onde $v > s$. Isto pode acontecer dos seguintes modos:

1. Se $k = s$ (a última instrução em P foi obedecida) e I_s é uma instrução aritmética;
2. Se $I_k = J(m, n, q)$, $r_m = r_n$ e $q > s$;
3. Se $I_k = J(m, n, q)$, $r_n \neq r_m$ e $k = s$.

Dizemos, então, que a computação pára depois da instrução I_k. A configuração final é a sucessão $r_1, r_2, r_3, \cdots$, os conteúdos dos registros nesta fase.

Existem computações que nunca param: por exemplo, a computação sob o programa $S(1), J(1, 1, 1)$ para qualquer configuração incial jamais parará. A computação sob este programa é representada pelo diagrama da Figura 3.8. A instrução de salto invariavelmente faz com que a URM siga a instrução determinada, ou voltar atrás, para a instrução $S(1)$.

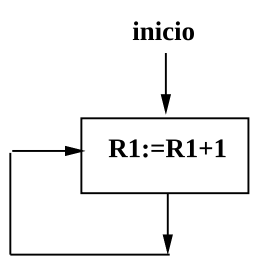

Figura 3.8: Fluxograma do programa URM $S(1), J(1, 1, 1)$

Existem modos mais sofisticados nos quais uma computação pode executar sempre, mas isto é causado essencialmente pela repetição ou retorno na execução do programa.

Adotaremos algumas convenções que nos serão úteis posteriormente.

Seja $a_1, a_2, a_3, \cdots$ uma sucessão infinita sobre $\mathbb{N}$ e P um programa URM, escreveremos:

1. $P(a_1, a_2, a_3, \cdots)$ para a computação sob P, com configuração inicial, $a_1, a_2, a_3, \cdots$;
2. $P(a_1, a_2, a_3, \cdots) \downarrow$ para significar que a computação $P(a_1, a_2, a_3, \cdots)$ pára;
3. $P(a_1, a_2, a_3, \cdots) \uparrow$ para significar que a computação $P(a_1, a_2, a_3, \cdots)$ nunca pára.

   Na maioria das configurações iniciais que nós consideraremos, todos serão 0 exceto os menores ou igual a a_n. Assim a seguinte notação é útil. Seja $a_1, a_2, a_3, \cdots, a_n$ uma sucessão finita de números naturais; escrevemos o $P(a_1, a_2, a_3, \cdots, a_n)$ para denotar a computação $P(a_1, a_2, a_3, \cdots a_n, 0, 0, 0, \cdots)$.

   Consequentemente

   (a) $P(a_1, a_2, a_3, \cdots a_n)\downarrow$ denota $P(a_1, a_2, a_3, \cdots a_n, 0, 0, 0, \cdots)\downarrow$;
   (b) $P(a_1, a_2, a_3, \cdots a_n)\uparrow$ denota $P(a_1, a_2, a_3, \cdots, a_n, 0, 0, 0, \cdots)\uparrow$.

Frequentemente, dizemos que quando uma computação pára, ela converge, e uma que nunca pára, diverge.

3.5.2 Funções URM-Computáveis

Suponha que f é uma função de $\mathbb{N}^n$ para $\mathbb{N}$ (n maior ou igual um); o que significa dizer que f é computável por uma URM ? É natural pensar em termos de computar um valor $f(a_1, ..., a_n)$ por meio de um programa P com configuração inicial $a_1, a_2, \cdots, a_n, 0, 0, \cdots$. Quer dizer, consideramos computações da forma $P(a_1, a_2, \cdots, a_n)$. Se tal computação pára, precisamos ter um único número que possamos considerar a saída ou resultado da computação; adotamos a convenção que este é r_1, o valor contido em R_1. Podem ser considerados os conteúdos finais dos outros registros como valores auxiliares ou intermediários e que portanto podem ser ignorados uma vez que temos o resultado desejado em R_1.

Desde que uma computação $P(a_1, \cdots, a_n)$ pode não parar, poderemos estabelecer nossa definição de computabilidade sobre funções f de $\mathbb{N}^n$ para $\mathbb{N}$, cujo domínio pode não ser todo $\mathbb{N}^n$; isto é, funções parciais. Requereremos que as computações pertinentes param (e dão o resultado correto) justamente para as entradas do domínio de f. Assim fazemos as seguintes definições :

Definição 16 *Seja f uma função parcial de $\mathbb{N}^n$ em $\mathbb{N}$.*

Suponha que P é um programa URM, e $a_1, a_2, \cdots, a_n, b \in \mathbb{N}$.

i. A computação $P(a_1, a_2, \cdots, a_n)$ **converge** *para* b *se* $P(a_1, a_2, \cdots, a_n)\downarrow$ *e na configuração final* b *está em* R_1*. Escreveremos isto por* $P(a_1, a_2, \cdots, a_n) \downarrow b$*;*

ii. P **URM-computa** f *se, para todo* $(a_1, \cdots, a_n) \in Dom(f)$ *e* $b \in \mathbb{N}$, $P(a_1, a_2, \cdots, a_n) \downarrow b$ *se e somente se* $f(a_1, \cdots, a_n) = b$*. Em particular, isto significa que:*

1. $P(a_1, a_2, \cdots, a_n)\downarrow$ *se e somente se* $(a_1, \cdots, a_n) \in Dom(f)$*.*
2. *a função* f *é* **URM-computável** *se existe um programa URM que seja URM-computa* f*.*

A classe de funções URM-computáveis é denotada por ρ, e o n-ésimo elemento das funções URM-computáveis por ρ_n.

Vejamos alguns exemplos de funções URM computáveis:

Exemplo 6 $f(x, y) = x + y$.

Obtemos $x + y$ *somando 1 a* x *(usando a instrução sucessor)* y *vezes. Um programa URM para computar* $x + y$ *tem que começar em* x*,* y*,* $0, 0, 0, \cdots$*. Nosso programa continuará somando 1 a* r_1 *e usará* R_3 *como um contador para manter o registro de quanto* r_1 *é aumentado. Uma configuração típica durante a computação é:*

R1	R2	R3	R4	R5	R6	R7	•••
x+k	y	k	0	0	0	0	•••

Figura 3.9: Configuração típica de uma URM durante a computação de uma soma

O programa será projetado para parar quando $k = y$*, deixando* $x+y$ *em* R_1 *como exigido.*

O procedimento que desejamos descrever em nosso programa é representado a seguir:

1. $J(3, 2, 5)$
2. $S(1)$
3. $S(3)$
4. $J(1, 1, 1)$

Note que a parada foi alcançada por uma instrução de salto a I_5 *que não existe. Assim,* $f(x, y) = x + y$ *é computável.*

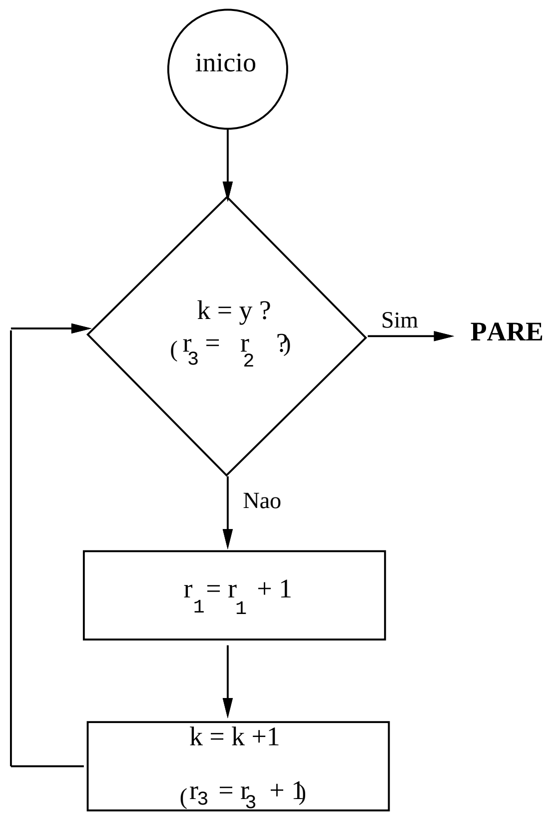

Figura 3.10: Fluxograma que representa a adição (Exemplo 6)

3.5.3 Predicados e Problemas de Decisão

Em matemática uma tarefa comum é decidir se números possuem uma determinada propriedade. Por exemplo, uma determinada tarefa é decidir se determinados números x, y possuem a seguinte propriedade: x é um múltiplo de y. Um algoritmo que computa operação seria um procedimento efetivo que introduz x, y e apresentando como saída Sim ou Não. Adotamos a seguinte convenção: 1 para Sim, e 0 para Não. Então a função que funciona para esta operação é a seguinte:

$$f(x,y) = \begin{cases} 1 & \text{, se } x \text{ é múltiplo de } y \\ 0 & \text{, se } x \text{ não é múltiplo de } y \end{cases}$$

Podemos dizer que a propriedade ou predicado x é um múltiplo de y é algoriticamente ou efetivamente decidível, ou simplesmente decidível se esta função f é computável (por alguns dos modelos aqui apresentados).

Suponha que $M(x_1, x_2, \cdots, x_n)$ é um predicado n-ário sobre números naturais. A função característica $\chi_M(x)$ (fixando $x = (x_1, \cdots, x_n)$) de M é definida por

$$\chi_M(x) = \begin{cases} 1 & \text{, se } M(x) \text{ é verdadeiro} \\ 0 & \text{, se } M(x) \text{ é falso} \end{cases}$$

Definição 17 *O predicado $M(x)$ é* **decidível** *se a função χ_M é computável; $M(x)$ é* **indecidível** *se $M(x)$ não é decidível.*

Exemplo 7 *O predicado $P(x)$ definido por "$x = 0$" é decidível. A função característica é dada por*

$$\chi_P(x) = \begin{cases} 1 & \text{, } se\ x = 0 \\ 0 & \text{, } se\ x \neq 0 \end{cases}$$

O programa seguinte computa χ_P:

1. $J(1,2,3)$
2. $J(1,1,4)$
3. $S(2)$
4. $T(2,1)$

Note que quando discutimos decidibilidade (ou indecidibilidade) sempre nos preocupamos com a computabilidade (ou não-computabilidade) de funções totais.

No contexto de decidabilidade, são descritos às vezes propriedades ou predicados como problemas. Assim, poderíamos dizer que o problema $x \neq y$ é decidível.

3.5.4 Computabilidade em Outros Domínios

Nossa definição de computabilidade e decidibilidade só se aplica a funções e predicados sobre números naturais. Estas noções são estendidas facilmente a outros tipos de objetos (números inteiros, polinômios, matrizes, etc.) por meio de codificação . Uma codificação de um domínio D de objetos é uma injeção $\alpha : D \to \mathbb{N}$. Dizemos que um objeto $d \in D$ é codificado pelo número natural $\alpha(d)$. Suponha, agora, que f é uma função de D para D; então f é naturalmente codificada pela função f^* de $\mathbb{N}$ para $\mathbb{N}$ que determine o código de um objeto $d \in Dom(f)$ para o código de $f(d)$ como ilustra o seguinte diagrama comutativo:

$$\begin{tikzcd} D \arrow[r, "f"] \arrow[d, "\alpha"'] & D \arrow[d, "\alpha"] \\ \mathbb{N} \arrow[r, "f^*"'] & \mathbb{N} \end{tikzcd}$$

Explicitamente temos

$$f^* = \alpha \circ f \circ \alpha^{-1}$$

onde α^{-1} é o inverso da função α.

Agora, podemos estender a definição de URM-computabilidade para D dizendo que f é computável se f^* é uma função de números naturais computável.

Exemplo 8 *Considere o domínio $\mathbb{Z}$. Uma codificação do conjunto dos números inteiros no conjuntos dos números naturais é determinada pela função $\alpha : D \to \mathbb{N}$ definida por:*

$$\alpha(n) = \begin{cases} 2n & , \text{ se } n \geq 0 \\ -2n-1 & , \text{ se } n < 0 \end{cases}$$

Então α^{-1} é determinado por

$$\alpha^{-1}(m) = \begin{cases} \frac{1}{2}m & , \text{ se } m \text{ é par} \\ -\frac{1}{2}(m+1) & , \text{ se } m \text{ é ímpar} \end{cases}$$

Considere a função $f : \mathbb{Z} \to \mathbb{Z}$ definida por $f(x) = x - 1$, então $f^* : \mathbb{N} \to \mathbb{N}$ é determinada por

$$f^*(x) = \begin{cases} 1 & , \text{ se } x = 0 \\ x+2 & , \text{ se } x \text{ é ímpar} \\ x-2 & , \text{ se } x \text{ é par e } x \neq 0 \end{cases}$$

É um exercício simples escrever um programa que computa f^*. Consequentemente $x+1$ é uma função computável em $\mathbb{Z}$.

As definições de função n-ária computável em um domínio D e de predicado decidível em D são obtidas extendendo da maneira acima.

3.6 Funções Parciais Recursivas

1. As funções básicas

   Primeiro notamos que alguma funções particularmente simples são computáveis; destas funções básicas (definida no lema abaixo) construiremos funções computáveis mais complicadas.

   Lema 1 *As funções básicas seguintes são computáveis:*

   i. a função zero $Z : \mathbb{N} \to \mathbb{N}$, $Z(x) = 0$;

ii. a função sucessor, definida por $S : \mathbb{N} \to \mathbb{N}$, $S(x) = x + 1$;

iii. para cada $n \geq 1$ e $1 \leq i \leq n$, a função de projeção $\{U * n\}_i : \mathbb{N}^n \to \mathbb{N}$, $\{U * n\}_i(x_1, x_2, \cdots, x_n) = x_i$.

Demonstração: Estas funções correspondem às instruções aritméticas da máquina URM. Especificamente, os programas são como segue:

i. $Z(x) = 0$, programa $Z(1)$;

ii. $S(x) = x + 1$, programa $S(1)$;

iii. $\{U * n\}_i = x_i$, programa $U(1)$ ■

2. Programas agregados

Precisamos, em determinados momentos, escrever programas que incorporam outros programas.

Um exemplo simples de construção de um programa é quando temos os programas P e Q, e desejamos escrever um programa para o procedimento composto: primeiro faça P, e então faça Q. Nosso intuito é simplesmente escrever as instruções em P seguido pelas instruções em Q. Mas há dois pontos técnicos a considerar.

Suponha que $P = I_1, I_2, \cdots, I_s$. Uma computação de P é completada quando a próxima instrução para a computação é I_v, para algum $v > s$; requeremos que a computação de nosso programa composto proceda à primeira instrução de Q. Isto acontecerá automaticamente se $v = s + 1$, mas não no caso contrário. Assim para construir programas compostos temos que limitar a atenção de nossos programas que invariavelmente param, para que a próxima instrução seja I_{s+1}.Tais programas são ditos estarem na **forma padrão**. Claramente, são só as instruções de salto que podem causar este problema (um programa parar em situação não-padrão). Assim temos a definição seguinte.

Definição 18 *Um programa $P = I_1, I_2, \cdots, I_s$ está na* **forma padrão** *se, para toda instrução de salto $J(m, n, q)$ em P, $q \leq s + 1$.*

Lema 2 *Para qualquer programa P existe um programa na forma padrão, P^*, tal que qualquer computação em P^* é idêntica à computação correspondente em P, exceto, possivelmente, da maneira da parada. Em particular, para qualquer $a_1, \cdots, a_n, b$,*
$P(a_1, \cdots, a_n) \downarrow b$ se e somente se $P^*(a_1, \cdots, a_n) \downarrow b$, e consequentemente $f_P^{(n)} = f_{P^*}^{(n)}$ para todo $n > 0$.

Demonstração: *Suponha que $P = I_1, I_2, \cdots, I_s$. Obtemos os P^* de P, simplesmente, mudando as instruções de salto de forma que todas as paradas de salto aconteçam para o salto I_{s+1}. Explicitamente, $P^* = I_1^*, I_2^*, \cdots, I_s^*$*
onde

i. se I_k *é uma instrução aritmética, então* $I_k^* = I_k$*;*

ii. se $I_k = J(m, n, q)$ *e* $q \leq s + 1$*, então* $I_k^* = I_k$*;*

iii. se $I_k = J(m, n, q)$ *e* $q > s + 1$ *então* $I_k^* = J(m, n, s + 1)$*.*

Claramente, P^* *assim construído é como requerido* ■

Assumiremos agora que os programas P e Q estão na forma padrão. O segundo problema, quando concatenamos P e Q, concerne às instruções de salto em Q. Um salto $J(m, n, q)$ acontecendo em comandos de Q um salto para a instrução q-ésima de Q (se $r_m = r_n$). Mas a q-ésima instrução de Q se tornará a $s + q$-ésima instrução no programa composto; assim cada salto $J(m, n, q)$ em Q deve ser modificado a se tornar $J(m, n, s + q)$ no programa composto para que o sentido seja preservado.

Agora sem qualquer preocupação adicional podemos definir a união ou concatenação de dois programas na forma padrão:

Definição 19 *Seja P e Q programas na forma padrão de s e t passos, respectivamente. A* **união** *ou* **concatenação** *de P e Q, escrito PQ, é o programa:*

$$I_1, I_2, \cdots, I_s, I_{s+1}, \cdots, I_{s+t}$$

onde $P = I_1, \cdots, I_s$*, e as instruções* $I_{s+1}, \cdots, I_{s+t}$ *são as instruções de Q com cada salto,* $J(m, n, q)$*, substituído por* $J(m, n, s + q)$*.*

Com esta definição está claro que o efeito de PQ é como o desejado: qualquer computação em PQ é idêntica à computação correspondente em P, seguido pela computação em Q, cuja configuração inicial é a configuração final da computação em P.

Existem duas considerações adicionais que precisamos fazer antes de proceder às tarefas principais deste assunto. Suponha que desejamos compor um programa Q que possui um determinado programa P como uma subrotina. Para escrever Q é importante observar se somos capazes de achar alguns registros que permanecem inalterados através de computações de P. Isto pode ser feito como segue.

Considerando que P é finito, existe um número menor u tal que nenhum dos registros R_v para $v > u$ é alterado por P; isto é, se $Z(n)$, ou $S(n)$, ou $T(m, n)$, ou $J(m, n, q)$ é uma instrução em P, então $m, n \leq u$. Claramente, durante qualquer computação de P, os conteúdos de R_v, para $v > u$, permanecem inalterados, e não tem nenhum efeito nos valores de $r_1, \cdots, r_u$. Assim, escrevendo para nosso programa novo, Q, os registros R_v, para $v > u$, podem ser usados, por exemplo, para armazenar informações sem afetar qualquer computação de P sobre uma seqüência de dados. Denotamos o número u por $\rho(P)$.

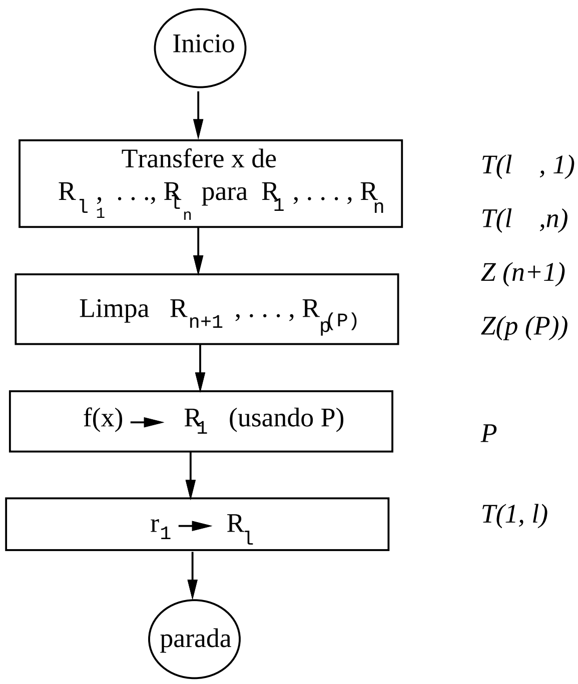

Figura 3.11: Diagrama de Fluxo para $P[l_1, \cdots, l_n \to l]$

Finalmente, introduziremos alguma notação que simplificará as principais provas desta seção. Suponha que P é um programa na forma padrão projetado para computar uma função $f(x_1, \cdots, x_n)$. Frequentemente, quando usamos P como uma subrotina em um programa maior, a entrada $x_1, \cdots, x_n$, para qual $f(x_1, \cdots, x_n)$ é designado pode estar contido no registro $R_{l_1}, \cdots, R_{l_n}$ em lugar de $R_1, \cdots, R_n$ como o programa P requer; mais adiante, a saída $f(x_1, \cdots, x_n)$ pode pretender ser requerido no futuro em algum registro R_l em lugar do convencional R_1; e finalmente o funcionamento registra $R_1, \cdots, R_{\rho_{(P)}}$ para P poder conter todos os tipos de informações não desejadas. Podemos modificar P levando em conta todos estes pontos como segue.

Escrevemos $P[l_1, \cdots, l_n \to l]$ para o programa representado pelo diagrama de fluxo da figura 3.11. O programa $P[l_1, \cdots, l_n \to l]$ tem o efeito de computação $f(r_{l_1}, \cdots, r_{l_n})$ e colocando o resultado em R_l. Além disso, os únicos registros afetados por este programa são (no máximo) $R_1, R_2, \cdots, R_{\rho_{(p)}}$ e R_l. (Assumiremos que $P[l_1, \cdots, l_n \to l]$ e $R_{l_1}, \cdots, R_{l_n}$ são distintos de $R_1, \cdots, R_n$; este será o nosso caso).

3.6.1 Substituição

Um modo comum de construir novas funções a partir de outras é substituir funções por outras funções, caso conhecido como composição de funções. No teorema seguinte mostramos que quando este processo é aplicado a funções computáveis, as funções resultantes também são computáveis. Em resumo, dizemos que ρ está fechado sob a operação de substituição.

Teorema 1 *Suponha que $f(y_1, \cdots, y_k)$ e $g_1(x), \cdots, g_k(x)$ são funções computáveis, onde $x = (x_1, \cdots, x_n)$. Então a função $h(x)$, definida por $h(x) = f(g_1(x), \cdots, g_k(x))$ é computável. (Nota. $h\ (x)$ é definida se e somente se $g_1(x), \cdots, g_k(x)$ são todas definidas e $(g_1(x), \cdots, g_k(x)) \in Dom(f)$; assim, se f e $g_1, \cdots, g_k$ são funções totais, então h é total.)*

DEMONSTRAÇÃO: *Suponha que $F, G_1, \cdots, G_k$ são programas na forma padrão que computam $f, g_1, \cdots, g_k$, respectivamente. Escreveremos um programa H que assume o procedimento natural seguinte para computar h. Dado x, usado pelos programas $G_1, \cdots, G_k$ para computar em sucessão $g_1(x), g_2(x), \cdots, g_k(x)$, fazendo um registro desses valores obtidos. Então usa o programa F para computar $f(g_1(x), \cdots, g_k(x))$.*

Temos que ter cuidado em não perder informações necessárias nos mais recentes estágios do procedimento, ou seja, x e $g_i(x)$ já obtidos.

$m = max(n, k, \rho(F), \rho(G_1), \cdots, \rho(G_k))$, começaremos armazenando x em $R_{m+1}, \cdots, R_{m+n}$; os registros $R_{m+n+1}, \cdots, R_{m+n+k}$ são usados para armazenar os valores $g_i(x)$, computados para $i = 1, 2, \cdots, k$. Estes registros são completamente ignorados pelas computações de $F, G_1, \cdots, G_k$. Uma configuração típica durante a computação de H será:

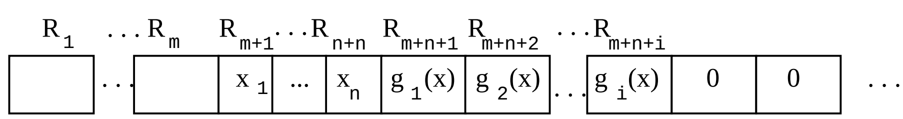

Figura 3.12: Máquina URM - Registros armazenados

Um diagrama de fluxo informal para computar h é ilustrado na figura 3.13. Isto é traduzido facilmente no programa H que computa h:

$$
\begin{array}{l}
T(1, m+1) \\
\vdots \qquad \vdots \\
T(n, m+n) \\
G_1[m+1, m+2, \cdots, m+n \to m+n+1] \\
G_k[m+1, m+2, \cdots, m+n \to m+n+k] \\
F[m+n+1, \cdots, m+n+k \to 1]
\end{array}
$$

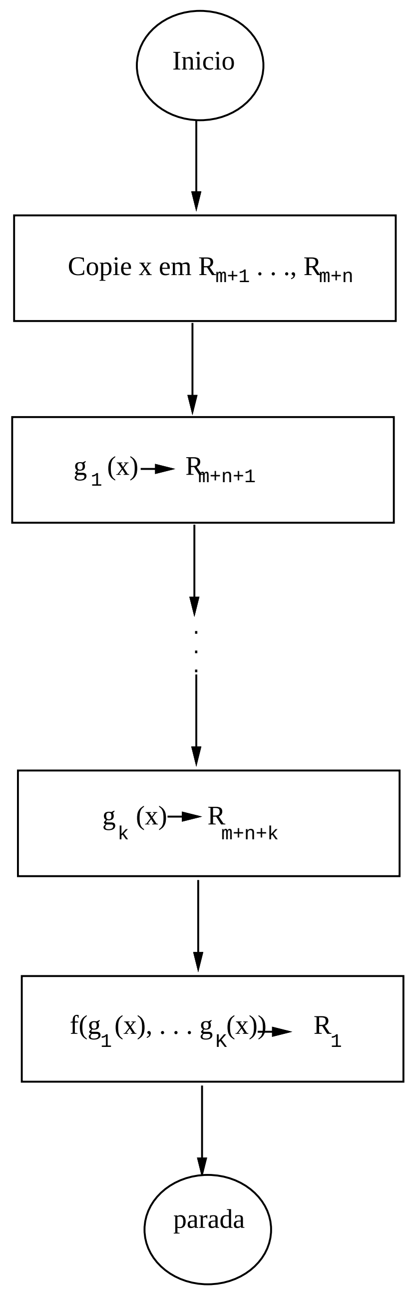

Figura 3.13: Diagrama da Substituição (Teorema 1)

Claramente, uma computação $H(x)$ parará se e somente se cada computação $G_i(x)$, parar $(1 \leq i \leq k\ )$ e a computação $F(g_1(x), \cdots, g_k(x))$ parar ■

Novas funções podem ser obtidas através de qualquer função , determinada reorganizando ou identificando suas variáveis, ou somando variáveis novas; por exemplo, de uma função $f(y_1, y_2)$ podemos obter

$$h_1(x_1, x_2) = f(x_2, x_1) \text{ (reestruturação)},$$
$$h_2(x) = f(x, x) \text{ (identificação)},$$
$$h_3(x_1, x_2, x_3) = f(x_2, x_3) \text{ (somando variáveis)}.$$

A seguinte aplicação do teorema anterior mostra que quaisquer destas operações (ou uma combinação delas) transforma funções computáveis em funções computáveis.

Teorema 2 *Suponha que $f(y_1, \cdots, y_k)$ é uma função computável e que $x_{i_1}, x_{i_2}, \cdots, x_{i_k}$ é uma seqüência de k variáveis obtidas de $x_1, \cdots, x_n$ (possivelmente com repetições). Então a função h dada por*

$$h(x_1, \cdots, x_n) = f(x_{i_1}, \cdots, x_{i_k})$$

é computável.

Demonstração: Escrevendo x= $(x_1, \cdots, x_n)$ temos

$$h(x) = f(U^n_{i_1}(x), U^n_{i_2}(x), \cdots, U^n_{i_k}(x))$$

que, pelo Lema 1 (i.i.i) e pelo teorema anterior, é computável. ■

Usando este resultado podemos também ver que o último teorema é verdadeiro quanto as funções $g_1, g_k, \cdots$ substituídas em f não necessariamente, são funções de todas as variáveis $x_1, x_n, \cdots$, como no exemplo seguinte.

Exemplo 9 *A função $f(x_1, x_2, x_3) = x_1 + x_2 + x_3$ é computável; isto pode ser deduzido do fato de que $x + y$ é computável, substituindo $x_1 + x_2$ para x , e x_3, para y em $x + y$.*

Substituição combinada com o princípio descrito na próxima seção fornece um método poderoso de gerar funções computáveis.

3.6.2 Recursão

Recursão é um método de definir uma função especificando cada um de seus valores em termos de valores previamente definidos, e usando outras funções já definidas anteriormente.

Vejamos uma forma de recursão a seguir chamada de **recursão primitiva**. Suponha que $f(s)$ e $g(x, y, z)$ são funções (não necessariamente totais ou computáveis). Considere a definição seguinte de **equações de recursão** como uma nova função $h(x, y)$:

i. $h(x, 0) = f(x)$,

ii. $h(x, y + 1) = g(x, y, h(x, y))$.

A primeira vista, isto pode parecer um pouco duvidoso como uma definição, pois na segunda linha aparece h que está sendo definido em termos de si mesmo - uma definição circular! Porém, com um pouco de esforço podemos nos convencer que esta é uma definição válida, por exemplo: achar o valor de $h(x, 3)$ segue os passos seguintes:

1. ache $h(x, 0)$ através do ítem (i) da definição anterior;
2. então, sabendo que $h(x, 0)$ use o ítem (ii) para obter $h(x, 1)$;
3. semelhantemente, obtenha $h(x, 2)$, e finalmente $h(x, 3)$ através de aplicações adicionais de (ii).

Assim, circulatoriamente é definido valores de $h(x, y)$, para $x \neq 0$, como sendo determinado sempre em termos de um valor já obtido.

Uma função h é definida através da recursão primitiva das funções f e g se as funções h, f e g satisfazem as equações de recursão. Se ambos f e g são totais, então h, como definida acima, é total. O domínio de h satisfará as condições

i. $(X, 0) \in Dom(h)$ se e somente se $x \in Dom(f)$

ii. $(X, y+1) \in Dom(h)$ se e somente se $(x, y) \in Dom(h) e (x, y, h(x, y)) \in Dom(g)$.

Resumimos a discussão acima em um teorema cujo prova encontra-se em [Cut97].

Teorema 3 *Seja* $x = (x_1, \cdots, x_n)$ *, e supondo que* $f(x)$ *e* $g(x, y, z)$ *são funções; segundo [Cut97], há uma única função* $h(x, y)$ *que satisfaz as equações de recursão:*

i. $h(x, 0) = f(x)$

ii. $h(x, y + 1) = g(x, y, h(x, y))$.

Nota: Quando $n = 0$ (os parâmetros x não aparecem) as equações de recursão levam a forma:

$h(0) = a$,

$h(y + 1) \simeq g(y, h(y))$,

onde $a \in \mathbb{N}$ ■

Exemplo 10 *Adição: para qualquer x, y temos*

$h(x, 0) = x + 0 = x$

$h(x, y + 1) = x + (y + 1) = (x + y) + 1$.

Assim a adição (a função $h(x, y) = x + y$) é definida através de recursão das funções $f(x) = x$ e $g(x, y, z) = z + 1$.

Exemplo 11 *y!: através da convenção 0!=1, temos*

0!=1, (y+1) !=y!(y+1).

Assim a função y! é definida através de recursão de 1 e a função $g(x, z) = z(y + 1)$.

Existem formas de definição por recursão que é mais geral que as discutidas até aqui, trabalharemos com essas formas mais adiante; agora conheceremos um tipo particularmente simples de definição chamado de recursão primitiva.

Teorema 4 *Suponha que $f(x)$ e $g(x, y, z)$ são funções computáveis, onde $x = (x_1, \cdots, x_n)$. Então a função $h(x, y)$, obtida de f e g através de recursão, é computável.*

Demonstração:

Sejam F e G programas, na forma padrão, que computam as funções $f(x)$ e $g(x, y, z)$. Contruiremos um programa URM, H, para a função $h(x, y)$ dada pela recursão das equações da definição por recursão. Dado a configuração inicial $x_1, \cdots, x_n, 0, 0, 0, \cdots, H$ computará primeiro $h(x, 0)$ (usando F); então, se $y \neq 0$, H usará G para computar sucessivamente $h(x, 1), h(x, 2), \cdots, h(x, y)$, e então parar.

Seja $m = max(n + 2, \rho(F), \rho(G))$. Começamos armazenando x, y em $R_{m+1}, \cdots, R_{m+n+1}$; os próximos dois registros serão usados para armazenar o valor atual dos números k e $h(x, k)$, para $k = 0, 1, 2, \cdots, y$. Escrevendo t para $m + n$, uma configuração típica durante o procedimento será assim

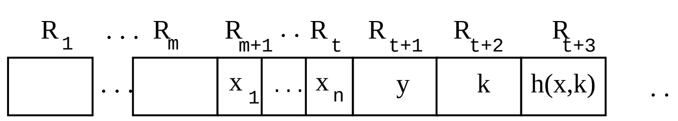

Figura 3.14: Registros Armazenados - R_{m+1} a R_{t+3}

com $k = 0$ inicialmente.

Um diagrama de fluxo informal para o procedimento é determinado na figura abaixo. Este diagrama de fluxo se traduz facilmente no seguinte programa, H, que computa h :

1. $T(1, m+1)$

$\vdots$

$n+1$. $T(n+1, m+n+1)$

$n+2$. $F[1, 2, \cdots, n \to t+3]$

$n+3$. $J(t+2, t+1, 8)$

$n+4$. $G[m+1, \cdots, m+n, t+2, t+3 \to t+3]$

$n+5$. $S(t+2)$

$n+6$. $J(1, 1, 4)$

$n+7$. $T(t+3, 1)$

Consequentemente h é computável $\blacksquare$

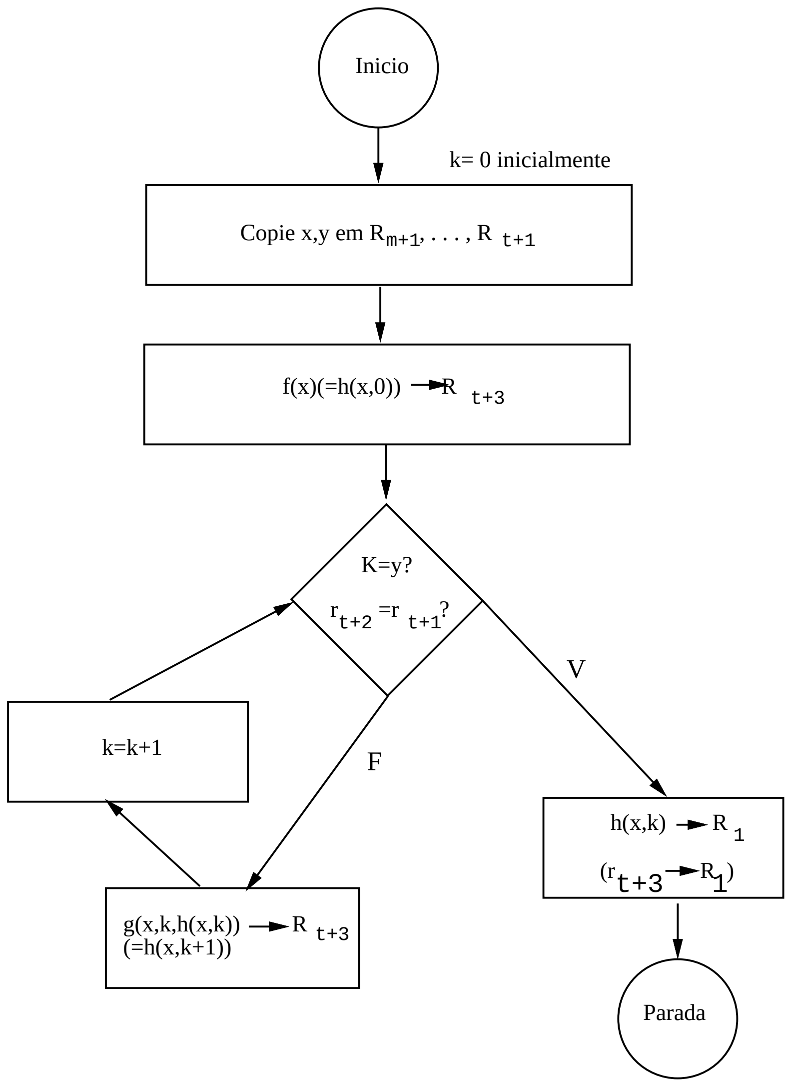

Figura 3.15: Recursão (Teorema 4)

Usaremos em usar os teoremas 3 e 4 para compilar uma coleção de funções computáveis. A coleção é potencialmente infinita. Assim, nossa escolha é influenciada por:

i. as necessidades de desenvolvimento subsequente de nossa teoria, e;

ii. a necessidade de provar que a tese de que todas as funções que consideramos computável na visão informal realmente são URM-computáveis. Por razões que ficarão aparentes depois que incluirmos algumas funções , como $x+y$ e $x \dot{-} 1$, para qual já escrevemos programas.

   Usaremos repetidamente o fato de que, através do teorema 2, uma definição através da recursão, as funções computáveis f e g não precisam ser funções com todas as variáveis nomeadas, para a função h ser computável.

Teorema 5 *As funções seguintes são computáveis.*

1. $x+y$
2. xy
3. x^y
4. $x \dot{-} 1$
5. $$x \dot{-} y = \begin{cases} x-y & ,\ se\ x \geq y \\ 0 & ,\ caso\ contrário \end{cases}$$
6. $$sg(x) = \begin{cases} 0 & ,\ se\ x = 0 \\ 1 & ,\ se\ x \neq 0 \end{cases}$$
7. $x!$
8. $min(x,y) = mínimo\ de\ x\ e\ y.$
9. $max(x,y) = máximo\ de\ x\ e\ y.$

DEMONSTRAÇÃO:

1. O exemplo 9 fornece uma definição por recursão da computabilidade das funções $Z(x)$

2. $x0 = 0$,

   $x(y+1) = xy + x$,

   é uma definição por recursão das funções computáveis $Z(x)$ e $z + x$

3. $x^0 = 1$,

   $x^{y+1} = x^y x$; por recursão em (2)

4. $0 \dot{-} 1 = 0$ $(x+1) \dot{-} 1 = x$; por recursão

5. $x \dot{-} 0 = x$,

   $x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1$; por recursão em (4)

6. $sg(0) = 0$,

   $sg(x+1) = 1$, por recursão

7. Exemplo 10 fornece uma definição por recursão de funções computáveis

8. $min(x,y) = x \dot{-} (x \dot{-} y)$;por substituição

9. $max(x,y) = x + (y \dot{-} x)$; por substituição ■

Vale salientar que as operações **min** e **max** são computáveis, conforme foi visto acima. Estas operações nos serão importantes no capítulo 6.

Os seguintes corolários são úteis e envolvem decidibilidade de predicados:

Corolário 1 *Suponha que $f_1(x), \cdots, f_k(x)$ são funções computáveis totais, e $M_1(x), \cdots, M_k(x)$ são predicados de decidibilidade, tal que para todo x exatamente um de $M_1(x)$,$\cdots$,$M_k(x)$ é verdadeiro.*

Então a função $g(x)$ é dada por

$$g(x) = \begin{cases} f_1(x), & \textit{se } M_1(x) \textit{ é verdadeiro,} \\ f_2(x), & \textit{se } M_2(x) \textit{ é verdadeiro,} \\ f_3(x), & \textit{se } M_3(x) \textit{ é verdadeiro,} \\ \vdots & \vdots \\ f_k(x), & \textit{se } M_k(x) \textit{ é verdadeiro,} \end{cases}$$

é computável.

Demonstração:

$g(x) = c_{M_1}(x)f_1(x) + \cdots + c_{M_k}(x)f_k(x)$, computável por substituição que usa adição e multiplicação ■

Corolário 2 *(Álgebra de decidibilidade)*

Suponha que $M(x)$ e $Q(x)$ são predicados decidíveis. Então também são decidíveis:

i. a negação de $M(x)$

ii. a conjunção de $M(x)$ e $Q(x)$

iii. a disjunção de $M(x)$ ou $Q(x)$

DEMONSTRAÇÃO:

Segundo [Cut97], as funções características destes predicados são as seguintes:

i. negação de $M(x)$: $1 \dot{-} \chi(x)$,

ii. $M(x)$ e $Q(x) : \chi(x)\ c_Q(x)$,

iii. $M(x)$ ou $Q(x)$: $max(\chi(x), c_Q(x))$

Cada uma das funções da direita são computáveis uma vez que c_M e c_Q são, por substituição em funções do teorema 5 ■

3.6.3 Minimalização

Na seção anterior vimos que um grande número de funções podem ser computáveis usando as operações de substituição e recursão, e operações derivadas destas. Há uma operação importante que gera funções computáveis chamada de **minimalização ilimitada**, ou **simplesmente minimalização** , o qual descreveremos agora.

Suponha que $f(x, y)$ é uma função (não necessariamente total) e desejamos definir uma função $g(x)$ por:

$$g(x) = \text{“O menor } y \text{ tal que } f(x, y) = 0\text{”},$$

De tal modo que se f é computável, então g também é. Dois problemas podem surgir.

1. Primeiro, para algum x pode não haver nenhum y tal que $f(x, y) = 0$.

2. Segundo, assumindo que f é computável, considere o algoritmo natural seguinte para computar $g(x)$. Compute $f(x, 0), f(x, 1), \cdots$ até y ser achado tal que $f(x, y) = 0$. Este procedimento pode não terminar se f não for total, até mesmo se y existir; por exemplo, se $f(x, 0)$ é indefinido, mas $f(x, 1) = 0$.

Assim somos conduzidos à definição seguinte de **minimalização** o operador μ que gera funções computáveis de funções computáveis.

Definição 20 *Seja $f : \mathbb{N}^2 \to \mathbb{N}$ uma função, definimos a* **minimalização** *de f como sendo a função de $\mathbb{N} \to \mathbb{N}$ a seguir:*

$$\mu(y)(f(x,y)=0) = \begin{cases} \text{o menor } y \text{ tal que:} \\ \ \text{i. } f(x,z) \text{ é definida, para todo } z \leq y, \text{ e} \\ \ \text{ii. } f(x,y)=0, \text{ se existe } y, \\ \text{indefinido, se } y \text{ não existe} \end{cases}$$

O operador $\mu(y)(\cdots)$ pode ser chamado, simplesmente, de $\mu - operador$.

O próximo teorema mostrará que ρ é fechado sob a minimalização.

Teorema 6 *Seja $f(x,y)$ uma função computável; então a função $g(x) = \mu(y)(f(x,y) = 0)$ também é computável.*

Demonstração:

Suponha que $x = (x_1, \cdots, x_n)$ e F é um programa na forma padrão que computa a função $f(x,y)$. Seja $m = max(n+1, \rho(F))$. Escrevemos um programa G que descreve o algoritmo natural para g: para $k = 0, 1, 2, 3, \cdots$, computa $f(x,k)$ até um valor de k ser achado tal que $f(x,k) = 0$; este valor de k é a saída exigida.

O valor de x e o valor corrente de k serão armazenado em registros $R_{m+1}, \cdots, R_{m+n+1}$ antes de computar $f(x,k)$: assim a configuração típica será:

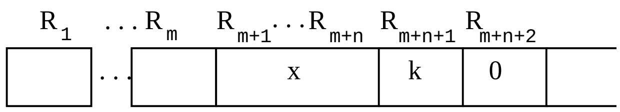

Figura 3.16: Registros Armazenados - (R_{m+1} a R_{m+n+2})

com $k = 0$*, inicialmente. Note que* r_{m+n+2} *sempre é 0.*

Um diagrama de fluxo que representa este procedimento para g é determinado na figura abaixo:

1. $T(1, m+1)$

$\vdots$

n. $T(n, m+n)$

p. $F[m+1, m+2, \cdots, m+n+1 \rightarrow 1]$

$n+2$. $J(1, m+n+2, q)$

$n+3$. $S(m+n+1)$

$n+4$. $J(1, 1, p)$

q. $T(m+n+1, 1)$

(I_p *é a primeira instrução da subrotina* $F[m+1, m+2, \cdots, \rightarrow 1]$*)* ■

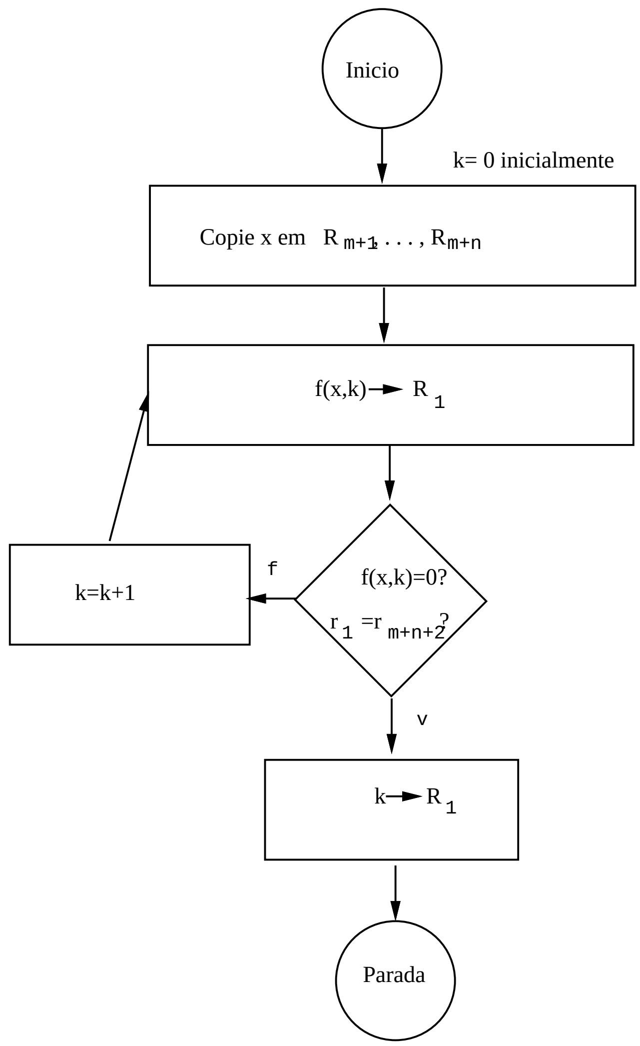

Figura 3.17: Minimalização (Teorema 6)

Definição 21 *Seja* $f : \mathbb{N}^r \mapsto \mathbb{N}^s$, f *é uma* **função parcial recursiva** *em* $\mathbb{N}$ *se é uma das funções zero, sucessor ou de projeção em* $\mathbb{N}$ *ou puder ser obtida através de um número finito de operações de substituição, recursão e minimalização sobre essas funções.*

Em outras palavras, é a menor classe de funções parciais sobre os números naturais contendo as funções básicas e fechada sobre as operações de substituição, recursão e minimalização.

3.7 Outras Definições Sobre Computabilidade

3.7.1 Máquinas de Acesso de Randômico (MAR)

Todos os computadores contemporâneos permitem o "ter acesso randômico" da memória. Um endereço é enviado à memória que devolve os dados armazenados em determinado endereço. O nome "Máquina de Acesso Randômico", cuja abreviatura MAR é originada do inglês "Random Acess Machine (RAM)", vem do fato de que os modelos mais antigos não eram nenhum de acesso randômico; eles estavam baseado em fitas sequentes ou eram funcionais em sua natureza. Como um ponto de partida de nossa investigação, consideraremos um modelo que fortemente se assemelha à linguagem de programação *assembler* em um computador convencional. O modelo que introduzimos executará operações muito simples em registros. Todo computador real tem uma capacidade fixa de memória. Esta memória pode ser estendida arbitrariamente, a grande perda de eficiência, somando uma fita ou uma unidade de disco. Então a última capacidade da máquina só está limitada pela habilidade de se fabricar ou do poder de compra de fitas ou discos (quando falamos em poder de compra estamos nos referindo a viabilidade de existência de um determinado produto em relação ao seu custo). Não querendo considerar apenas assuntos de eficiência contudo, nossa Máquina de Acesso Randômico (MAR) terá um conjunto potencialmente infinito de registros, $R_1, R_2, \cdots$, cada um com as propriedades dos números naturais. A advertência que fizemos há pouco sobre armazenamento principal e armazenamento periférico (e a administração deles) como produtos de como nós (necessariamente) executaremos computações. Terá tudo a ver com dados, e computação sobre aqueles dados.

Programas MAR serão abstrações de programas na linguagem *assembler* no sentido de que eles usam um número limitado e fixo de tipos de instruções e a única estrutura de controle permitida é a instrução "Branch". Não há nada especial sobre nossa escolha de instruções. Entre as possíveis escolhas para conjuntos de instrução, o escolhido aqui está baseado em alguma noção de simplicidade não-técnica. Programas MAR são sucessões finitas de instruções muito básicas. Consequentemente, cada programa MAR se refere a um número finito de registros. Embora a capacidade de memória de uma MAR seja ilimitada, qualquer computação descrita por um programa de uma MAR terá acesso finito de muitos dados, a menos que, claro, a computação nunca termine. No caso da computação não terminar, a quantidade de dados acessados em qualquer determinado momento é finita. Cada instrução pode ter um rótulo onde nomes de rótulos são escolhidos da lista: $N_0, N_1, \cdots$. Os sete tipos de instruções são descritas abaixo:

1. INC R_i Incremento (por 1) o conteúdo do registro R_i.

2. DEC R_i Decremento (por 1) o conteúdo do registro R_i. Se R_i contém 0 antes desta instrução ser executada, o conteúdo de R_i permanece inalterado.

3. CLR R_i Coloca 0 no registro R_i.

4. $R_i \leftarrow R_j$ Substitui o conteúdos do registro R_i com o conteúdo do registro R_j. O conteúdo de R_j permanece inalterado.

5. JMP $N_i x$ Se $x = a$, então a próxima instrução a executar é a instrução que antecede a rótulo N_i. Se $x = b$ então a próxima instrução a executar é a instrução seguinte ao rótulo N_i. Será representado o a para "sobre" e o b para "sob". Esta convenção incomum permite agregar programas que precisam ter atenção de realocar os rótulos de instrução.

6. R_j JMP $N_i x$ Executa uma instrução de JMP, como acima, se o registro R_j contém 0.

7. STOP não faz nada.

Definição 22 Um programa MAR *é uma sucessão finita de instruções tal que cada instrução de JMP (condicional ou não) tem um destino válido, isto é, o endereço da instrução seguinte contido no rótulo da instrução existe, e a última instrução é STOP.*

Definição 23 *Um programa MAR* **pára** *quando alcança a instrução final STOP.*

Definição 24 *Um programa MAR* **P computa** *uma função parcial ϕ, de n argumentos, se e somente se P, quando iniciado em $x_1, \cdots, x_n$, nos registros $R_1, \cdots, R_n$, respectivamente, e todos os outros registros usados por P contém $0, P$ só pára se $\phi(x_1, \cdots, x_n)$ for definido e R_1 contiver o valor $\phi(x_1, \cdots, x_n)$. Uma função parcial é* **MAR-computável** *se algum programa MAR o computa.*

Há uma diferença sutil entre afirmar a existência de um programa MAR que computa alguma função e a possibilidade de fato de se produzir o programa. Por exemplo, considere o problema de tentar decidir quantas vezes aparece a palavra "recurso" como um substring em uma cadeia infinita de símbolos de um alfabeto $\{a, \cdots, z\}$. Não importa quanto da cadeia examinamos, nunca saberemos se vimos todas as ocorrências. Porém, existe um programa MAR que nos contará exatamente quantas ocorrências há. Devido à possibilidade de que existem infinitas instâncias da palavra " recursão " embutidas na cadeia, usaremos a convenção de que um programa MAR pode sinalizar que há exatamente n repetições da palavra. A saída 0 será reservado para indicar a situação onde há infinitas ocorrências do substring que estamos tentando contar. Como para qualquer número natural n, há um programa MAR que computa a função constante n. Um destes programas nos revelará exatamente quantas ocorrências do palavra " recursão" estão na cadeia.

A maioria das linguagens assembler têm instruções aritméticas mais poderosas. Como um primeiro exemplo, o programa seguinte computa a soma de dois argumentos.

N_1.JMP N_2b
.INC R_1
$.DECR_2$
.JMP N_1a
N_2 STOP

Seja N_c um rótulo não usado por P. Escolha n grande bastante tal que nenhum registro R_m sirva de referencial de referência por P, para qualquer $m \geq n$. Isto garantirá que aquele registro R_n conterá um zero inicialmente. Finalmente, forma P^* de P substituindo cada $N_k CLR\ R_i$ instrução com o segmento de código seguinte:

R_i.JMP N_cb
.DEC R_i
R_n.JMP N_ka
N_c.STOP

3.7.2 Máquinas de Turing

A definição de computabilidade proposta por A. M. Turing [1936] está baseada em uma análise da implementação de um algoritmo baseado em um agente humano que usa caneta e papel. Turing viu isto como uma sucessão de atos muito simples dos tipos seguintes

1. escrevendo ou apagando um único símbolo
2. Transferindo atenção de uma parte do papel para outro.

Em cada fase o algoritmo especifica a ação a ser executada. Isto só depende de:

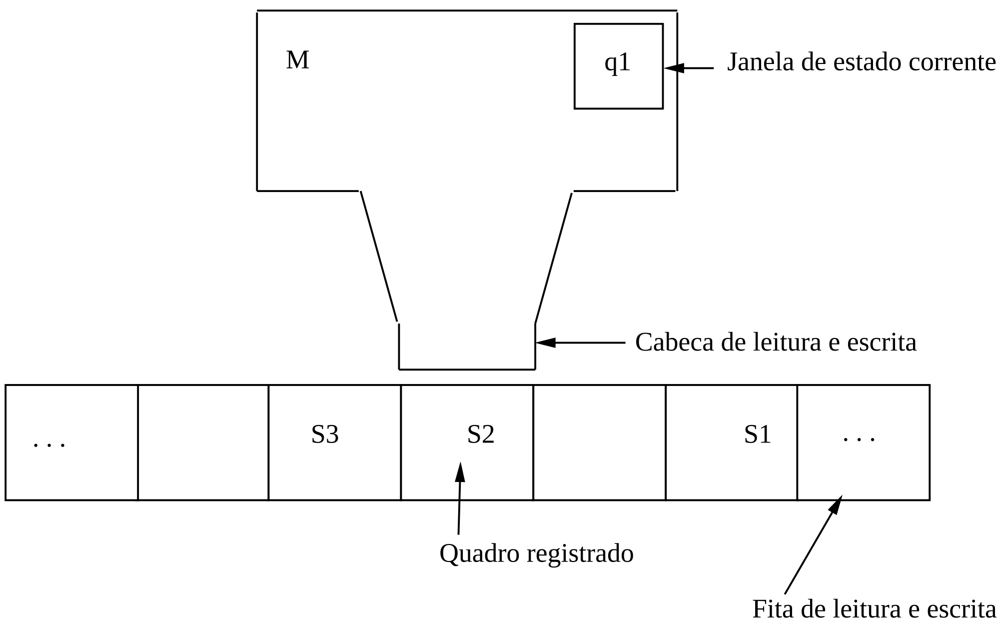

Figura 3.18: Representação Esquemática da Máquina de Turing

1. o símbolo escrito sob o papel atualmente pelo agente, e
2. O estado atual (de mente) do agente que pode mudar o resultado da ação nesta fase.

Turing inventou máquinas finitas que descrevem algoritmos concebidas deste modo. Há uma máquina diferente para cada algoritmo. Descreveremos estas como foi conhecida: **máquina de Turing**.

Uma **máquina de Turing** M é um dispositivo finito que executa operações em uma fita de papel. Esta fita é infinita em ambas as direções, e é dividida em quadros ao longo dela. A fita representa o papel usado por um agente humano que implementa um algoritmo; cada quadro representa uma porção do papel capaz de ser visto em um determinado momento. Em qualquer término de computação em M será usada só uma parte finita da fita, embora não podemos saber com antecedência de quanto será preciso. A fita não é realmente infinita.

Em qualquer momento cada quadro da fita está em branco ou contém um único símbolo de uma lista finita fixa de símbolos $S_1, S_2, \cdots, S_n$, o alfabeto de M. Seja B um espaço em branco, e conta isto como o símbolo S_0 pertencendo ao alfabeto de M.

M tem uma cabeça de leitura que em qualquer momento copia ou lê um quadro da fita. Podemos visualizar isto como mostrado na figura 3.18:

M é capaz de executar três tipos de operações simples na fita, isto é:

1. apagar o símbolo no quadrado que está sendo lido e substituí-lo por outro símbolo do alfabeto de M;

2. mover a cabeça de leitura um quadro à direita de sua posição fazendo a sua leitura.

3. mover a cabeça de leitura um quadro à esquerda de sua posição fazendo a sua leitura.

Em um determinado momento M está em um estado interno de número finito de estados, representado pelos símbolos $q_1, \cdots, q_m$. Durante a operação o estado de M pode mudar. Podemos imaginar o símbolo para o estado atual como sendo exibido em uma janela no exterior de M, isto representa uma espécie de guia parcial que mostra o que e quando determinada operação aconteceu e o que acontecerá no futuro.

A ação que a máquina M deverá executar depende do estado atual dela e do símbolo que é lido a cada momento. Esta dependência é descrita em $M's$ especificações que consistem em um conjunto finito Q de quádruplas, cada uma das quais levam uma das formas seguintes:

i. $q_i s_j s_k q_l$

ii. $q_i s_j R q_l$

iii. $q_i s_j L q_l$

onde, $1 \leq i, l \leq m$ e $0 \leq j, k \leq n$

A quadrupla $q_i s_j \alpha q_l$ em Q especifica a ação a ser executada por M quando está no estado q_i e lendo o símbolo s_j, como segue:

1. Opera na fita assim:

    i. se $\alpha = s_k$, apague s_je escreva s_k sob quadrado apagado;

    ii. se $\alpha = R$, mova a cabeça de leitura um quadro à direita;

    iii. se $\alpha = L$, mova a cabeça de leitura um quadro à esquerda;

2. Mudança no quadro q_l.

A especificação Q é a seguinte: para todo par $q_i s_j$ existe uma única quádrupla da forma $q_1 s_j \alpha \beta$; caso contrário poderia haver ambiguidade sobre M (não determinismo).

Para começar uma computação, M deve ser provida com uma fita e deve ser posicionado de forma que um quadro específico esteja sendo copiado; mais adiante, M deve ser fixado em algum estado inicial prescrito. Então, se M está no estado q_i e escreve o símbolo s_j, executa conforme o descrito acima contanto que existe uma quádrupla da forma $q_i s_j \alpha q_l$ em Q. Este tipo de ação é repetida então para o estado novo e para o símbolo copiado, e assim por diante. M continua operando enquanto for possível. A operação de M só termina quando está em um quadro q_i copiando um símbolo s_j tal que não existe nenhum quádruplo da forma $q_i s_j \alpha \beta$ em Q; isto é, não há nenhum quádruplo em Q isso especifica o que fazer. (É possível que isto nunca aconteça.)

Exemplo 12 *Seja M uma máquina de Turing cujo alfabeto consiste nos símbolos 0,1 (e um espaço em branco) e os possíveis estados q_1e q_2. A especificação de M é:*

1. $q_1 0 R q_1$

2. $q_1 10 R q_2$

3. $q_2 0 R q_2$

4. $q_2 1 R q_1$

Suponha que M está provida com a fita:

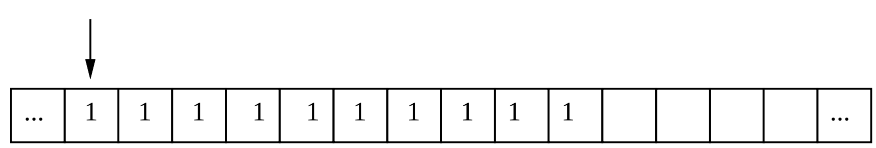

Figura 3.19: Máquina de Turing em um determinado instante

lê o qradro marcado pela seta ↓, que inicialmente está no estado q_1. Facilmente verifica-se que a ação de M_s é trabalhar da esquerda para direita ao longo da fita substituindo 1s pelo símbolo 0; M pára quando lê o primeiro quadro em branco desde que não exista nenhum quádruplo que especifica o que deveria fazer. A fita resultante é:

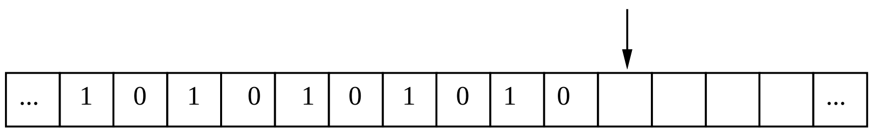

Figura 3.20: Máquina de Turing após algumas operações

lendo o quadro marcado pela seta ↓, M está no estado q_1.

Por outro lado, se M é provido com uma fita tal que todo quadro contenha o símbolo 0 ou 1, então, a operação de M nunca pára.

Está claro através deste exemplo que uma máquina de Turing M é um dispositivo que serve para efetuar um algoritmo que opera em fitas. São contidos detalhes completos do algoritmo na especificação Q de M. Assim, para um matemático, uma máquina de Turing é definida para ser o conjunto de quádruplo que especificam isto. Habituamente não são construídas máquinas de Turing físicas, com exceção de propósitos ilustrativos.

3.7.3 Funções Turing-Computáveis

Para considerar uma máquina de Turing M computando uma função numérica, temos que usar alguma convenção para representar números em uma fita. Usaremos o 1 como um "verificador de símbolo", e então representaremos um número x na fita como segue (ignore agora o marcador ↓):

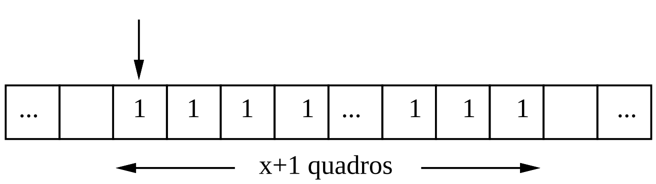

Figura 3.21: Máquina de Turing - representação

Usaremos $x + 1$ símbolos para representar x, para distinguir 0 de uma fita em branco.

A **função parcial** $f(x)$ computada por M é definida a seguir. Considere a computação de M sob a fita, começando no estado q_1, e lendo o quadro marcado inicialmente pela seta $\downarrow$. Então:

$$f(x) = \begin{cases} y \text{ onde } y, \text{ é o total de ocorrências do símbolo 1} \\ \quad \text{na fita ao final da computação, se esta pára} \\ \uparrow \text{, caso contrário.} \end{cases}$$

Semelhantemente, a função parcial n-ária $f(x_1, \cdots, x_n)$ computada por M, é definida contando o número de 1's na fita ao final da computação M quando iniciada no estado q_1 e, quando estiver lendo o quadro marcado por $\downarrow$ na fita seguinte pára e indefindo ($\uparrow$), caso contrário:

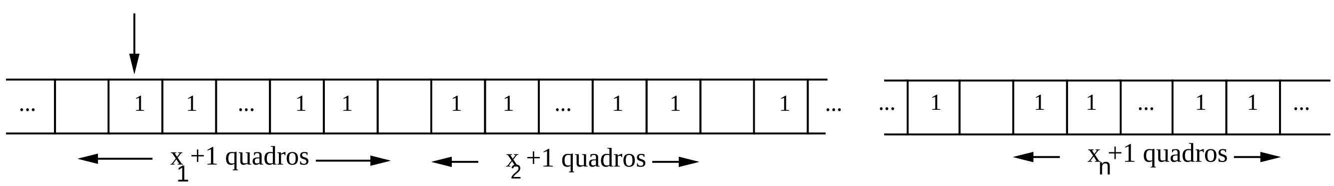

Figura 3.22: Configuração Inicial da Máquina de Turing M

Definição 25 *Uma função parcial é* **Turing-computável** *se existir uma máquina de Turing que a computa. A classe de todas as funções Turing-computáveis é representada por* $\vartheta\varrho$.

Exemplo 13 *A função* $x + y$ *é Turing-computável; a máquina de Turing dada pela especificação seguinte Turing-computa esta função.*

1. $q_1 1 B q_2$
2. $q_2 1 B q_3$
3. $q_3 1 R q_3$
4. $q_3 B 1 q_4$

A representação da fita de (x, y) *contém* $x + y + 2$ *ocorrências do símbolo 1, assim a Máquina* M *é projetada para apagar os dois primeiros destas ocorrências que começam da esquerda e substitui o branco que separa* x *de* y *por 1. Assim a configuração final da fita será composta por* $x + y + 1$ $1's$.

3.8 A Tese de Church

Voltemos nossa atenção agora para a seguinte pergunta: como pode ser a idéia informal e intuitiva de função efetivamente computável capturada pelas várias caracterizações formais?

Na luz das investigações, Church e Turing avançaram na idéia que a classe de funções que eles tinham definido coincide com a classe informalmente definida de funções efetivamente computáveis. Devido ao Resultado Fundamental, estas idéias são todas matematicamente equivalentes. O nome *a tese de Church* (às vezes *tese Church-Turing*) é agora usada para descrever quaisquer destas outras idéias. Assim, em termos da aproximação de URM, podemos declarar:

A Tese de Church
Intuitivamente e informalmente foi definida que as classe de funções parciais efetivamente computáveis coincide exatamente com a classe # das funções URM-computáveis.

Note imediatamente que esta tese não é um teorema que seja suscetível a prova matemática; tem o estado de uma **idéia** ou de **convicção** que deve ser substanciado através de evidência. A evidência para a tese de Church sob a qual resumimos, é impressionante.

1. O resultado fundamental: muitas propostas independentes para uma formulação precisa da idéia intuitiva conduziram à mesma classe de funções que chamamos #.

2. Uma coleção vasta de funções efetivamente computáveis foi apresentada para pertencer explicitamente a #; as funções particulares deste capítulo constitui o começo de tal uma coleção, podem ser aumentados *ad infinitum* pelas técnicas deste capítulo, e outros métodos mais sofisticados.

3. A implementação de um programa P na URM para computar uma função é claramente um exemplo de um algoritmo; assim, diretamente da definição da classe #, vemos que todas as funções em # são computáveis no sentido informal. Semelhantemente, com todas as outras classes equivalentes, as demais definições é como demonstrar que as funções envolvidas são efetivamente computáveis.

4. Ninguém nunca achou uma função que seria aceita como computável no sentido informal tal que não pertença a #.

Tomando como base esta evidência, e da própria experiência deles, a maioria dos matemáticos são conduzidos a aceitar a tese de Church. De nossa parte, propomos aceitar e usar a tese de Church onde for nescessário neste trabalho, que de certo modo exclarecemos agora.

Suponha que temos um algoritmo informalmente descrito para computar os valores de uma função f. Tal algoritmo pode ser descrito em português, ou por meio de diagramas, ou em condições matemáticas semi-formais, ou por qualquer outro meios que comunica como calcular efetivamente os valores de f, onde foi definido, em uma quantia finita de tempo. Em tal situação podemos desejar provar que f é URM-computável. Há, amplamente, dois métodos abertos.

Método 1 *Escreva um programa que URM-computa f (e prova), ou prova através de meios indiretos que tais programas existem. Isto poderia ser feito, por exemplo, pelos métodos deste capítulo, ou mostrando que f pertence a uma das muitas classes mostradas pelo resultado Fundamental para ser equivalente a $\#$. A prova formal que f é URM-computável pode ser um processo longo e bastante técnico. Essencialmente envolveria tradução do algoritmo informalmente descrito em um programa ou no idioma de uma do outra caracterização formal. Provavelmente haveria vários diagramas de fluxo como traduções de intermédio.*

Método 2 *Apresente uma prova informal (entretanto rigorosa) que o determinado algoritmo informal realmente é um algoritmo que serve para computar f. Então use a tese de Church e conclua que f é URM-computável.*

Em nossa dissertação aceitaremos o método 2 como sendo um método válido de provar, chamaremos de prova pela *tese de Church.*

Exemplo 14 *Seja P um programa URM; definiremos a função f por:*

$$f(x,y,t) = \begin{cases} 1 & , \textit{ se } P(x) \downarrow y \textit{ depois de } t \textit{ ou em menos passos da computação } P(x) \\ 0 & , \textit{ caso contrário} \end{cases}$$

Um algoritmo informal para computar f é como segue.

Dado (x,y,t), simule a computação $P(x)$, levando em consideração t passos de $P(x)$ a menos que esta computação pare a menos que t passos. Se $f(x,y,t) = 1$. Caso contrário (exemplo, se $P(x)$ pára em t ou menos passos com algum número diferente de y em R_1, ou se $P(x)$ não parou depois de t passos) temos $f(x,y,t) = 0$.

Simulação de $P(x)$ pára no máximo t passos são claramente procedimentos mecânicos que pode ser completado em uma quantia finita de tempo. Assim, f é efetivamente computável. Consequentemente, pela tese de Church, f é URM-computável.

Exemplo 15 *Seja f e g funções efetivamente computáveis. Definiremos a função h por:*

$$h(x) = \begin{cases} 1 & , \textit{se } x \in Dom(g) \\ \uparrow & , \textit{caso contrário} \end{cases}$$

Um algoritmo para h pode ser descrito em termos de determinados algoritmos para a função efetivamente computável f e g como segue:

Determinado x, comece os algoritmos para computar $f(x)$ e $g(x)$ simultaneamente. Se uma destas computações termina, então pára completamente, e fixa $h(x) = 1$. Caso contrário, continue indefinidamente.

Este algoritmo resulta $h(x) = 1$ para qualquer x tal que $f(x)$ ou $g(x)$ é definida; e continua sempre que nenhuma das duas funções são definidas. Assim temos um algoritmo para computar h, assim pela tese de Church h é URM-computável.

Capítulo 4

O Modelo BSS

4.1 Introdução

A teoria da computabilidade ou teoria da recursão tem se desenvolvido rapidamente, desde seu início, há aproximadamente 60 anos. Embora no início, praticamente só os lógicos estivessem interessados em teoria da computabilidade, atualmente a teoria das funções recursivas é importante na fundamentação teórica da computação, além de que seus métodos e questões estejam penetrando em outras disciplinas matemáticas.

Teorias computacionais precisam dar conta das propriedades de computabilidade de seus objetos. É através da noção de computabilidade que podemos estabelecer os limites das computações. Nesse sentido a tese de Church, em suas várias versões tem orientado os pesquisadores e técnicos em Ciências da Computação.

A computabilidade sobre conjuntos contáveis é obtida a partir do desenvolvimento de uma teoria da computabilidade sobre os números naturais [BL74] de tal modo que a computabilidade nos outros conjuntos se reduz à computabilidade no conjunto dos números naturais. Como cada número real parcial pode ser visto como limite de uma seqüência convergente de intervalos racionais e inversamente, podemos esperar que os números reais parciais tenham um papel numa teoria da computabilidade para conjuntos com a cardinalidade do contínuo semelhante à dos números naturais na teoria da computabilidade para conjuntos contáveis. Na literatura existem diferentes abordagens para a computabilidade nos números reais, mas, uma importante diferença entre estas abordagens está na maneira como é representado o número real. Num dos primeiros artigos sobre teoria dos domínios D. Scott [Sco70] sugeriu que o cpo consistindo de intervalos da reta real poderia ser usado para estudar a computabilidade dos chamados domínios (contínuos em nosso caso) efetivamente dados. Previamente Martin-Löf [ML70] construiu um espaço similar de aproximações. Em ambos os casos a reta real é mergulhada no espaço de aproximações onde a noção de computabilidade pode ser definida de uma maneira natural. Muitos resultados concernentes à teoria da computabilidade sobre os números reais podem ser obtidas neste contexto. Nesta abordagem uma aproximação da saída com precisão arbitrária é computada a partir de uma aproximação razoável da entrada [Bra95]. Uma outra linha de pesquisa para a computabilidade real foi desenvolvida em 1986 por Blum, Shub e Smale [BSS89]. Nesta aproximação, chamadas máquinas BSS, um número real é visto como uma entidade acabada e as funções computáveis são geradas a partir de uma classe de funções básicas (numa maneira similar as funções parciais recursivas).

Na primeira abordagem (de limite), as construções são baseadas na representação dos números reais como seqüências convergentes de intervalos racionais que apesar de não serem uma representação apropriada para implementar computações sobre números reais [Gia93], é razoável para uma teoria da computabilidade sobre números reais (parciais).

Esta visão permitiu a Bedregal e Acióly [BA97] extender a conhecida máquina de Turing, desenvolvida para computações em mundos contáveis, para uma máquina chamada por eles de C-Turing, na qual esta nova noção de computabilidade foi fortemente sustentada na bem consolidada e aceita Teoria dos Domínios.

As teorias de computabilidade e complexidade computacional clássicas vistas em sua maioria, trabalham sob problemas discretos (números inteiros, sob um alfabeto finito, grafos, etc.). Entretanto, precisaremos trabalhar sob grandezas contínuas, como números reais, complexos ou intervalos reais (no nosso caso, os intervalos reais devem ser vistos como grandezas contínuas, e as operações intervalares executadas sob o conjunto aplicadas nos extremos) para resolver os problemas da computação científica citadas na introdução desta dissertação. Temos dois modelos de máquinas que computam sob os contínuos, as Máquinas de Turing e as máquinas que operam com funções parciais recursivas.

Este capítulo se divide em 6 seções .

Seção 4.1. Introdução . Esta introdução .

Seção 4.2. O Modelo BSS Clássico. Apresentaremos nesta seção , toda teoria do modelo BSS, que servirá de base para o nosso novo modelo.

Seção 4.3. Computações em Máquinas BSS. Veremos as computações em máquinas BSS onde estará a definição formal do modelo, definiremos função processada e conjunto enumerável.

Seção 4.4. Observaremos a decidibilidade sob a estrutura algébrica anel de uma forma diferente da clássica tese de Church, alguns exemplos envolvendo o conjunto Cantor's Middle third, etc.

Seção 4.5. Fractais. Faremos uma breve abordagem teórica sobre os fractais, seus principais estudiosos e sua finalidade.

Seção 4.6. Exemplos. Por ser um capítulo importante, separamos uma seção só para os principais exemplos, dentre eles, vários conjuntos que envolvem fractais, uma Máquina BSS que calcula o maior inteiro, os tradicional problemas: da mochila (knapsack problem), o Método de Newton, várias figuras e modelos.

4.2 O Modelo BSS Clássico

Muitos métodos tem sido propostos para análise computacional com finalidade de resolver problemas no contínuo, dentre eles o modelo BSS, que se trata de um modelo que trabalha com funções parciais recursivas o qual apresentaremos a seguir.

O conceito de um modelo de máquina, que trabalha basicamente sob um anel ordenado, foi definido por Blum, Shub e Smale tendo foi o início de uma teoria da computação e complexidade sobre conjuntos contínuos. Essa máquina foi conhecida como BSS em homenagem aos seus criadores.

Definição 26 *Seja $N = \{1, \cdots, n\}$ para $n > 1$, A um anel ordenado, $l, m \in N$ e I=A^l, O=A^m, S=$\mathbb{Z}^+ \times \mathbb{Z}^+ \times A^\infty$ (entrada, saída e estado de possíveis configurações, respectivamente) onde $\mathbb{Z}^+$ é o conjunto dos números inteiros positivos. Uma* **máquina BSS** *M sobre A, com entrada I e saída O, é um grafo dirigido conexo com n nós.*

Estes nós são divididos em cinco categorias:

Seja v um nó pertencente ao grafo representativo do Modelo BSS. Dizemos que este nó v será um:

Nó-entrada *Se os arcos de entrada no nó v é zero (não chega nenhum arco) e possui apenas um arco de saída.*

Nó saída *Se os arcos de saída no nó v é zero.*

Nó-escolha *O nó v tem dois arcos de saída e seus sucessores são $\beta^-(v)$ ou $\beta^+(v)$.*

Vejamos a ilustração seguinte:

Em um nó escolha é feito uma tomada de decisão para qual nó seguirá: β^+ ou β^-, a partir de um critério previamente estabelecido.

Nó-computacional: *Neste caso, o nó v tem um arco de saída e tem associado uma função polinomial $g_v : S \to S$ (g_v pode ser racional se A é um corpo), este nó é necessário quando $g_v(i, j, x) = (g_v^1(i), g_v^2(j), g_v^3(x))$, onde $g_v^1(i) \in \{1, i+1\}$, $g_v^2(j) \in \{1, j+1\}$ e $g_v^3(x)$, $g_v^3 : A^\infty \to A^\infty$ trata-se da função propriamente dita.*

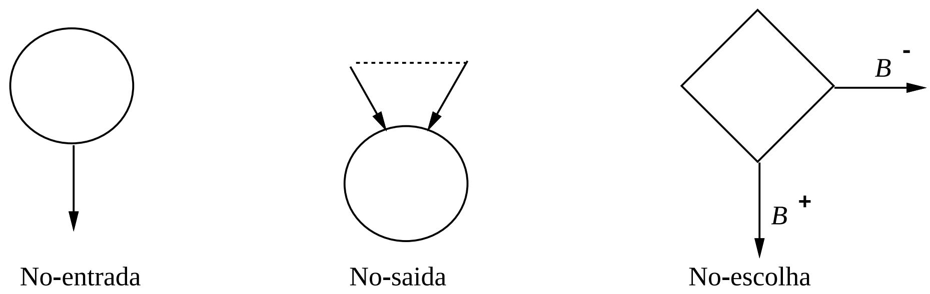

Figura 4.1: Nós de uma Máquina BSS

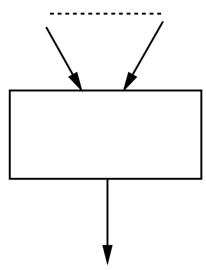

Figura 4.2: Nó computacional - Nó quinta-classe

Nó quinta classe *Este nó (tem um arco único de saída, e o próximo nó é β semelhante a um nó computacional) operando sobre v transformando-o no elemento $(i, j, x_1, x_2, \cdots, x_{j-1}, x_i, , x_{j+1}, \cdots) \in S$, isto é, x_i substitui x_j na j-ésima posição em A^∞. Não são feitas outras trocas no nó quinta classe. Se v representa este nó então esta função será escrita da seguinte forma:*

$$g_v : S \to S.$$

Discutiremos agora as questões sobre as modificações necessárias na máquina de dimensão finita. Utilizaremos $I = A^l, O = A^m$, ao mesmo tempo com um grafo dirigido (finito).

Os quatro primeiros nós são iguais diante de um grafo teórico, mas a definição de funções associadas deve ser extendida para ocultar o caso de dimensão infinita. Temos definido uma função polinomial $A^\infty \to A^\infty$. Mas este não afetará x_i para um i suficientemente grande na expressão $X = (x_1, x_2, \cdots) \in A^\infty$. Esta consideração forçou a introdução de outro tipo de nó dentro da máquina que foi chamado de nó quinta classe. O nó quinta classe é usado para acessar o x_i com o i suficientemente grande. Para estes nós foi utilizado $S = \mathbb{Z}^+ \times \mathbb{Z}^+ \times A^\infty$ com um elemento típico $(i, j, x_1, x_2, \cdots)$, i, j inteiros positivos.

É interessante observar que uma máquina BSS M é similar a uma máquina de acesso randômico clássica tendo um número infinito de registros e com capacidade de manter um elemento de A. Ele usa alguma entrada de A^l, dá alguma saída em A^m e pode ter um desempenho computacional (observada a complexidade) a nível polinomial (ou racional) envolvendo unicamente um número finito de registros. Esta máquina M também pode ter um desempenho simples se confrontar e executar uma instrução "escolha" dependendo do resultado da comparação.

Os dois registros adicionais inteiros são postos para permitir uma computação uniforme no caso que espaço de entrada é A^∞; de fato, sendo k_M a dimensão de M (o máximo entre as dimensões destas funções associadas com os nós computacionais de M) se o nó quinta classe é admitido, a máquina M pode usar e modificar variáveis indexadas a k_M.

Se o "nó quinta classe" não é admitido, a máquina BSS M apenas não poderá usar nem modificar variáveis com indexação sob k_M. Desta forma a entrada "ativa" é neste caso limitado por k_M.

Se quisermos uma máquina capaz de fazer computações que dependam da dimensão da entrada, nós temos que fazer em um nível não polinomial. Esta é a forma de adicionar dois registros inteiros (i e j). Eles são classificados em dois registros ordinários que podem ser incrementados ou reinicializados pelas instruções de computação ; quando o nó quinta classe é atingido o conteúdo do registro i-ésimo (A-valorado) é copiado para dentro do registro j-ésimo. Estes caminhos não são atualmente limitados pela dimensão da máquina, e assim, computações uniformes acima das ilimitadas são permitidas para receber espaço.

4.3 Computações em Máquinas BSS

Apresentaremos agora a definição formal da máquina BSS M.

Seja $N = \{1, \cdots, n\}$ para $n > 1$, A um anel ordenado, $l, m \in N$ e I=A^l, O=A^m, S=$\mathbb{Z}^+ \times \mathbb{Z}^+ \times A^\infty$ (entrada, saída e estado de possíveis configurações, respectivamente) onde $\mathbb{Z}^+$ é o conjunto dos números inteiros positivos. Uma **máquina BSS** M sobre A, com entrada I e saída O, é um grafo dirigido conexo com n nós.

Observaremos primeiro as seguintes notações :

Para $x \in S$, escreveremos x^k, para denotar x_{k+2} (o k-ésimo registro A-valorado), também definiremos $St_M = N \times S$ (o estado de descrição instantânea de M). A função sucessor imediato $H_M : St_M \to St_M$ é definida a seguir:

$$H_M(v, x) = (\beta'(v, \chi(x)), g_v(x))$$

onde:

- $\chi : S \to \{-1, 0, 1\}$

$$\chi(x) = \begin{cases} -1 & \text{, se } x^1 < 0 \\ 0 & \text{, se } x^1 = 0 \\ 1 & \text{, se } x^1 > 0 \end{cases}$$

- $\beta' : N \times \{-1, 0, 1\} \to N$

$$\beta'(v, \sigma) = \begin{cases} \text{n} & \text{, se } v = n \\ \beta(v) & \text{, se } v < n \text{ e } v \text{ não é um nó-escolha} \\ \beta^+(v) & \text{, se } n \text{ é um nó-escolha e } \sigma \in \{0, 1\} \\ \beta^-(v) & \text{, em outros casos} \end{cases}$$

- $g_v : S \to S$ é definido a seguir:

  1. Se v é um nó-computacional, g_v é a **função associada** com v;
  2. Se v é um quinto-nó, g_v é definido a seguir:

     $$g_v(i, j, x) = (i, j, y)$$

     com

     $$y_k = \begin{cases} x_i & \text{, se } k = j \\ x_k & \text{, em outros casos} \end{cases}$$

  3. Em outros casos, g_v é a identidade de S.

Além disso, duas funções são definidas:

- *A* **função de entrada** $In : I \to S$ tal que $In(x) = (1, 1, y)$ onde $y_{2k+1} = x_k$ para $k = 1, \cdots, l, y_1 = |x|$ (o tamanho da maior coordenada não-nula de x) e $y_k = 0$ para qualquer outro k.

- *A* **função de saída** $O' : S \to O$ tal que $O'(i, j, x) = y$ onde $y_k = x_{2k+1}$ para $k = 1, \cdots, m$.

Seja $y \in I$, a execução de M sobre a entrada y é a seqüência $(n_0, x_0), (n_1, x_1), \cdots$ satisfazendo:

i. $n_0 = 1$ e $x_0 = In(y)$;

ii. $\forall k (n_{k+1}, x_{k+1}) = H_M(n_k, x_k)$,

$$n_{k+1} = \beta(n_k, \chi(x_k)) \;\; x_{k+1} = g_{n_k}(x_k)$$

Diz-se que a máquina M está **parada** sobre a entrada y se $n_k = n$ para algum k, neste caso, definimos $T_M(y)$ com menor de tais k (o tempo de parada de M sobre a entrada y).

Seja:

$$\Omega_M = \{y \in I : M \text{ pára sobre a entrada } y\}$$

(o conjunto de paradas de M).

Se M pára sobre y, o estado $(n_{T_M}(y), x_{T_M}(y))$ é chamado de **estado final** de M sobre a entrada y, e nós dizemos que $O'(x_{T_M}(y))$ é a saída de M sobre a entrada y, mais precisamente, nós definimos a função (parcial) $\varphi_M : I \rightharpoonup O$ a seguir:

$$\varphi(y) = \begin{cases} O'(x) & \text{, se } M \text{ pára sobre a entrada y com estado final } (n, x) \\ \uparrow & \text{, em outros casos} \end{cases}$$

Esta é chamada de **função processada** por M.

Podemos dar a notação de conjuntos recursivos enumeráveis sobre A por analogia a noções dos casos discretos. Um conjunto $S \subseteq A^l$ é **recursivo enumerável** sobre A se existe uma máquina BSS M com espaço de entrada A^l, tal que $\Omega_M = S$. S é **decidível** ou **recursivo** sobre A se ambos S e S^c (o complementar de S) são recursivos enumeráveis sobre A.

4.4 Decidibilidade Sob um Anel

A noção de decidibilidade de um dado conjunto é bastante diferente nesta seção do clássico que segue a tese de Church. Não obstante, se vê facilmente que um sub-conjunto de $\mathbb{Z}$ é R.E. (ou decidível), se e somente se, estiver de acordo com a definição clássica.

Observe que a noção de conjunto recursivo depende estritamente do anel que é considerado. Como um exemplo, mostraremos agora que todo subconjunto de $\mathbb{Z}$ é decidível sob o anel $\mathbb{R}$.

Vejamos informalmente o seguinte caso. Seja $S \subseteq \mathbb{Z}^+$, que leva a seguinte constante real (em notação binária):

$$\alpha_S = 0.s_1 s_2 s_3 \cdots$$

onde $s_i = 1$ se $i \in S$, 0 se $i \notin S$. O valor constante α_S codifica o conjunto S. Agora considere a máquina M_S (sob A, com espaço de possíveis configurações S) descrita informalmente pelo seguinte programa:

1. ler n
2. se $n < 1$ então entre em loop para sempre
3. n não é um inteiro então entre em loop para sempre
4. $x_1 \leftarrow \lfloor 2^{n-1}\alpha_S \rfloor$
5. $x_2 \leftarrow \lfloor 2^{n}\alpha_S \rfloor$
6. $x_3 \leftarrow x_2 - 2x_1$
7. se $x_3 = 0$ então entre em loop para sempre

Veremos no exemplo 17 uma máquina para computar $\lfloor x \rfloor$ para um determinado x. Nesta máquina, α_S é uma constante interna que pode decidir o relacionamento com o conjunto S. É fácil ver (para ser mais preciso, este algoritmo só trabalha quando $\mathbb{Z}^+ \setminus S$ é infinito; de qualquer maneira, se este não é o caso, S ainda é decidível com um determinado algoritmo) que $\Omega_{M_S} = S$, e assim S é R.E. sob A. Podemos construir de um modo análogo, outra máquina com conjunto de parada $A \setminus S$, obtendo que S é decidível sob A.

Um mesmo resultado geral sobre conjuntos R.E. sob um determinado anel são obtidos por Blum, Shub e Smale: eles relacionam conjuntos R.E. com conjuntos semi-algébricos.

Seja A um anel, $f_1, \cdots, f_m$ um conjunto finito de n-polinômios em A e seja $\epsilon_1, \epsilon_2, \cdots, \epsilon_m \in \{-1, 0, 1\}$ um conjunto de "sinal-condição ". Considere a seguinte função:

$$\text{signum}(f_i(x)) = \epsilon_i \text{ onde } i = 1, \cdots, m$$

O conjunto de todas as soluções $x \in A^n$ que estão acima do sistema é chamado um **subconjunto semi-algébrico básico** de A^n. Um **conjunto semi-algébrico básico** é uma união finita de conjuntos semi-algébricos básicos . Podemos considerar essa definição idêntica ao caso de um quasi-anel.

Proposição 4 [BSS89]

Seja M uma máquina sob A, com estado de possíveis configurações $I = A^l$. Então Ω_M é uma união contável de subconjuntos semi-algébricos de I ■

Este resultado dá condições necessárias para um conjunto ser R.E. sob um anel ou um quasi-anel, e pode ser usado para mostrar facilmente que um determinado conjunto não é decidível.

Exemplo 16 *Considere o intervalo real $[0, 1]$, o divideremos em três sub-intervalos $[0, \frac{1}{3}], [\frac{1}{3}, \frac{2}{3}], [\frac{2}{3}, 1]$. Agora desconsideraremos o intervalo mediano tornando o conjunto outro intervalo . Através de algumas interações neste processo chegaremos ao chamado conjunto* **Cantor's Middle Third** *(CMT). Mostraremos agora que CMT^C (o complemento de CMT) é R.E. mas não é decidível sob A. A máquina M seguinte tem CMT^C como conjunto de parada:*

entrada x
se $(x < 0)$ ou $(x > 1)$ então pare
$x \leftarrow 3x$
repita
  se $(2 \leq x)$então $x \leftarrow x - 2$
  $x \leftarrow 3x$

até $1 < x < 2$

Assim CMT^C é R.E. sob R. Mas CMT^C não é decidível, como CMT não é R.E. sob R. De fato, ([Bar88]]) mostra que CMT é um conjunto incontável totalmente desconexo, enquanto em ([Bec86]) um conjunto semi-algébrico tem só um número finito de componentes conexos. Assim, pela Proposição acima, um conjunto R.E. pode ter só um número de componentes contáveis conectados, e assim CMT não é R.E. sob R.

Da mesma maneira, pode-se mostrar que a maioria dos conjuntos de Julia [Dev96] é indecidível sob R. É preciso mencionar aqui, o fato que podem ser ré-obtidos muitos resultados clássicos de teoria de funções recursivas numa colocação muito mais geral. Por exemplo, pode-se caracterizar a classe de funções computáveis, ou exibir uma máquina universal sob algum determinado anel ou quasi-anel.

4.5 Fractais

O próximo passo nesse capítulo será apresentar alguns exemplos de uso da Máquina BSS. Essa subseção servirá de base para o bom entendimento dos exemplos.

Para iniciarmos essa sub-seção primeiramente responderemos a duas questões: "O que são fractais? " e "para que os fractais servem?" [Est93] cita a definição de Mandelbrot: "Um fractal é uma curva cuja dimensão de Hausdorff-Besicovitch é maior que a dimensão Euclidiana". Para melhor entendimento, detalharemos essa definição.

4.5.1 O Início de Curvas de Fractal

Puxe uma linha em uma folha de papel. A Geometria de Euclideana define que esta é uma figura de uma dimensão. Agora estenda a linha. Faça deslizar ao redor, de um lado para outro, sem se cruzar, até que encha a folha inteira do papel. A Geometria de Euclides diz que esta ainda é uma linha, uma figura de uma dimensão. Mas nossa intuição nos mostra fortemente que se a linha enche o papel inteiro completamente, deveria ser bi-dimensional.

Tal pensamento começou uma revolução na matemática a aproximadamente cem anos. Matemáticos como Cantor, Von Kech, Peano, Hausdorff, e Besicovitch puxaram curvas e foram chamados de "monstros", "psicótico", e "patológico" por matemáticos tradicionais. Um novo tipo de dimensionamento foi proposto em que uma curva poderia ter uma dimensão fracionária. Foram descobertas técnicas de recursividade e expressões interativas que poderiam descrever curvas que têm dimensões fracionárias. Mas, em computadores digitais sem alta velocidade, o desenho de tal curva era um processo longo e tedioso; nem, tão pouco o progresso aconteceu nesse campo durante quase cem anos.

O advento dos computadores digitais fez a investigação de tal curva um campo frutífero. Das primeiras investigações, poderíamos entender o que estavam tentando para fazer. Queriam puxar curvas que pareciam ter características dimensionais mais complexas que foi explicado através de geometria tradicional.

Foram programados em computadores expressões matemáticas interativas muito simples nas quais o próximo estado de um parâmetro dependia somente de uma relação simples: do estado atual do parâmetro. A repetição foi executada muitas vezes e a localização resultante do parâmetro em cada estado era plotado. Os resultados se mostraram interessantes por ter muitas características. Em primeiro lugar, eles nunca se repetiram. Além disso, eles tenderam a ter a característica de auto-semelhança. Em outro palavra, se uma porção pequena do resultado fosse aumentada, sua forma estava muito parecida como uma porção grande do resultado original.

As curvas ainda não fizeram muito sentido em termos da matemática tradicional, e por conseguinte permaneceu um anátema aos matemáticos tradicionais. Dr. Benoit Mandelbrot foi a primeira pessoa a fazer uso de um computador digital para investigar fractais a fundo, e os resultados não foram bem-vindos calorosamente por matemáticos tradicionais [Dev96].

4.5.2 Para que os Fractais Servem

Já explicamos o que é um fractal, mas explicar de que maneira um fractal é usado é mais difícil. Mandelbrot explica que da mesma maneira que as formas da geometria tradicional é o modo natural de representar objetos artificial (quadrados, círculos, triângulos, etc.), curvas de fractais é o modo natural de representar objetos que acontecem na natureza. Assim, os fractais servem como meio de representar objetos de arte de cenas naturais. Além disso, os fractais acontecem naturalmente nas expressões matemáticas tão quanto a predição de sistemas de tempo, a descrição de fluxo turbulento de líquido, e o crescimento e declínio de populações. Os Fractais também são úteis em transformações dimensionais que podem ser usadas por expressar e comprimir dados gráficos. Ignorando o valor artístico, a melhor resposta para a pergunta "para que os fractais servem?" é que os fractais parecem prover soluções para muitos perguntas previamente sem resposta às fronteiras das ciências físicas. Por conseguinte, para trabalhar às fronteiras da ciência, a pessoa precisa entender como são os fractais e como trabalhar com eles.

4.6 Exemplos

Dada a teoria, veremos agora alguns exemplos de computabilidade utilizando a máquina BSS. Nos casos pertinentes ilustraremos seus respectivos fractais.

Exemplo 17 *Dada uma Máquina BSS que computa o maior inteiro em* $x, \lfloor\ x\ \rfloor$ *, para* $x \geq 0$ *em* $\mathbb{R}$*:*

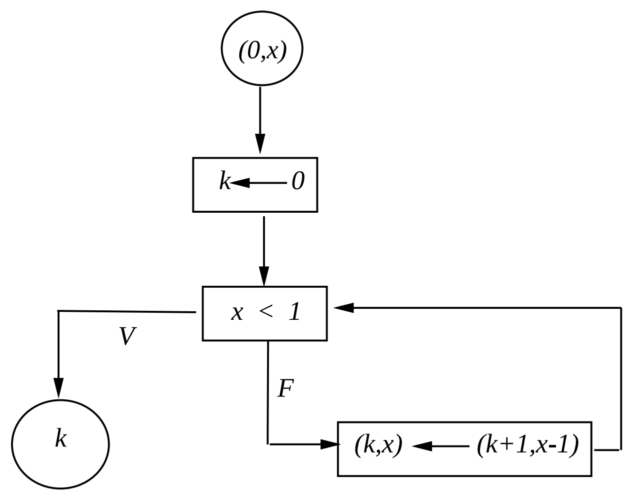

Figura 4.3: Máquina BSS computando o maior inteiro em x

onde em:

1 - Entrada da segunda coordenada de $\mathbb{R}^2$ x *onde* $x \in \mathbb{R}$

2 - Nó escolha onde se $x < 1$ *passa para o nó 3, senão para o nó 4*

3 - Nó saída: sai k *para a primeira coordenada*

4 - Nó computacional: acontece uma operação e retorna ao nó 2.

Observe que para $x > 0$ *em* R*, o custo da computação* $\lfloor x \rfloor$ *é de* $\lfloor x + 1 \rfloor$ *comparações e* $\lfloor x \rfloor$ *de aritmética basica computacional. Usando busca binária este custo pode ser reduzido para* $O(log(x+1))$ *mas essencialmente, não mais em nosso modelo. Ele é interessante para observar que em modelos de computação onde* $\lfloor x \rfloor$ *tão bem quanto as operações aritmética básica podem ser computadas em tempo constante, aparentemente problemas difíceis tal qual fatoração de inteiros e testes de satisfatoriedade de fórmulas proposicionais podem ser resolvidas eficientemente em tempo polinomial [Bol95].*

Exemplo 18 *O Conjunto de Mandelbrot*

Para introduzirmos o conjunto Mandelbrot estudaremos primeiramente a dicotomia fundamental:

Seja $Q_c(z) = z^2 + c$*, uma função quadrática,* K_c *um conjunto cantor,* z *e* c *números complexos. Então:*

1. *A órbita do ponto crítico* 0 *tende para o infinito, quando* K_c *consiste infinitamente de muitos componentes disjuntos, ou*

2. *A órbita de* 0 *permanece constante quando* K_c *estiver conectado.*

A dicotomia fundamental indica que há só dois tipos básicos de conjuntos Julia ocupados para Q_c*, esses são conectados e consistem infinitamente de muitos componentes disjuntos. Além disso, é a órbita de* 0 *que determina qual deste dois casos suportará. Isto conduz a uma definição importante.*

O **conjunto mandelbrot** η *consiste em todos os c-valores para qual o Conjunto Julia* K_c *estar conectado. Equivalentemente,*

$$\eta = \{c \in \mathbb{C} / |Q_c^n(0)| \to \infty\}$$

É importante citar que o conjunto Mandelbrot é um subconjunto de espaço de parâmetros para polinômios quadráticos.

O Conjunto de Mandelbrot é um conjunto decidível?

É de certa forma extraordinariamente complicado observando o fractal do conjunto Mandelbrot. Embora as regras que provêem sua definição são surpreendentemente simples, o próprio conjunto exibe uma variedade infinita de estruturas altamente elaboradas.

O conjunto de Mandelbrot

Este poderia ser um exemplo de não recursividade (isto é, indecidibilidade) do conjunto.

É conhecido que o limite do conjunto Mandelbrot tem uma estrutura rica e complexa. (Veja a próxima figura onde uma parte deste limite é mostrada como exemplo).

No limite: Vale do Cavalo-Marinho

Penrose é motivado a usar esta questão para fazer um argumento contra a inteligência artificial. Achamos que a pergunta sobre decidibilidade do conjunto de Mandelbrot tem outra justificação. Podemos responder em parte e pode servir de motivação à pergunta: Pode-se decidir se uma equação diferencial é caótica?

O conjunto Mandelbrot η *é definido como o conjunto de números complexos c tal que a sucessão* $c, c^2+c, (c^2+c)^2+c, \cdots$ *permanece conectado.*

Mais formalmente, seja $c \in \mathbb{C}$*, os números complexos, seja* $p_c(z) = z^2 + c$ *e seja* $p_c^n(z)$ *a n-ésima interação de* p_c *aplicado a z. Quer dizer,*

$$p_c^n(z) = p_c(p_c(p_c(p_c(z\cdots)))),\ n \ \textit{passos}.$$

assim $p_c(0) = c$*,* $p_c^2(0) = c^2 + c, \cdots$*. Então* η *é o complemento do conjunto*

$$\eta' = \{c \in \mathbb{C} \mid p_c^n(0) \to \infty \ \textit{e} \ n \to \infty\}.$$

O conjunto η *também pode ser descrito como o conjunto de entrada c que não pára, diante do quadro da figura 4.4:*

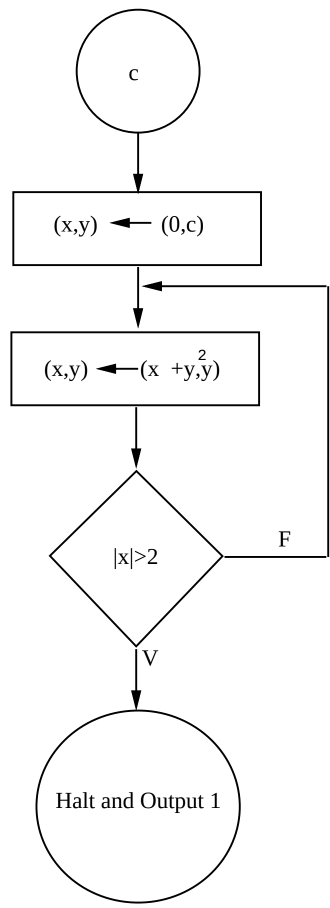

Figura 4.4: Conjunto η

MAQUINA 1: Um diagrama de fluxo associado ao conjunto de Mandelbrot

Para responder a questão de Penrose, precisamos de uma "máquina" ou "algoritmo" que, determinada entrada c, um número complexo, decidirá em um número finito de passos se c está ou não em η *.*

MAQUINA 2: Máquina de decisão desejada para η

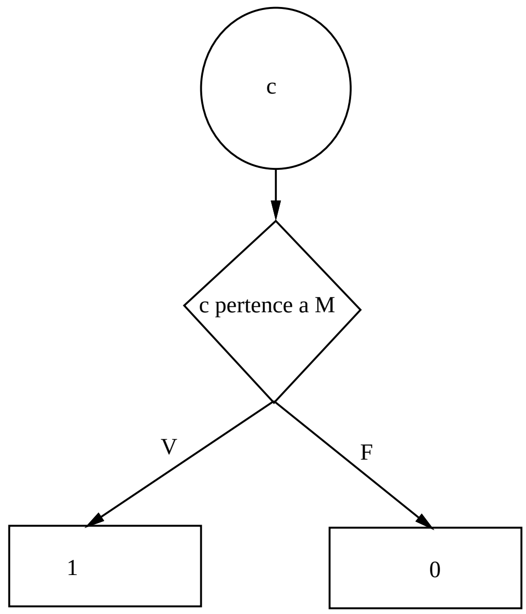

Figura 4.5: Máquina BSS para decisão

Exemplo 19 Exemplo do conjunto Julia de $T(z) = z^2 + 4$

Observaremos agora uma função polinomial $T : \mathbb{C} \to \mathbb{C}$ do ponto de vista de interações ou como um sistema dinâmico complexo. Será escrito $T^2 = T(T(z))$ e T^k para a composição de T com si próprio k vezes. Seja $T(z) = z^2 + 4$.

Observe que se $|z| > 2$, $|T^K(z)| \to \infty$ quando $k \to \infty$

Uma representação deste conjunto na Máquina BSS da figura 4.6:

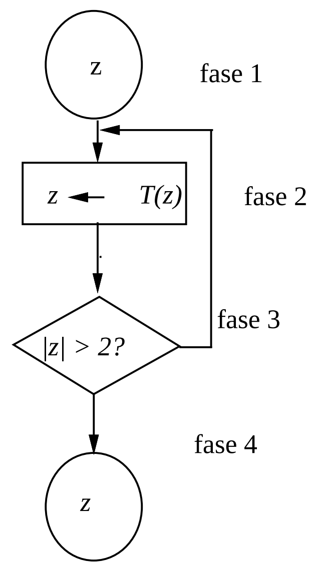

Figura 4.6: Máquina BSS - Representação do modelo do conjunto Julia

Daremos o nome a esta estrutura de Máquina M. Em M existem quatro nós, sendo vistos na seguinte seqüência: fase 1:nó entrada, fase 2: nó computacional, fase 3: nó escolha, fase 4: nó saída. Novamente temos um exemplo da máquina BSS sobre os números complexos $\mathbb{C}$.

Os Principais Fractais do Conjunto Julia

Exemplo 20 O problema da mochila

Analisaremos este exemplo agora utilizando-se de um anel comutativo A com unidade. Para ser mais específico, uma suposição seria A ser o anel do conjunto dos números inteiros $\mathbb{Z}$, os racionais $\mathbb{Q}$, os reais $\mathbb{R}$, ou dos números complexos $\mathbb{C}$. Consideremos o seguinte problema:

Pegar $x_1, \cdots, x_n \in A$ decidir se existe um subconjunto não-vazio $S \subset \{1, \cdots, n\}$ tal que $\sum_{i \in S} x_i = 1$.

Este problema ficou sendo conhecido como **O Problema da mochila** *ou* **Knapsack Problem** *sobre A. Basicamente, podemos ver esse problema como sendo:*

Pegar $x_1, \cdots, x_n, c \in A$ decidir se existe um subconjunto não-vazio $S \subset \{1, \cdots, n\}$ tal que $\sum_{i \in S} x_i = c$.

Neste caso imaginamos c sendo toda a capacidade da mochila, x_i o volume dos itens, e a questão é: como podemos preencher a mochila com a máxima capacidade utilizando-se de um subconjunto dos itens?

Observe que em nossa formulação do problema da mochila, o anel A não deve ser ordenado. Assim, nós consideraremos a máquina aqui apenas um ramo sobre a comparação de igualidade sobre R.

Em muitos casos, o Problema da Mochila é similar ao nosso problema de decidibilidade. Seja K_n o conjunto da mochila tal que apresentaremos da seguinte forma:

$$K_n = \{x \in R^n | \exists b \in \{0,1\}^n \text{ tal que } \sum b_i x_i = 1\}.$$

Agora observaremos uma máquina que decidirá, dado $x \in R^n$ se $x \in K_n$. Mas diferentemente dos exemplos anteriores, o **Problema da Mochila** *é facilmente decidível, usar $x \in R^n$, enumerar sucessivamente os elementos não-zero de $\{0,1\}^n$ e evoluir o seguinte somatório: $\sum b_i x_i$.*

Observe a Máquina da figura 4.7.

Fixando n, nós podemos verificar o espaço de entrada para nossa máquina de dimensão finita e espaço R^n. De qualquer maneira, a máquina é "uniforme" em n, nós podemos naturalmente observar o espaço de entrada em R^∞, a infinita direção total de A (essencialmente, o espaço finito, mas ilimitadas seqüências sobre R). Assim, fazendo n variar, temos um esboço real de uma máquina para decidir o conjunto em $K = \bigcup K_n$. No caso de dimensão finita, o teste polinomial dos nós-escolha podem ser embutidos dentro da máquina. No caso de "dimensão infinita", o teste dos nós-escolha são computados por subrotinas.

Exemplo 21 *O Método de Newton*

Os dois exemplos anteriores levantaram perguntas relativo à existência de máquinas que decidiriam Sim ou Não para questões da forma: Determinado $z \in \mathbb{C}$, é $z \in \eta$ (ou J)? onde η e J são os conjuntos Mandelbrot e Julia, respectivamente.

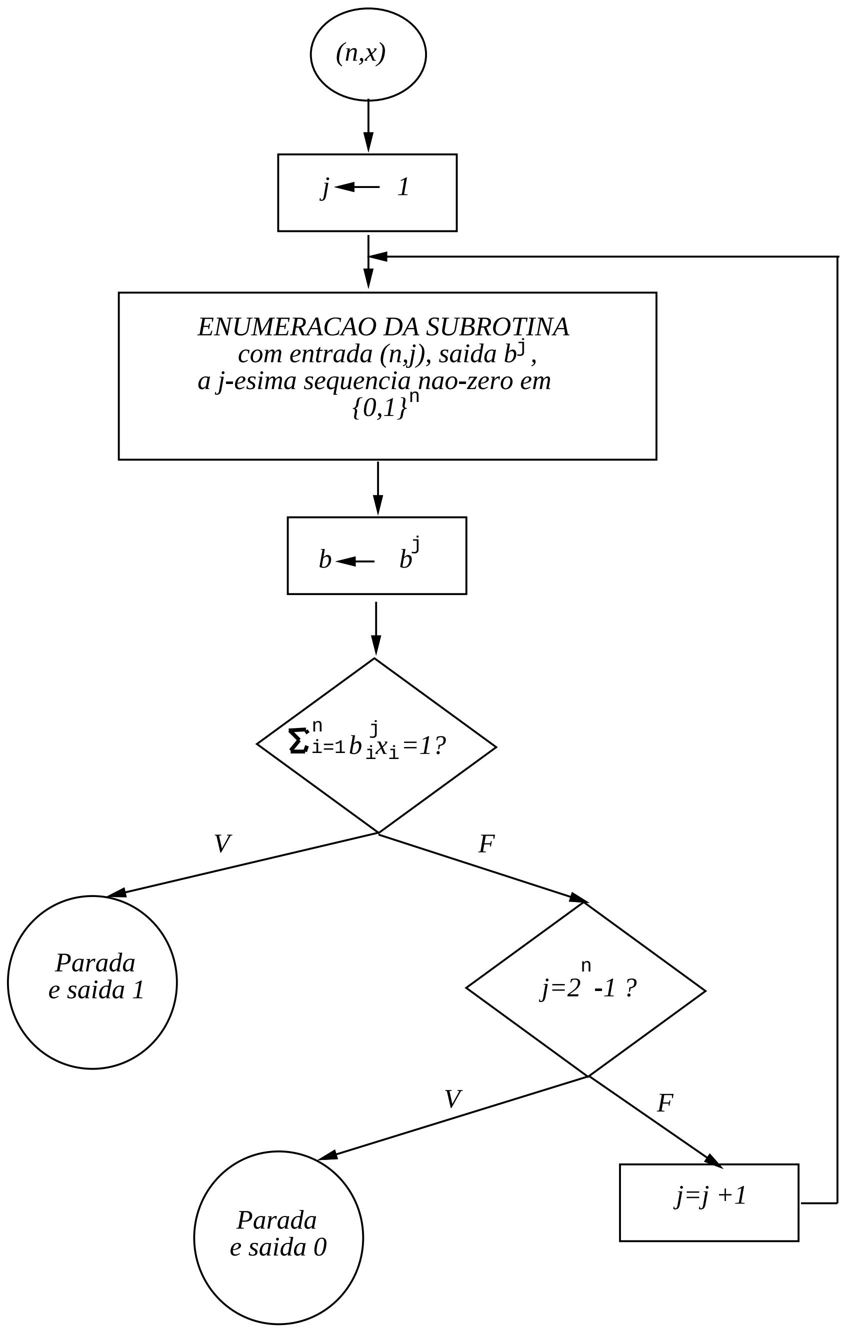

Figura 4.7: Busca exaustiva para máquina resolver o Problema da Mochila

Por outro lado, frequentemente necessitamos de algoritmos que procuram soluções a problemas da forma: Dado um polinômio f, ache ζ tal que $f(\zeta) = 0$.

O método de Newton é o "algoritmo de busca" sine qua non *de análise numérica e computação científica.*

Dado uma variável polinomial $f(z)$ sob os números complexos $\mathbb{C}$, defina-se o **Newton endomorfismo** $N_f : \mathbb{C} \to \mathbb{C}$ *por:*

$$N_f(z) = z - \frac{f(z)}{f'(z)}$$

Esta função é definida contanto que $f'(z) \neq 0$.

Agora para o método de Newton: Escolha ponto inicial $z_0 \in \mathbb{C}$ e gere a **órbita**

$$z_0, z_1 = N_f(z_0), z_2 = N_f(z_1), \cdots, z_{k+1} = N_f(z_k) = N_f^{k+1}(z_0), \cdots$$

Alguma regra de parada tal qual "pára se $\mid f(z_k) \mid < \epsilon$ e a saída z_k for determinada". Em prática, se o procedimento não parar com uma saída depois de um certo número de interações, ou fica indefinido em alguma fase, ou um novo ponto inicial é escolhido.

Podemos construir uma Máquina BSS para representar o método de Newton como na Figura 4.8. Nesta simples máquina, assumiremos que f, N_f e ϵ serão gerados internamente. Um ponto inicial $z \in \mathbb{C}$ é a "entrada" da máquina.

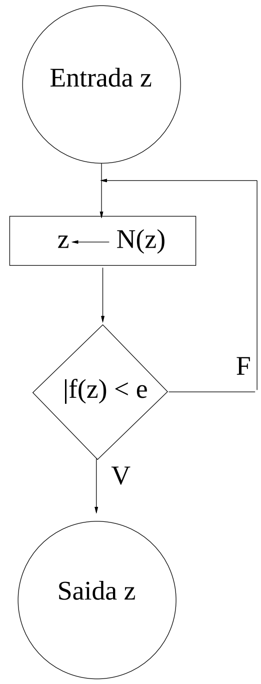

Figura 4.8: Máquina BSS - Método de Newton

O fractal que representa a Máquina de Newton para f é

Capítulo 5

Matemática Intervalar

5.1 Introdução

A computação científica é uma área relativamente nova da ciência da computação, com aplicações em várias áreas científicas tais como física [Kor92], química [SS93], biometria [GV94], estatística [Orl92], comunicação [EZ94], circuitos [Gar94,MK94], movimento de satélites [VV94], etc. A qualidade do resultado em computação científica depende do conhecimento e controle dos erros na computação. Algoritmos convencionais computam uma resposta e talvez uma estimativa de erro. Entretanto o usuário não pode obter uma resposta exata sem a ajuda de uma análise de erro, que pode ser dispendiosa, extensiva e nem sempre factível [Aci91]. Por outro lado, técnicas intervalares podem ser programadas por computador contendo em sua computação uma análise rigorosa do erro no resultado.

A Matemática Intervalar é uma teoria originada na década de 60, com o objetivo de responder questões de acuracidade e eficiência que emergem na prática da computação científica. Entretanto a denominação "Matemática Intervalar" foi proposta por Leslie Fox em 1974, combinando diferentes áreas tais como aritmética intervalar, análise intervalar, topologia intervalar, álgebra intervalar, dentre outras [DNC95].

Existem três fontes de erros em computação numérica: A primeira e mais séria, porque não é possível torná-la arbitrariamente pequena via computação adicional, é a propagação do erro nos **dados iniciais**. *A* segunda e terceira espécies de erros em computação, **arredondamento** , causado por computações com um arredondamento de dígitos finitos e o **erro de truncamento**, causado quando se trunca sequências infinitas de operações aritméticas, após um número finito de etapas, pode sempre, em princípio, serem minorados tornando-os arbitrariamente pequenos através de bastante computação. A Matemática Intervalar tem como objetivo fornecer esquemas computacionais de modo que, em se efetuando bastante computações, a segunda e terceira espécies de erros se tornem tão pequenos quanto se queira. *A* propagação do erro nos dados iniciais e a acumulação do erro de arredondamento em qualquer **sequência finita de operações aritméticas** podem ser ambos rigosamente controlados, simplesmente, colocando-os em aritmética intervalar em vez de aritmética de máquina ordinária.

Este capítulo esta dividido em três seções .

Seção 5.1. Introdução . Esta introdução .

Seção 5.2. Definições Básicas. Apresentaremos aqui as principais definições sobre intervalos, as operações aritméticas intervalares, as funções intervalares. Faremos uma introdução a teoria dos domínios contínuos e a teoria dos domínuos contínuos intervalares.

Seção 5.3. O Quasi-anel dos Intervalos Reais. Apresentaremos enfim, a estrutura que que o conjuntos dos intervalos reais formam: o quasi-anel, faremos a demonstração necessária utilizando-se das definições aritméticas intervalares e das propriedades de corpo dos reais.

5.2 Definições Básicas

Definição 27 *Seja* $\mathbb{R}$ *o conjunto dos números Reais, e* $x_1, x_2 \in \mathbb{R}$*, então o conjunto* $\{x \in \mathbb{R}/x_1 \leq x \leq x_2\}$ *é um* **intervalo real** *e sera denotado por* $X = [x_1, x_2]$*.*

Definição 28 *Sejam* $I(\mathbb{R})$ *o conjunto dos intervalos reais,* $a = [a_1, a_2]$ *e* $b = [b_1, b_2]$ *dois intervalos de* $I(\mathbb{R})$*. Dizemos que os dois intervalos são* **iguais** *se e somente se* $a_1 = b_1$ *e* $a_2 = b_2$*.*

Definição 29 *Sejam* $A, B \in I(\mathbb{R})$ *dois intervalos. As operações de* **soma, subtração, multiplicação e divisão** *em* $I(\mathbb{R})$ *são definidas por* $A \odot B = \{a \odot b / a \in A, b \in B\}$*, onde* $\odot \in \{+, -, \cdot, /\}$ *é quaisquer uma das quatro operações aritméticas.*

É importante observar que para a operação de divisão, precisamos assumir que $0 \notin B$, pois, caso contrário, a operação não estará bem definida.

Em decorrência desta definição as operações aritméticas com intervalos são definidas como segue. É possível caracterizar estas operações intervalares em função dos seus extremos.

Teorema 7 (Moo66) *(Operações Aritméticas Intervalares)*
Sejam $[a, b], [c, d] \in I(\mathbb{R})$ *dois intervalos reais.*
então, as seguintes propriedades valem:

- $[a,b]+[c,d]=[a+c,b+d]$
- $[a,b]-[c,d]=[a-d,b-c]$
- $[a,b]\cdot[c,d]=[Min\{ac,bc,ad,bd\},Max\{ac,bc,ad,bd\}]$
- $[a,b]/[c,d]=[a,b]\cdot[\frac{1}{d},\frac{1}{c}]$ *se* $0\notin[c,d]$ ■

Este teorema nos permite visualizar um intervalo como sendo uma natureza dual: como um conjunto e como um par de números reais.

Teorema 8 *(Propriedades Algébricas de* $I(\mathbb{R})$*)*
Sejam $a,b,c\in I(\mathbb{R})$ *intervalos reais. Então, valem:*

- $A+B=B+A$
- $A\cdot B=B\cdot A$
- $A+(B+C)=(A+B)+C$
- $A\cdot(B\cdot C)=(A\cdot B)\cdot C$
- $\exists!0=[0,0]\in I(\mathbb{R})$ *tal que* $A+0=0+A=A$
- $\exists!1=[1,1]\in I(\mathbb{R})$ *tal que* $A\cdot 1=1\cdot A=A$
- $a\cdot(B+C)\subseteq(A\cdot B)+(A\cdot C)$
- $\forall\alpha\in\mathbb{R}$ *vale que* $\alpha\cdot(B+C)=(\alpha\cdot B)+(\alpha\cdot C)$

Em $I(\mathbb{R})$ *não valem as seguintes propriedades:*

- $\forall A\in I(\mathbb{R})\ \exists -A\in I(\mathbb{R})/A+(-A)=0$
- $\forall A\in I(\mathbb{R})$ *com* $0\notin A\ \exists A^{-1}\in I(\mathbb{R})/A\cdot A^{-1}=1$
- $A\cdot(B+C)=(A\cdot B)+(A\cdot C)$ ■

Outras operações que podemos definir sobre o conjunto $I(\mathbb{R})$ são a interseção e a união de intervalos, $A\cap B=\{x/x\in A$ e $x\in B\}$ e $A\cup B=\{x/x\in A$ ou $x\in B\}$ (a união de intervalos só é um intervalo quando $A\cap B\neq\emptyset$).

Proposição 5 *(Interseção de Dois Intervalos)*
Sejam $A=[a_1,a_2]$ *e* $B=[b_1,b_2]$ *dois intervalos. Então a* **interseção** *dos intervalos* A *e* B *é o intervalo* $A\cap B=[max\{a_1,b_1\},min\{a_2,b_2\}]$*, se* $max\{a_1,b_1\}\leq min\{a_2,b_2\}$*. Se* $min\{a_2,b_2\}<max\{a_1,b_1\}$ *então* $A\cap B=\emptyset$.

Teorema 9 *(Propriedades da interseção)*
Sejam $A, B, C, D \in I(\mathbb{R})$. Se $A \subseteq C$ e $B \subseteq D$, então, $A \cap B \subseteq C \cap D$.

Proposição 6 *(União de Dois Intervalos)*
Sejam $A = [a_1, a_2]$ e $B = [b_1, b_2]$ dois intervalos tais que $A \cap B \neq \emptyset$. Então a **União** *dos intervalos A e B é o intervalo $A \cup B = [min\{a_1, b_1\}; max\{a_2, b_2\}]$.*

Nas duas definições a seguir observaremos os intervalos como estando imersos na retal real, ou seja, não teremos mais pontos fixos, teremos seguimentos de reta que terão como extremos os valores dos intervalos. Essas definições serão trabalhadas adiante com o conceito de distância de dois intervalos.

Definição 30 *(Diâmetro)*
Seja $A = [a_1, a_2] \in I(\mathbb{R})$ um intervalo. Então o **diâmetro** *do intervalo A é o número real não negativo $d = a_2 - a_1$.*

representaremos o diâmetro por Dia.

Definição 31 *(Ponto Médio de Um Intervalo)*
Seja $A = [a_1, a_2] \in I(\mathbb{R})$ um intervalo. Então o **ponto médio** *do intervalo A é o número real $m = \frac{a_1+a_2}{2}$. Denotamos por $med(A)$.*

Podemos observar que os $I(\mathbb{R})$, o conjunto de todos os intervalos sobre os reais, estabelece um estrutura algébrica que generaliza a estrutura dos números reais. A aritmética intervalar satisfaz a propriedade da monotonicidade [Moo66], isto é, se $I, J, K, L \in I(\mathbb{R})$, $I \subseteq K$ and $J \subseteq L$, então

- $I + J \subseteq K + L$
- $I - J \subseteq K - L$
- $I \cdot J \subseteq K \cdot L$
- $I/J \subseteq K/L$ (se $0 \notin J, L$)

5.2.1 Funções Intervalares

Nesta seção veremos como se pode definir as principais funções intervalares. Iniciaremos definindo informalmente a função intervalar:

Definição 32 *(Função Intervalar)*
Seja

$$f(x) = \begin{array}{l} \mathbf{X} \rightarrow \mathbf{Y} \\ X \mapsto f(X) \end{array}$$

uma função. Se X= $Dom(f) \subseteq I(\mathbb{R})$ e Y= $Cod(f) \subseteq I(\mathbb{R})$, então nós dizemos que f é uma **função intervalar de uma variável intervalar**.

Definição 33 *(Inclusão Intervalar)*
Dado $x \in \mathbb{R}$, dizemos que $X \in I(\mathbb{R})$ é uma **inclusão intervalar** *de x se $x \in X$.*

Exemplo 22 *O intervalo [3,4] é uma inclusão intervalar para o número real π.*

Teorema 10 *(Princípio da Inclusão da Aritmética Intervalar)*
Sejam $A, B \in I(\mathbb{R})$ dois intervalos reais e $\alpha, \beta \in \mathbb{R}$ dois números reais. Então $\alpha \odot \beta \in A \odot B$, sempre que $\alpha \in A$ e $\beta \in B$ para $\odot \in \{+, \cdot, -, /\}$.

Exemplo 23 *Tem-se que $2 \in [1,3]$ e $5 \in [4,6]$. Assim $10 = 2 \cdot 5 \in [4,18] = [1,3] \cdot [4;6]$.*

Imagem e Avaliação Intervalares

Daremos, agora, uma breve descrição de como as funções intervalares podem ser definidas a partir de funções reais. Depois, veremos alguns teoremas que garantem as propriedades básicas da avaliação e da imagem intervalar de funções reais.

Definição 34 *(Imagem Intervalar de uma Função Real)*
Sejam f uma função real de variável real e X um intervalo tal que $X \subseteq Dom(f)$ e f é contínua em X. Definimos a **imagem intervalar** *da função f em X (ou simplesmente a* **imagem** *de f em X) como sendo o intervalo definido por:*

$$Im(f, X) = \{f(x)/x \in X\} = [min\{f(x) \mid x \in X\}, max\{f(x) \mid x \in X\}].$$

Note que esta é uma maneira natural de se definir funções intervalares a partir de funções reais, ou seja, definimos $Y = f(X) = Im(f, X)$, onde f é uma função real e X é um intervalo contido no domínio da função f.

Observe, também, que se $X = [x, x]$ é um intervalo pontual (ou degenerado), então $Y = f(X)$ também é um intervalo pontual, dado por $Y = [f(x), f(x)]$.

Assim a função real está contida nesta extensão intervalar. Se $X = [x_1, x_2]$ é um intervalo com Dia$(X) > 0$, então $Im(f, X)$ é o intervalo de menor diâmetro que contém todos os valores reais de $f(x)$, quando $x \in X$.

Definição 35 *(Avaliação Intervalar de uma função real)*
Sejam f uma função real de variável real x e X um intervalo. Definimos a **avaliação intervalar** *de f em X (ou extensão intervalar de f) como sendo a função intervalar $F(X)$, definida da seguinte maneira: cada ocorrência da variável real x é substituída pela variável intervalar X e cada operação real $(+, -, \cdot, /)$ é substituída pela respectiva operação intervalar de tal modo que, quando $X = [x, x]$ for um intervalo pontual, então $F(X) = f(x)$.*

Veremos a seguir alguns teoremas e corolários que estabelecem as propriedades básicas das funções intervalares $Im(f, X)$ e $F(X)$. As demonstrações cabíveis encontram-se em [Bra95].

Teorema 11 *Seja f uma função real de variável real e X, Y dois intervalos. Se $X \subseteq Y \subseteq Dom(f)$, então $Im(f, X) \subseteq Im(f, Y)$ [Cut97]* ■

Teorema 12 *Seja f uma função real contínua de variável real x e $F(X)$ a sua respectiva extensão intervalar. Então, vale que $Im(f, X) \subseteq F(X)$ para todo $X \subseteq Dom(f)$* ■

Corolário 3 *Se $X = [x, x]$ então $Im(f, X) = F(X)$.*

Corolário 4 *Se $x \in X$, então $Im(f, x) \in Im(f, X)$ e $f(x) \in F(X)$.*

A imagem de uma função real $f : D \subseteq \mathbb{R} \longrightarrow \mathbb{R}$, sobre o conjunto $X \subseteq D$, é $Im(f, X) = \{f(x) \mid x \in X\}$. Quando $X = [a, b]$ é um intervalo fechado, limitado e f uma função contínua, $Im(f, X)$ será também um intervalo da mesma natureza. Um problema fundamental e típico da análise intervalar é o cálculo de $Im(f, X)$ ou ao menos uma boa aproximação para ele. Se f está definida em termos das operações aritméticas e funções com extenções intervalares, então o uso de computações intervalares [Moo66] nos fornece uma extenção intervalar F tal que $Im(f, X) \subseteq F(X)$, para $X \subseteq D$. Este cálculo tem a vantagem de ser completamente automático e não requerer conhecimento especial das propriedades de f .

Funções Intervalares Básicas

Definição 36 *(Função exponencial)*
Definimos uma função exponencial intervalar

$$Exp : \begin{array}{l} \mathbf{I}(\mathbb{R}) \rightarrow \mathbf{I}(\mathbb{R}) \\ X \mapsto Exp(X) \end{array}$$

por: $Exp([x_1, x_2]) = [exp(x_1), exp(x_2)]$.

Definição 37 *(Função Logaritmo)*
Definimos a função logaritmo intervalar

$$Ln : \begin{array}{l} \mathbf{I}(\mathbb{R}^+) \rightarrow \mathbf{I}(\mathbb{R}) \\ X \mapsto Ln(X) \end{array}$$

por: $Ln([x_1, x_2]) = [ln(x_1), ln(x_2)]$, *onde* $I(\mathbb{R}^+) = \{[x_1, x_2] \in I(\mathbb{R})/x_1 > 0\}$.

Definição 38 *(Função Seno)*
Definimos a função seno intervalar

$$Sen : \begin{array}{l} \mathbf{I}(\mathbb{R}) \rightarrow \mathbf{I}(\mathbb{R}) \\ X \mapsto Sen(X) \end{array}$$

por: $Sen(X) = Im(sen(x), X)$.

Definição 39 *(Função Cosseno)*
Definimos a função cosseno intervalar

$$Cos : \begin{array}{l} \mathbf{I}(\mathbb{R}) \rightarrow \mathbf{I}(\mathbb{R}) \\ X \mapsto Cos(X) \end{array}$$

por: $Cos(X) = Im(cos(x), X) = Sen(X + [\frac{\pi}{2}, \frac{\pi}{2}])$.

Definição 40 *(Função Raiz Quadrada)*
Definimos a função intervalar
$Y = \sqrt{X}$ *por:* $\sqrt{X} = \sqrt{[x_1, x_2]} = [\sqrt{x_1}, \sqrt{x_2}] = Im(\sqrt{x}, X)$.

Definição 41 *(Função X^2)*
Seja f uma função real de variável real x. Definimos a função

$$X^2 : \begin{array}{l} \mathbf{I}(\mathbb{R}) \rightarrow \mathbf{I}(\mathbb{R}) \\ X \mapsto X^2 \end{array}$$

por:

$$X^2 = Im(f^2, X) = \begin{cases} [x_1^2, x_2^2] & , se\ x_1 > 0 \\ [x_2^2, x_1^2] & , se\ x_2 < 0 \\ [0, |X|^2] & , se\ 0 \in X \end{cases}$$

Definição 42 *(Função Potência Intervalar)*
Definimos a função

$$X^n : \begin{array}{l} \mathbf{I}(\mathbb{R}) \rightarrow \mathbf{I}(\mathbb{R}) \\ X \mapsto X^n \end{array}$$

por:
$[X^n] = Im(f^n, X) e X^0 = [1, 1]$

Exemplo 24 *Seja $X = [-2, 3]$. Então, $X^3 = [-2, 3]^3 = [-8, 27]$*

Definição 43 *(Função Arco-tangente Intervalar)*
Definimos a função arco-tangente intervalar

$$Arctan : \begin{array}{l} \mathbf{I}(\mathbb{R}) \to \mathbf{I}(\mathbb{R}) \\ X \mapsto Arctan(X) \end{array}$$

por: $Arctan([x_1, x_2]) = [arctan(x_1), arctan(x_2)]$

Como a Análise Intervalar é uma teoria matemática cujo principal objetivo é responder a questões de exatidão e eficiência que surgem na prática da computação científica, é razoável se esperar que técnicas intervalares forneçam a garantia do controle de erros e possam ser aplicadas quase que automaticamente. Considerando técnicas intervalares o diâmetro de um intervalo solução é um indicativo da influência do erro no resultado final. Entretanto, respostas intervalares não garantem que estejam incluídos algo do nosso interesse. Para isso é preciso uma fundamentação matemática acurada em cada estágio do projeto e implementação do algoritmo. Para tirarmos partido das vantagens das técnicas intervalares, os algoritmos devem ser intervalares, isto é, eles não devem ser uma versão intervalar de um algoritmo pontual.

5.2.2 A Teoria dos Domínios de Scott

Acióly [Aci91] introduziu a idéia da utilização da Teoria dos Domínios de Scott e o λ-cálculo como uma fundamentação para a Análise Intervalar alterando a noção de Domínios de Scott. O conjunto dos números reais $\mathbb{R}$ com sua topologia usual (Hausdorff) é um subespaço topológico do espaço dos intervalos de Moore, ordenado pela relação de inclusão, com a topologia de Scott, identificando-se os números reais x e os intervalos degenerados $[x, x]$. Então, como funções contínuas para a topologia de Scott são monotônicas para a inclusão, [Aci91] sugeriu que se descartasse a topologia de Moore e que fosse adotada a topologia de Scott. Há também uma teoria de computabilidade para os Domínios de Scott. Considera o Domínio Intervalar de Moore, constituído pelo conjunto dos intervalos ordenado pela relação de inclusão. Este domínio é um Domínio de Scott Contínuo. Além disso, as funções contínuas para a Topologia de Scott no Domínio Intervalar de Moore são justamente aquelas que são monotônicas coma relação a inclusão para a topologia de Moore [Moo79].

A noção de monotonicidade com a relação a inclusão e continuidade segundo a topologia de Moore são compatíveis pela introdução de uma topologia auxiliar no Domínio Intervalar de Moore, denominada Topologia de Inclusão [Esc93]. A topologia de Inclusão é definida de tal forma que uma função intervalar é contínua para a topologia de inclusão se e somente se for monotônica para a inclusão. Considera-se a interseção entre a topologia da inclusão e a topologia de Moore, originando uma topologia denominada de Topologia de Moore-Inclusão. Uma função intervalar é contínua para a Topologia de Moore-inclusão se e somente se ela for monotônica para a inclusão e contínua para a topologia de Moore. Além disso, conclui-se que a Topologia de Moore-inclusão na verdade é Topologia de Scott. Isto mostra, a princípio que, neste sentido, as noções de monotonicidade para a inclusão e continuidade para a Topologia de Moore não são compatíveis, e que, portanto, a Topologia de Scott é a mais apropriada para a Análise Intervalar. A Topologia de Scott é quasi-metrizável segundo uma ordem, no sentido que os conjuntos abertos da topologia são gerados por uma quasi-métrica com o acréscimo de uma relação de ordem.

[Esc93] apresentam uma fundamentação para a Computação Científica clara e precisa. Pode-se aplicar a Teoria dos domínios de Scott para a Análise Intervalar. Pode-se concluir que a Análise Intervalar é uma fundamentação natural para a Análise Numérica e que a Teoria dos domínios de Scott é uma fundamentação natural para a Análise Intervalar.

5.2.3 Teoria dos Domínios Contínuos Intervalares

Em [Aci91] foi proposta uma topologia (Topologia de Scott) que procurou superar as deficiências e incompatibilidades da teoria clássica dos domínios contínuos trabalhando a idéia de aproximação que induz uma ordem de informação, ou seja, para quaisquer intervalos X e Y se $X \subseteq Y$, então, sobre um número real r, Y fornece mais (no mínimo igual) informações que X. Desta forma um número real é aproximado por intervalos de extremos racionais e não mais por intervalos de extremos reais. O real r é o supremo de uma cadeia de intervalos com extremos racionais encaixados, constituindo, então, um espaço denotado por $\mathbb{I}(\mathbb{Q})$.

Uma das principais diferenças, entre esta abordagem e a clássica, está no fato de que nesta a análise de intervalos não é uma extensão da análise real, mas uma linguagem sobre aproximações em análise real. Este ponto de vista tem a vantagem de ser de "lógica construtiva" e "computacional", além de unificar as teorias das semânticas de linguagens de programação e a Matemática Computacional (Análise Numérica). Ela provê uma lógica (lógica de Scott) para raciocinar sobre programas em Matemática Computacional.

Considerando $[p,q],[r,s] \in I(\mathbb{R})$ tal que $[p,q] \ll [r,s]$ se e somente se $p < r < s < q$, onde "$\ll$" é chamado de relação auxiliar ou ordem forte, tem-se que a completação por cortes de Dedekind [Aci91] [Dim91] de $\beta = \langle \mathbb{I}(\mathbb{Q}), \sqsubseteq, \ll, [-\infty,+\infty] \rangle$ constitui um cpo (conjunto completo parcialmente ordenado) contínuo $R = (\mathbb{I}(\mathbb{R}), \sqsubseteq, \ll, [-\infty,+\infty])$, onde "$\sqsubseteq$" é a ordem parcial tal que $[p,q] \sqsubseteq [r,s]$ se e somente se $[r,s] \subseteq [p,q]$ ($[p,q],[r,s]$ são considerados como conjunto e "$\subseteq$" é a relação de inclusão). O conjunto $\mathbb{I}(\mathbb{R})$ é isomorfo a $I(\mathbb{R})$ da teoria clássica. Os números Reais são os objetos totais do domínio R , cuja base é β .

A Teoria dos domínios tem sido amplamente empregada como modelos para semânticas de linguagens de programação, desde o estudo desenvolvido por Scott e Stratchey. Salienta-se o uso de noções topológicas subjacentes e da topologia de Scott como uma topologia ordem-consistente. A topologia de Scott permite abstrair o conjunto de propriedades dos objetos, independentemente de como eles são computados, ou seja, ela isola um certo conjunto enumerável de propriedades suficientemente finas para separar objetos distintos.

Existe uma relação direta entre topologias de Scoot e Espaços quasi-métricos

É possível introduzir uma topologia sobre o conjunto $I(\mathbb{R})$, que torna as operações aritméticas contínuas [Moo66].

Definição 44 *(Distância entre dois intervalos)*

Sejam $X = [a, b]$, $Y = [c, d]$ intervalos. Chama-se de **distância** *entre X e Y à expressão dada por:*

$$dist(X, Y) = max(|a - c|, |b - d|)$$

Facilmente observa-se que $dist(X, Y)$ satisfaz as propriedades de uma métrica.

$\forall X, Y, Z \in I(\mathbb{R})$. Portanto $(I(\mathbb{R}), dist)$ é um espaço métrico e consequentemente a topologia induzida dessa métrica é de Hausdorff.

Nota-se que a Teoria dos Intervalos baseia-se na idéia de que um intervalo é uma aproximação dos Reais que ele contém. Entretanto, a topologia que está sendo proposta (Hausdorff) é incompatível com essa idéia. Nos espaços topológicos de Hausdorff (como o espaço dos intervalos segundo a teoria clássica) todos os objetos são totais, significando que cada objeto do espaço somente aproxima ele mesmo. Verifica-se também que existe uma incompatibilidade entre a inclusão monotônica e a métrica proposta. A inclusão monotônica como está definida em [Moo66] sugere distinções qualitativas em função de uma ordem intuitiva de aproximação. Ora, esta ordem (de informação) induz uma topologia que não compatibiliza com a topologia (Hausdorff) induzida pela métrica de [Moo66].

5.3 O Quasi-Anel dos Intervalos Reais

Diante do exposto teórico sobre Intervalos é necessário fazermos uma análise quanto a estrutura algébrica que o conjunto dos Intervalos Reais formam. Se este formasse um anel, uma máquina BSS poderia trabalhar sob ele, porém como não satisfaz as condições de distributividade e associatividade dos anéis provaremos que satisfaz as condições de quasi-anel ordenado. Pretenderemos definir que uma máquina BSS extendida para trabalhar sobre quasi-anéis ordenados e sendo o Conjunto dos Intervalos Reais um quasi-anel ordenado, essa máquina BSS será capaz de trabalhar sobre ele.

Sendo assim, faz-se necessário a seguinte demonstração :

Proposição 7 $[I(\mathbb{R}), +, \cdot]$ *é um quasi-anel ordenado*

DEMONSTRAÇÃO:

Para demonstrarmos a citação acima temos que satisfazer as condições de quasi-anel ordenado. Seja $[I(\mathbb{R}), +, \cdot]$ uma estrutura algébrica binária definida as operações $+$ e $\cdot$ onde $I(\mathbb{R})$ é o conjunto dos intervalos em $\mathbb{R}$ e $a, b, c, d \in \mathbb{R}$.

Observação Nas demonstrações a seguir serão usadas as propriedades de corpo dos reais e as definições das operações aritméticas dos intervalos reais.

i.

$$\begin{aligned}[a,b]+[c,d] &= [a+c, b+d]\\ &= [c+a, d+b]\\ &= [c,d]+[a,b]\end{aligned}$$

aqui demonstramos ser verdadeira a propriedade da comutatividade nos intervalos reais

ii. Seja $x = [a,b]$ e $\mathbb{O} = [0,0]$ onde $x \in I(\mathbb{R})$ e 0 é o elemento neutro dos reais. Precisamos demonstrar que $x + 0 = 0 + x = x$.

$$\begin{aligned}x+\mathbb{O} &= [a,b]+[0,0]\\ &= [a+0, b+0]\\ &= [0+a, 0+b]\\ &= \mathbb{O}+x\\ &= [0+a, 0+b]\\ &= [a,b] = x\end{aligned}$$

Assim demonstramos a existência de um zero nos intervalos reais.

iii. Demonstraremos agora a existência de um $\imath \in I(\mathbb{R})$ tal que $x \cdot \imath = \imath \cdot x = x$. Seja $\imath = [1,1]$ onde 1 é a unidade (elemento neutro da multiplicação em $\mathbb{R}$):

$$\begin{aligned}x\cdot\imath &= [a,b]\cdot[1,1]\\ &= [min\{a\cdot 1, b\cdot 1, a\cdot 1, b\cdot 1\}, max\{a\cdot 1, b\cdot 1, a\cdot 1, b\cdot 1\}]\\ &= [min\{1\cdot a, 1\cdot b, 1\cdot a, 1\cdot b\}, max\{1\cdot a, 1\cdot b, 1\cdot a, 1\cdot b\}]\\ &= [1,1]\cdot[a,b]\\ &= \imath\cdot \text{x}\\ &= [min\{1\cdot a, 1\cdot b, 1\cdot a, 1\cdot b\}, max\{1\cdot a, 1\cdot b, 1\cdot a, 1\cdot b\}]\\ &= [min\{a,b\}, max\{a,b\}]\\ &= [a,b]\\ &= \text{x}\end{aligned}$$

iv. Nesta etapa demonstraremos que $x \cdot \mathbb{O} = \mathbb{O} \cdot x = \mathbb{O}$

$$\begin{aligned}
\mathrm{x} \cdot \mathbb{O} &= [a,b]\cdot[0,0]\\
&= [min\{a\cdot 0, b\cdot 0, a\cdot 0, b\cdot 0\}, max\{a\cdot 0, b\cdot 0, a\cdot 0, b\cdot 0\}]\\
&= [min\{0\cdot a, 0\cdot b, 0\cdot a, 0\cdot b\}, max\{0\cdot a, 0\cdot b, 0\cdot a, 0\cdot b\}]\\
&= [0,0]\cdot[a,b]\\
&= \mathbb{O}\cdot \mathrm{x}\\
&= [min\{0\cdot a, 0\cdot b, 0\cdot a, 0\cdot b\}, max\{0\cdot a, 0\cdot b, 0\cdot a, 0\cdot b\}]\\
&= [0,0] = \mathbb{O}
\end{aligned}$$

Através das demonstrações anteriores, provamos que $[I(\mathbb{R}), +, \cdot]$ é um quasi-anel; provaremos agora que é ordenado, para que seja é necessário primeiramente ser uma ordem e ser fracamente ordenado.

Seja $I(\mathbb{N}) = \{[a,b]/a,b \in \mathbb{N}, a \leq b\} \subseteq I(\mathbb{R})$

Sendo $x, y, z \in I(\mathbb{N})$ e $x = [a,b]$, $y = [c,d]$ e $z = [e,f]$ onde $a,b,c,d,e,f \in \mathbb{N}$ temos que:

v. Precisamos demonstrar que $x \leq x$, ora

$$\begin{aligned}
a &\leq a \text{ e } b \leq b\\
\rightarrow &\ [a,b] \leq [a,b]\\
\rightarrow &\ x \leq x
\end{aligned}$$

vi. $x \leq y$ e $y \leq x \rightarrow x = y$

$$\begin{aligned}
[a,b] \leq [c,d] \rightarrow a \leq c, b \leq d\\
[c,d] \leq [a,b] \rightarrow c \leq a, d \leq b
\end{aligned}$$

como

$$a \leq c \text{ e } c \leq a \rightarrow a = c$$

da mesma forma,

$$b \leq d \text{ e } d \leq b \rightarrow b = d$$

sendo assim:

$$\begin{aligned}
&[a,b] = [c,d]\\
&= x = y
\end{aligned}$$

vii. $x \leq y$ e $y \leq z \rightarrow x \leq z$

sendo $x \leq y$ e $y \leq z$ temos:

$$\begin{array}{l} [a,b] \leq [c,d] \\ a \leq c, b \leq d \\ [c,d] \leq [e,f] \\ c \leq e, d \leq \text{f} \end{array}$$

por transitividade em $\mathbb{R}$ podemos afirmar que

$$a \leq e \text{ e } b \leq f \rightarrow [a,b] \leq [e,f] \rightarrow x \leq z$$

Provamos assim que $I(\mathbb{R})$ é uma ordem, o próximo passo da nossa demonstração será provar ser fracamente ordenado.

viii. $x \leq y \rightarrow x + z = y + z$ ora, se $x \leq y$ temos:

$$\begin{array}{lll} [a,b] \leq [c,d] & \rightarrow a \leq c, b \leq d & \text{sabemos que} \\ & a + e \leq c + e, b + \text{f} \leq d + \text{f} & \\ & \rightarrow [a+e, b+f] \leq [c+e, d+f] & \\ & \rightarrow x + z \leq y + z & \end{array}$$

ix. $x \leq$ y $\rightarrow kx \leq ky$, onde $k \in I(\mathbb{R})$ e $k = [g,h]$

$$\begin{array}{ll} x \leq y \rightarrow [a,b] \leq [c,d] & \\ kx & = k[a,b] \\ & = [g,h][a,b] \\ & = [min\{ga,gb,ha,hb\}, max\{ga,gb,ha,hb\}] \quad \text{e,} \\ ky & = k[c,d] \\ & = [g,h][c,d] \\ & = [min\{gc,gd,hc,hd\}, max\{gc,gd,hc,hd\}] \end{array}$$

como $a \leq c, b \leq d$ e, evidentemente, $a \leq b, c \leq d$, podemos afirma que:

$$min\{ga,gb,ha,hb\} \leq \{gc,gd,hc,hd\}, \text{ e}$$
$$max\{ga,gb,ha,hb\} \leq max\{gc,gd,hc,hd\}$$

com isto, podemos concluir que:

$$[min\{ga, gb, ha, hb\}, max\{ga, gb, ha, hb\} \leq$$
$$[min\{gc, gd, hc, hd\}, max\{gc, gd, hc, hd\}]$$

$$k[a, b] \leq k[c, d]$$

$$kx \leq ky$$

Finalmente, provaremos agora que $I(\mathbb{R})$ é quasi-anel ordenado demonstrando que:

x. Se $\mathbb{O} \leq x \leq y$ e $z > \mathbb{O} \Rightarrow xz \leq yz$ e $zx \leq zy$, onde $\mathbb{O} = [0, 0]$

Ora, partindo da hipótese que $\mathbb{O}\ l \leq x \leq y$ e $z > \mathbb{O}$, provamos anteriormente que para $x, y, k \in \mathbb{R}$ e que sendo $x \leq y$ (como de fato é neste caso) vale que $kx \leq ky$, para qualquer k fazendo $k = z$ temos $zx \leq zy$, desta forma:

$$\begin{aligned}
& zx \leq zy \\
& \qquad [e, f][a, b] \leq [e, f][c, d] \\
& \qquad [min\{ea, eb, fa, fb\}, max\{ea, eb, fa, fb\}] \leq [min\{ec, ed, fc, fd\}, \quad max\{ec, ed, fc, fd\}] \\
& \qquad [min\{ea, be, af, bf\}, max\{ae, be, af, bf\}] \leq \{min\{ce, de, cf, df\}, \quad max\{ce, de, cf, df\}] \\
& \qquad [a, b][e, f] \leq [c, d][e, f] \\
& \qquad xz \leq yz
\end{aligned}$$

desta forma:

$$kx \leq ky \text{ e } xk \leq yk$$

Agora podemos afirmar que $I(\mathbb{R})$ é um quasi-anel ordenado ■

Capítulo 6

O Modelo BSS-Intervalar

6.1 Introdução

Considerando que as operações $+$ e $\cdot$ são computáveis em $\mathbb{R}$, pois BSS já provou isso para os anéis, pretendemos demonstrar que as operações aritméticas sob os $I(\mathbb{R})$ também são computáveis em máquinas BSS como uma respectiva extensão natural da computabilidade em $\mathbb{R}$. Vejamos a seguinte demonstração:

Seja, $a, b, c, d \in \mathbb{R}$ e $[a, b]$ e $[c, d] \in I(\mathbb{R})$

$$[a, b] + [c, d] = [a + c, b + d]$$

$a + c$ e $b + c$ são números reais logo computáveis.

$$\begin{aligned} [a, b] - [c, d] &= [a - c, b - d] \\ &= [a + -(c), b + (-d)] \\ &= [a, b] + [-c, -d] \end{aligned}$$

Provamos anteriormente que a soma entre intervalos reais é computável.

Necessitamos apenas demonstrar que as operações Max e Min da operação de multiplicação são computáveis, sendo assim, nosso problema se reduz a contrução de uma máquina BSS clássica que computa as operações em questão.

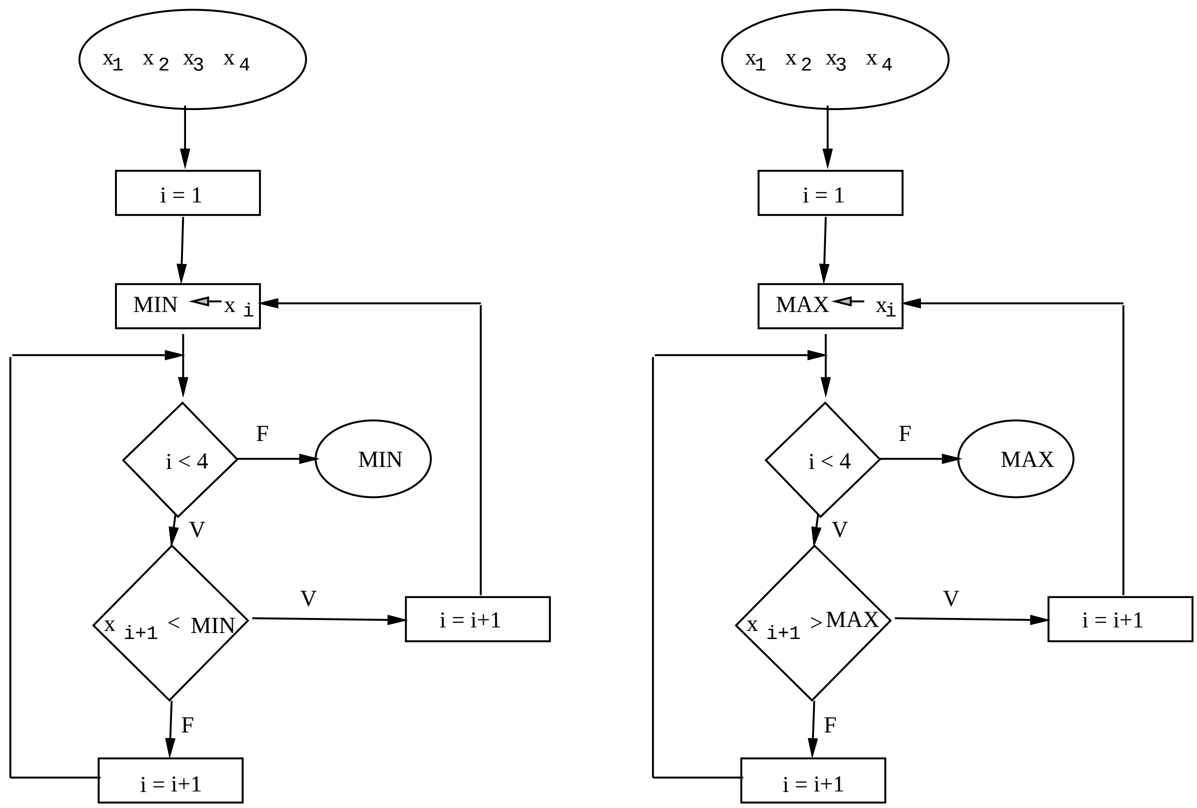

Figura 6.1: A esquerda temos a Máquina BSS que computa o mínimo entre quatro Números Reais e a direita a que computa o máximo

E finalmente, no caso da última operação $[a, b]/[c, d] = [a, b] \cdot [\frac{1}{d}, \frac{1}{c}]$ se $0 \notin [c, d]$ nota-se que a operação se reduz ao produto de dois intervalos reais que são computáveis como vimos na demonstração anterior. De fato, qualquer modelo X de computabilidade nos $\mathbb{R}$ podem ser transformado em um modelo de computabilidade, digamos X-intervalar para $I(\mathbb{R})$, da seguinte forma: uma função $f : I(\mathbb{R}) \to I(\mathbb{R})$ é computável no modelo X-intervalar se $\exists f_1 : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ e $f_2 : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ computáveis no modelo X tais que

$$f([a, b]) = [f_1(a, b), f_2(a, b)].$$

Como a noção de computabilidade sobre os reais é complexa, a computabilidade assim obtida resultará ser mais complexa, uma vez que não é interna (direta). Assim análises de computabilidade de uma função tal como a integração intervalar sera “nebulosa”. O problema com esta abordagem é que ela não é uma noção interna de computabilidade intervalar e, portanto, a análise de funções intervalares intuitivamente computáveis dependerá da computabilidade de duas funções reais, provavelmente, menos intuitivas. Assim, neste capítulo desenvolveremos uma teoria da computabilidade interna, baseada no modelo BSS, sobre os intervalos reais.

6.2 Extendendo Máquinas BSS Para Computações Intervalares

A máquina BSS Intervalar segue o modelo: Seja A um quasi-anel, $m, n \in \mathbb{N}$ e sejam $I = R^m$, $O = R^n$ e $S = \mathbb{Z}^+ \times \mathbb{Z}^+ \times R^\infty$, os espaços das entradas, saídas e dos estados de possíveis configurações, respectivamente. Uma máquina M sob A é basicamente uma máquina de acesso randômico clássica tendo uma quantidade infinita de registradores, cada um com a capacidade de armazenar um elemento de R. A máquina toma uma entrada de R^m , da alguma saída de R^n e pode realizar a cada passo uma computação polinomial ou racional envolvendo somente um número finito de registradores. A máquina pode realizar comparações simples e executar uma instrução de pulo dependendo do resultado da comparação. Os dois registradores inteiros adicionais são adicionados para permitir computação uniforme no caso que o espaço das entradas seja R^∞ [Bol95].

Assim a definição de Máquinas BSS-Intervalar é análoga à definição do modelo BSS clássico, só que nesta nova máquina substituímos a condição de se trabalhar sobre anéis ordenados por quasi-anéis. Esta alteração não trazerá mudanças no resto da definição da máquina, pois as propriedades imprescindíveis de anéis para máquinas BSS estão também presente em quasi-anéis. Como pela proposição 1 todo anel é um quasi-anel, e, pela proposição 2 todo anel ordenado é um quasi-anel ordenado, podemos concluir que todas as máquinas BSS clássicas também são máquinas do nosso novo modelo. Assim, tudo o que pode ser computado por máquinas BSS também podem ser computados por máquinas BSS-Intervalar. Além disso, existem estruturas que são quasi-anéis mas não são anéis, por exemplo os Intervalos Reais munidas das operações de adição $+$ e produto $\cdot$. Portanto máquinas BSS-Intervalar fazem computações que não são possíveis de serem efetuadas usando máquinas BSS clássicas.

Observemos agora o seguinte exemplo:

Exemplo 25 *Uma Máquina BSS-Intervalar calculando o valor absoluto de um Intervalo Real pode ser vista na figura 6.2*

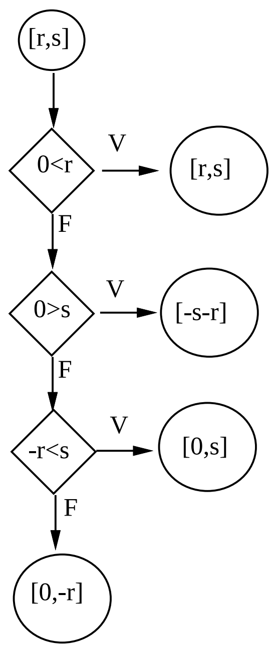

Figura 6.2: Diagrama de fluxo de uma Máquina BSS-Intervalar que Computa o Valor Absoluto de um Intervalo

6.3 O Método de Newton Intervalar

O método de Newton real é um algoritmo que serve para calcular a raiz de uma dada equação, através da construção de uma seqüência convergente de pontos da reta real. De maneira análoga, a versão intervalar do Método de Newton permite contruir uma seqüência convergente de intervalos, cujo limite será um intervalo que contém a raiz real da função dada.

Por ser um método autovalidável, se tomarmos um intervalo inicial que não contenha a raiz real, então, numa dada iteração, iremos obter um intervalo vazio como resultado. Caso contrário, se tomarmos um intervalo inicial, que contém a raiz real da equação $f(x) = 0$, e, considerando que a seqüência intervalar que se obtém é de intervalos encaixados, então obteremos como limite o intervalo de menor diâmetro possível, que ainda contém a raiz real desejada. Na prática, esta é a vantagem do uso do Método de Newton Intervalar.

Veremos agora, como se pode definir uma versão intervalar para o Método de Newton real:

A primeira idéia que se tem para definir o Método de Newton Intervalar é tomar uma extensão intervalar para o operador Newtoniano real, ou seja, definir $N(X) =$

$X - \frac{F(X)}{F'(X)}$, onde $F(X)$ e $F'(X)$ são extensões intervalares para as funções $f(x)$ e $f'(x)$ (função real contínua e derivada contínua, respectivamente).

Agora, veremos porque o método de Newton construído dessa maneira é sempre divergente.

Por exemplo, para calcularmos a raiz $\sqrt{2}$ da função $f(x) = x^2 - 2 = 0$ no intervalo $[1, 2]$; temos:

$$F(X) = X^2 - [2, 2] \text{ e } F'(X) = [2, 2] \cdot X.$$

Tomando $X_0 = [1, 2]$, calculamos

$$\begin{aligned}
X_1 &= X_0 - \frac{F(X_0)}{F'(X_0)} \\
&= [1, 2] - \frac{[1,2]^2 - [2,2]}{[2,2] \cdot [1,2]} = [1, 2] - \frac{[1,4] - [2,2]}{[2,4]} \\
&= [1, 2] - \frac{[-1,2]}{2,4} \\
&= [1, 2] - [-1, 2] \cdot [\tfrac{1}{4}, \tfrac{1}{2}] \\
&= [1, 2] - [-\tfrac{-1}{2}, 1] \\
&= [1, 2] + [-1, \tfrac{1}{2}] \\
&= [0, \tfrac{5}{2}]
\end{aligned}$$

Observe que:

$$diam(X_1) = diam([0, \tfrac{5}{2}]) = \tfrac{5}{2} > 1 = diam(X_0);$$

e que $X_0 \subseteq X_1$. Observe também que:

$$F'(X_1) = [2, 2] \cdot X_1 = [2, 2] \cdot [0, \tfrac{5}{2}] = [0, 5]$$

contém o zero, logo, na próxima interação (cálculo de X_2), não poderemos efetuar a operação de divisão.

Como queremos que a seqüência de intervalos X_k convirja para o intervalo pontual $[x_*, x_*]$ (onde x_* é a raiz), é necessário que $diam(X_k) \to 0$, quando $k \to +\infty$. Mas como,

$$diam(X_{k+1}) = diam(X_k - \tfrac{F(X_k)}{F'(X_k)}) = diam(X_k) + diam(\tfrac{F(X_k)}{F'(X_k))} > diam(X_k),$$

conclui-se que a seqüência de intervalos definida por $X_{k+1} = X_k - \frac{F(X_k)}{F'(X_k)}$ é sempre divergente.

Assim, devemos modificar o método de maneira que o diâmetro de um intervalo recém calculado X_{k+1} diminua em relação ao diâmetro de X_k e que ainda contenha x_*. A idéia agora é construirmos uma seqüência de intervalos encaixados $X_0 \supseteq X_1 \supseteq X_2 \supseteq X_3 \supseteq X_4 \supseteq \cdots \supseteq X_k \supseteq \cdots \supseteq [x_*, x_*]$, cujos diâmetros vão diminuindo à medida que o valor de k aumenta. Para tanto, vamos avaliar com intervalos somente a derivada, cuidando para que o intervalo reultante da avaliação intervalar da derivada não contenha o zero.

Desta forma, precisamos exigir que a função f satisfação certas condições no intervalo inicial X_0, ou seja, para que a avaliação intervalar da derivada não contenha o zero, é necessário que a função f não tenha pontos críticos (máximos ou mínimos locais) em X_0. Veremos, a seguir, como definir o Método de Newton Intervalar que sempre é convergente.

Teorema 13 *Seja $f(x)$ uma função real contínua em $X_0 = [a, b]$, de modo que $f(a) \cdot f(b) < 0$. Assim, existe $x_* \in X_0$ tal que $f(x_*) = 0$. Calculamos um intervalo $M = [m_1, m_2]$, tal que $0 < m_1 \leq \frac{f(x)-f(y)}{x-y} \leq m_2 < +\infty$, para todo $x, y \in X_0$, com $x \neq y$, que servirá como uma inclusão intervalar para a imagem intervalar da derivada $f'(x)$ da função $f(x)$.*

Finalmente definindo o **Operador Newtoniano Intervalar** *por:*

$$N(X) = med(X) - \frac{f(med(X))}{M},$$

constuímos a seqüência intervalar recursiva:

$$X_{k+1} = X_k \cap N(X_k) \text{ e } X_0 = [a, b], \text{ que tem as seguintes propriedades:}$$

1. $x_* \in X_k, \forall k \geq 0$
2. $X_0 \supseteq X_1 \supseteq X_2 \supseteq X_3 \supseteq \cdots e \; lim_{k \to +\infty} X_k = [x_*, x_*]$
3. $diam(X_{k+1}) \leq (1 - \frac{m_1}{m_2}) \cdot diam(X_k)$, *ou seja,* $diam(X_{k+1}) \leq (1 - \frac{m_1}{m_2})^k \cdot diam(X_0)$ *o que implica que* $lim_{k \to +infty} diam(X_k) = 0$
4. *Se existe $k_0 \geq O$ tal que $X_{k_0} \cap N(X_{k_0}) = \emptyset$, então não existe raiz da função f no intervalo inicial X_0.*

A última condição, garante a autovalidação do algoritmo, ou seja, se escolhermos um intervalo inicial que não contém raiz da função, então o método pára, caso contrário, ele calcula o intervalo limite da seqüência X_k, que contém x_*, que é a raiz da função f, validando a resposta.

Observações

1. $x_* \in x - \frac{f(x)}{M}, \forall x \in X_0$.
2. Podemos tomar $M_1 = Im(\frac{f(x)}{x - x_*}, X_0) \subseteq M$ ou ainda $M_2 = [m_1, m_2] = Im(f'(x), X_0) \subseteq M$.

Exemplo 26 *Calcular a raiz da função* $f(x) = x^2 - 2 = 0$ *em* $X_0 = [1, 2]$

$f(x) = x^2 - 2$ é uma função contínua em $\mathbb{R}$, logo é contınua no intervalo inicial $X_0 = [1, 2]$.

Temos que:

$$f(1).f(2) = (1^2 - 2) \cdot (2^2 - 2) = (-1) \cdot (2) = -2 < 0,$$

logo a função tem pelo menos uma raiz no intervalo $[1, 2]$.

O cálculo da inclusão intervalar M para a derivada pode ser feito por:

$$\begin{aligned} \tfrac{f(x)-f(y)}{x-y} &= \tfrac{(x^2-2)-(y^2-2)}{x-y} \\ &= \tfrac{x^2-y_2}{x-y} \\ &= \tfrac{(x-y)\cdot(x+y)}{x-y} \\ &= x + y \end{aligned}$$

e daí, temos que:

$$0 < 2 \leq \tfrac{f(x)-f(y)}{x-y} = x + y \leq 4,$$

para quaisquer $x, y \in X_0 = [1, 2]$, pois $1 \leq 2$ e $1 \leq y \leq 2$ logo, $1 + 1 \leq x + y \leq 2 + 2$.

Como o cálculo da derivada $f'(x)$ da função $f(x)$ é, este caso, mais fácil, podemos tomar:

$$\begin{aligned} M &= [m_1, m_2] \\ &= Im(f'(x), X_0) \\ &= Im(2 \cdot x, [1, 2]) \\ &= [min\{2 \cdot x/x \in [1, 2]\}, max\{2 \cdot x/x \in [1, 2]\}] \\ &= [2, 4] \end{aligned}$$

Assim, percebemos que as duas inclusões que foram calculadas para a derivada intervalar coincidem.

O Operador Intervalar Newtoniano fica, então:

$$N(X) = [med(X), med(X)] - \tfrac{[(med(X))^2-2,(med(X))^2-2]}{[2,4]}$$

Finalmente, construímos a seqüência intervalar recursiva

$$X_{k+1} = X_k \cap N(X_k) \text{ e } X_0 = [1, 2]$$

Computando os valores da seqüência, temos que o valor da raiz da função $f(x) = x^2 - 2$ é um valor real $x_* \in [1.414213562373094, 1.414213562373096]$

6.3.1 Máquina BSS Intervalar Calculando a Raiz de uma Função pelo Método de Newton

Vejamos a seguir um modelo BSS Intervalar que computa o exemplo anterior:

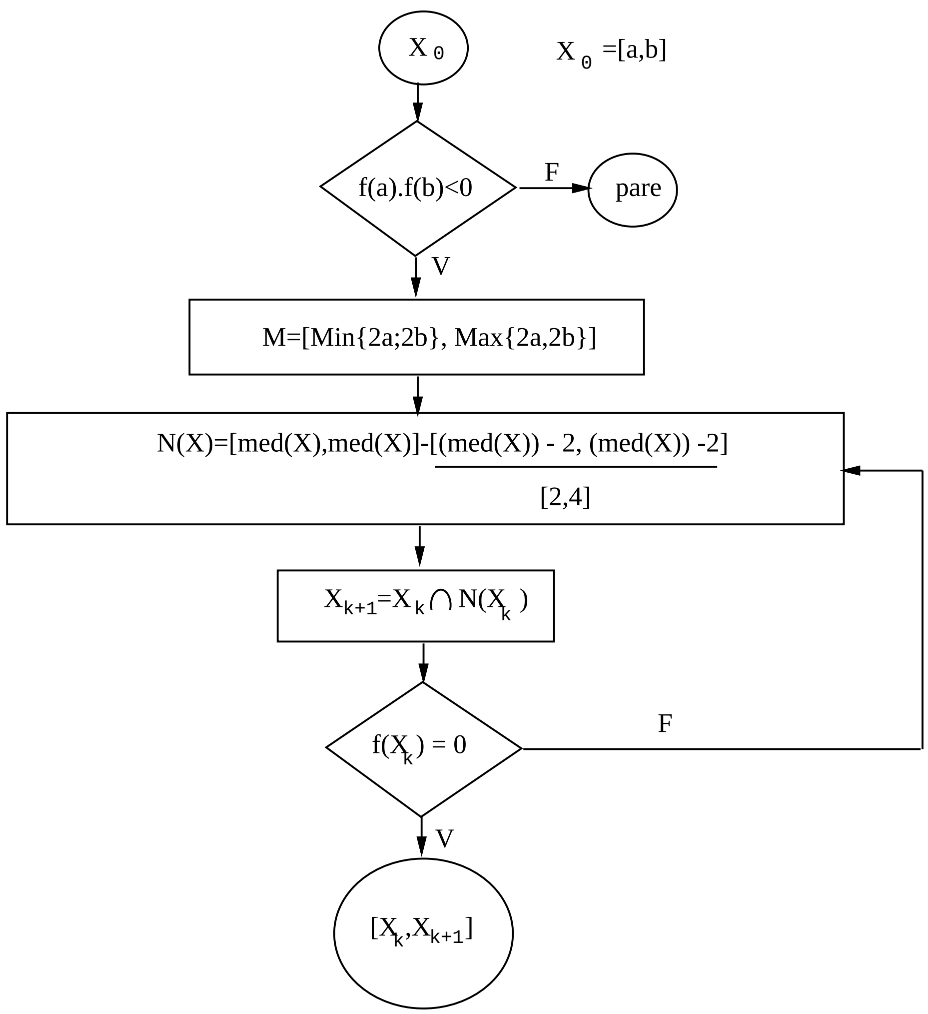

Figura 6.3: Máquina BSS Intervalar que Calcula as Raízes da função $f(x) = x^2 - 2$

6.4 Caracterização das funções Intervalares BSS Computáveis

O quadro a seguir apresenta uma caracterização das funções computáveis no espírito das funções parciais recursivas:

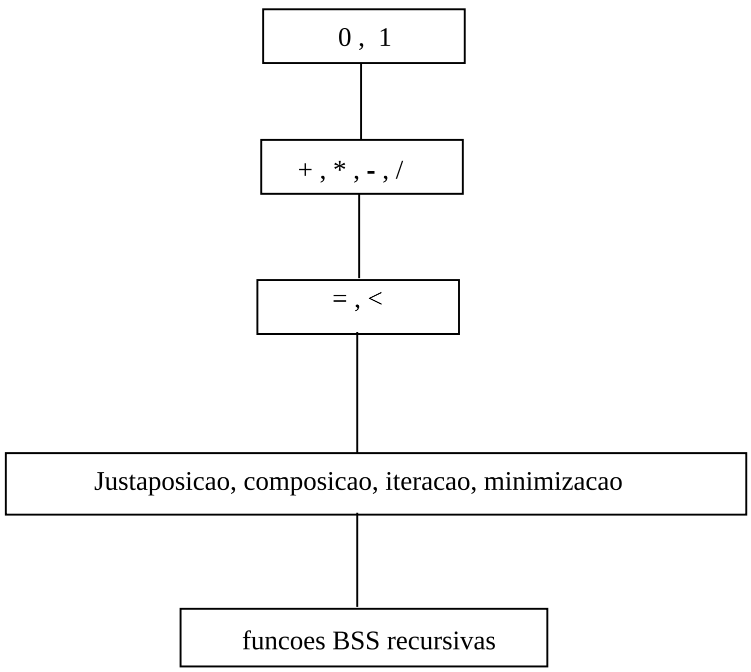

Figura 6.4: Caracterização das Funções BSS Computáveis

Definiremos informalmente a justaposição, composição e iteração :

Justaposição

O operador de **justaposição** corresponde à intuição que uma relação (f, g) é computável se e somente se cada componente é computável. Na teoria clássica este operador não é considerado explicitamente porque há funções de pareamento bijetiva computável de $\mathbb{N}^2 \to \mathbb{N}$. No caso dos números reais não há função contínua e injetiva de $\mathbb{R}^2 \to \mathbb{R}$. Portanto a justaposição será útil neste domínio. Outra vantagem é que na presença do operador de justaposição o operador de substituição clássica pode ser substituído pelo operador de composição mais facilmente e o operador de recursão primitiva pode ser substituído pelo operador de repetição (contanto que as projeções estejam disponíveis).

Composição

A **composição** foi considerada em semântica de um modo semelhante. O "tudo ou nada" condição no domínio que garante propriedades de fechamento razoáveis, enquanto a pura composição com $dom(g \circ f) := \{x \in X/f(x) \cap dom(g) \neq \emptyset\}$ nem sempre preserva a continuidade. Se f é uma função então escreveremos para $gf := g \odot f$.

Iteração

A iteração é uma generalização da composição.

6.5 Continuidade das Funções BSS-Computáveis

É praticamente um consenso nas aproximações de computabilidade sobre os números reais, que toda função computável seja contínua. Assim, é de se esperar que essa propriedade também seja satisfeita em modelos de computabilidade para os intervalos reais. De fato, em [BA97b], foi mostrado a continuidade, com respecto a uma topologia quasi-métrica, das funções intervalares computáveis no modelo por eles introduzido.

Moore propõe uma métrica sobre os intervalos reais de modo que a métrica usual para os reais seja uma restrição da métrica de Moore para os intervalos degenerados. Tal métrica é definida por:

$$d([r,s],[t,u]) = max\{|r-t|,|s-u|\}.$$

A topologia usual induzida nos intervalos reais pela métrica d denominou-se **Topologia de Moore**. Assim, a reta é "isometricamente" mergulhada nesse espaço topológico.

Segundo Bedregal e Aciòly [BA97b] as funções aritméticas intervalares são computáveis, eles demonstraram que *"cada função intervalar computável é contínua segundo a topologia de Scott sobre $I(\mathbb{R})$"*.

Eles apresentaram uma quasi-métrica q sobre $I(\mathbb{R})$ ($q : I(\mathbb{R}) \times I(\mathbb{R}) \to \mathbb{R}^+$), onde q é $< I(\mathbb{R}), \Omega_S(R) >$ onde $\Omega_S(R)$ é a topologia de Scott sobre o domínio contínuo R e que o espaço topológico $< I(\mathbb{R}), \Omega_S(R) >$ é quasi-metrizável.

A métrica q^* associada ao espaço q é definida por:

$$\begin{aligned} q^*([r,s],[t,u] &= max\{q([r,s],[t,u]), q([t,u],[r,s]), 0)\} \\ &= max\{t-r, s-u, r-t, u-s, 0\} \\ &= max\{|r-t|, |u-s|, 0\} \\ &= max\{|r-t|, |u-s|\} \end{aligned}$$

Eles demonstraram também que a teoria dos $I(\mathbb{R})$ é um espaço quasi-métrico. Sendo $\langle I(\mathbb{R})$ x $I(\mathbb{R}), q\rangle$ espaço quasi-métrico as operações aritméticas intervalares são uniformemente contínuas.

O espaço quasi-métrico q associado à métrica q^*, induz a topologia de Scott e, por outro lado, sua métrica associada, q^*, é a métrica de Moore. Essa nova topologia, estudada por Bedregal e Acióly, se sobrepõe a de Scott e a de Moore no sentido de utilizar uma quasi-métrica que induz a topologia Scott e que portanto é compatível quanto a inclusão monoticidade (não compatível em Moore) e que por outro lado sua métrica associada é a métrica de Moore [AB97]. Portanto, a topologia de Scott sobre $I(\mathbb{R})$ se relaciona com a topologia tradicional de Moore, no sentido que ambas podem ser obtidas canonicamente desde a quasi-métrica q.

Isto resolve o problema fundamental em relação aos Intervalos Reais sobre a Topologia de Moore pois eles não são compatíveis com a inclusão monotonicidade, uma vez que existem funções contínuas que, segundo essa topologia, não são monotônicas [Aci91, CEF 92].

Para demonstrar que todas as funções intervalares computáveis em nosso modelo são continuas na topologia quasi-métrica, deveriamos mostrar que as funções aritméticas são contínuas, que a igualdade e ordem estrita também são contínuas e, finalmente, que a justaposição, composição e minimização de funções contínuas são contínuas.

Em nosso modelo BSS-intervalar, podemos afirmar que as funções aritméticas intervalares são computáveis em máquinas BSS intervalares (como já demonstramos anteriormente) e são contínuas na topologia quasi-métrica.

Vale observar que a divisão entre intervalos é uma função computável parcial por não ser definida para intervalos que contenha o 0. Sendo assim, definiremos esta função como sendo parcialmente contínua (contínua em qualquer intervalo que não esteja contido o zero).

Por outro lado, não podemos demonstrar facilmente que a igualdade seja contínua, prentendemos fazer isto em futuros trabalhos. A demonstração de que a justaposição, composição, iteração e minimização preserva continuidade é ligeiramente mais simples, mas como não adiantaria de nada sem termos a continuidade da igualdade, decidimos realizar esta demonstração só quando tivermos demonstrado a contínuidade da igualdade.

Capítulo 7

Conclusão

A computação científica é uma área relativamente nova, mas que vem crescendo consideravelmente como conseqüência, inclusive, da sua aplicação em diversas outras ciências. Atualmente tem sido muito divulgada a necessidade do uso de técnicas intervalares com finalidade de alcançar limites garantidos para os resultados de computações científicas, através do controle rigoroso e automático do erro do resultado.

Neste enfoque, a Matemática Intervalar consolida-se como uma das soluções para o problema do controle do erro em Computação Científica. Salientam-se os trabalhos pioneiros de Moore [Moo66], que introduziu a Teoria Clássica dos Intervalos, assim como as diversas extensões à teoria original, que visam garantir maior aplicabilidade.

Com relação à fundamentação, entretanto, verifica-se que a Computação Científica e a Teoria dos Intervalos ainda apresentam problemas, embora diversas alternativas tenhas sido apresentadas. Todas essas teorias têm embasamento na teoria de Domínio de Scott, e, portanto, se apresentam como uma variação daquela teoria. Tendo analisado diversos trabalhos nesse sentido, concluímos que o caminho à fundamentação ainda está aberto a novas idéias e teorias, dentre elas, o Modelo BSS intervalar.

A partir do trabalho desenvolvido em 1986 por Blum, Shub e Smale [BSS89] onde foi definido uma máquina que trabalha sobre um anel ordenado arbitrário e denominada de modelo BSS, desenvolvemos a nossa dissertação extendendo este modelo para um modelo que trabalha sob a Matemática Intervalar. A estrutura algébrica que suporta os algorítmos intervalares em computação científica é apenas um quasi-anel (uma estrutura parecida com um anel). Essa estrutura deve, também, ser considerada quando se deseja realizar uma fundamentação para a computação científica.

A nossa dissertação adaptou o modelo BSS para intervalos, para isto generalizamos os conceitos básicos das máquinas. Por exemplo, como o espaço dos intervalos não é um anel e sim um quasi-anel analisamos quais as implicações de se trabalhar com esta estrutura. É claro que tivemos certos cuidados, como por exemplo, garantir que quando a estrutura for um anel esta máquina se comporte da mesma maneira que BSS e realizamos uma análise que relaciona a teoria da computabilidade resultante com as propriedades topológicas.

Uma questão fundamental que analisamos neste trabalho foi as conseqüências de se atribuir uma estrutura de quasi-anel em substituição a um anel conforme propuseram BSS em sua máquina. Esta modificação não causou nenhum efeito, pois a única parte de BSS que usa propriedades de anel são os nó computacionais, mas, as propriedades que são usadas são também propriedades de quasi-anel.

Por outro lado provamos que os intervalos reais são quasi-anéis ordenados, que sendo a máquina BSS capaz de manipular com quasi-anéis também será capaz de trabalhar sobre os intervalos reais.

Desta forma, nosso modelo melhor se adequa a Computação Científica, pois, podemos concluir que toda as máquinas BSS clássicas também são máquinas do nosso modelo, desta forma, tudo que o modelo BSS computa, o modelo BSS intervalar também computará e mais estruturas intervalares que não são anéis, são quasi-anéis. Portanto nosso modelo realiza computações que não são possíveis de serem realizadas no modelo clássico.

Esperamos com isto, ter contribuído para o desenvolvimento da Computação Científica no sentido de ser mais um meio de resolução de seus problemas.

7.1 Futuros Trabalhos

Essa dissertação abre um grande leque para que futuros trabalhos sejam apresentados dando continuidade a este novo modelo.

Estes trabalhos poderão ser frutos do aprofundamento da própria teoria, como um estudo mais aprofundado da topologia, uma análise da complexidade do Modelo BSS Intervalar, ou a aplicação aos problemas já existente na computação científica (problemas ocasionados por truncamentos de seqüências de operações e arredondamentos de números com grande quantidade de dígitos por outros menores devido a capacidade da máquina).

Podemos observar nessa dissertação que toda função contínua é computável. Apesar de não termos demonstrado, esta afirmação é factível, e servirá de sugestão para futuros estudos.

Pretendemos também comparar o nosso modelo com outros modelos intervalares existentes, tais como os modelos propostos por Bedregal e Acióly [BA97b] e Edalat[Eda97].

É interesse nosso continuar estudando esses modelo em todas as suas áreas contribuindo desta forma para o avanço da Computação Científica.

Bibliografia

[AC92] Benedito Melo Acióly e Dalcídio Moraes Cláudio. *Um Misto de Visão Clássica e Moderna de Topologia*. Technical Report RP-191, UFRGS-CPGCC - II - Departamento de informática Teórica, Porto Alegre, 1991.

[Aci91] Benedito Melo Acióly. *Computational Foundation of Interval Mathematics* (in portuguese). PhD thesis, CPGCC da UFRGS, Porto Alegre, 1991.

[BA93] Benjamín René Callejas Bedregal and Benedito Melo Acióly. Rational intervals: An effectively given information system for the real number. In S. M Markov, editor, *Scientific Computation and Mathematica Modelling*, pages 159-169, Sofia-Bulgaria, 1993. DATECS Publishing.

[BA97b] Benjamin René Callejas Bedregal and Benedito Melo Acióly. *Computability on the Interval Space: A domain approach*. XXIII Conferência Latinoamericana de Informática-CLEI, Valparaíso-Chile, Novembro de 1997.

[Bar88] M. Barnsley. *Fractals everywhere*. Academic Press Inc., London, 1988.

[Bec86] E. Becker. On the real spectrum of a ring and its application to semi-algebraic geometry. *Bulletin of the American Mathematics Society*, 15:19-60, 1986.

[Bed96] Benjamín R. Callejas Bedregal. *Continuous Information Systems: A Computational and Logical Approach to Interval Mathematics* (in portuguese). PhD thesis, UFPE-Depto. de Informática, Recife, 1996.

[BL74] Walter S. Brainerd and Lawrence H. Landweber. *Theory of Computation*. John Wiley & Sons, U.S.A., 1974.

[Bol95] Paolo Boldi. *Computability and complexity over the reals*. University of Milan, Italy, February, 1995.

[Bra95] Vasco Brattka. *Recursive Characterization of Computable Real-valued Functions and Relations*. Preprint submitted to Elsiever Science, 1995.

[BSS89] L. Blum, M. Shub and S. Smale. *On a Theory of Computation and Complexity over Real Number: NP-completness, recursive functions and universal machines*. Bull. of the Amer. Math. Soc. 21, 1989, pg. 1-46.

[BSSC95] L. Blum, M. Shub, S. Smale and Felipe Cucker. *Complexity and Real Computation: A Manifesto*. 1995.

[Car94] Carl H. Smith. *A Recursive Introduction to the Theory of Computation*. Springer-Verlag. 1994.

[CEF92] Dalcídio M. Cláudio, Martín H. Escardó, and Beatriz R. T. Franciosi. An order-theoretic approach to interval analysis. *Interval Computation*, 5(3):38-45, 1992. Special Issue: Proceeding of the conference INTERVAL'92, Moscow, September 22-25, 1992.

[Cho90] Daniel Cohen I. A. *Introduction to Computer Theory*. Singapore. 1990.

[Cla96] Dalcídio M. Cláudio, et al. *Introdução a Teoria dos Intervalos*. Escola de Inverno de Matemática Aplicada e Computacional. Porto Alegre/RS. P.217-244.1996.

[Cut97] Nigel. Computability. *An introduction to recursive funtion theory*. Department of Pure Mathematics, University of Hull. New York. USA. 1997.

[Dev95] Robert L. Devaney. *Chaotic Dynamical Systems*. Second Edition. USA. 1995.

[Dev96] Robert L. Devaney. *A first Course in Chaotic Dynamical Systems*. Theory and Experiment . USA. 1996.

[Dim91] Graçaliz Pereira Dimuro. *Domínios intervalares da matemática computacional*. Master's thesis, PGCC da UFRGS, Porto Alegre, 1991.

[Dim97] Graçaliz Pereira Dimuro. *Uma Representação Computacional Global para Sistemas Ordenados de 2. Ordem em Espaços Coerentes Intervalares Bi-Estruturados, com Aplicação em Matemática Intervalar*, Tese de Doutoramento, Porto Alegre-RS, Brasil, Junho de 1997.

[DNC95] Taraju Asmuz Diverio, Philippe Olivier A. Navaux, and Dalcidio Moraes Claudio. Alto desempenho e eficiência em processamento numerico. In *Anais 7. Simpósio Brasileiro de Arquitetura de Computadores - Processamento de Alto Desempenho*, pages 257-270, Canela-RS, Brasil, 29 de julho a 1. de agosto de 1995.

[EZ94] O. Ermakov and V. Zyuzin. The construction on two-sided approximations for a solution of ODE systems with the use of taylor series. In *Abstracts International Conference on Interval and Computer-Algebraic Methods in Science and Engineering, INTERVAL'94*, page 87, St. Petersburg, Russia, 1994. International Journal Interval Computations.

[Gar94] Z. Garczarczyk. An efficient evaluation the range of functions and its application in the nonlinear circuit analysis. In *Abstracts International Conference on Interval and Computer-Algebraic Methods in Science and Engineering, INTERVAL'94*, pages 89-92, St. Petersburg, Russia, 1994. International Journal Interval Computations.

[GH75] Griffiths and Hilton. *Matemática Clássica, uma interpretação contemporânea* Universidade de São Paulo.São Paulo. Editora Edgard Blucher Ltda.1975.

[Gia93] Pietro Di Gianantonio. *A Functional Approach to Computability on Real Numbers*. PhD thesis, Università di Pisa-Genova-Udine, Italy, march 1993.

[Grz57] A. Grzegorczyk. *On the definitions of computable real continuous functions*. Fundamenta Mathematicae, 44(1), 1957.

[GV94] Tatiana D. Golubkova and Alexander P. Voschinin. Some applications of interval regression analysis in biometrics. In *Abstracts International Conference on Interval and Computer-Algebraic Methods in Science and Engineering, INTERVAL'94*, pages 96-98, St. Petersburg, Russia, 1994. International Journal Interval Computations.

[Kor92] Alexander V. Korlyukov. *A* new application of interval mathematics. *Interval Computations*, 3(5):116-121, 1992.

[Lan72] Sérgio Lang. *Estruturas Algébricas*. Universidade de Columbia. Nova Iorque. 1972.

[Mcc95] J. L. McCauley. *Chaos, Dynamics and Fractals an algorithmic approach to deterministic chaos*. Cambridge Nonlinear Science Series. Department of Physics, University of Houston. 1995.

[ML70] Per Martin-Löf. *Notes on Contructive Mathematics*. Almqvist & Wiksell, Stockholm, 1970.

[Moo66] R. E. Moore. *Interval Analysis*. Englewoods Cliffs: Prentice Hall, 1966.

[Orl92] Alexander I. Orlov. Interval statistics. *Interval Computations*, 1(3):44-52, 1992.

[Sco70] Dana S. Scott. Outline of a mathematical theory of computation. In *4 th. annu Princeton Conference on Inf. Science and Systems*, pages 65-106, 1970.

[Sco72] Dana S. Scott. Continuous lattices. In *Toposes, Algebraic Thory and Logic*, pages 97-136, Berlim, 1972. Lectures Notes in Mathematics, Vol. 274, Springer Verlag.

[SS93] Carol A. Schnepper and Mark A. Stadther. Applications of a paralel interval Newton/generalized bisection algorithm to equations-based chemical process flowsheeting. *Interval Computations*, (4):40-64, 1993.

[VV94] A. A. Vakhidov and N. N. Vasiliev. Accuracy checking of analytical theory of artificial satellite motion. In *Abstracts International Conference on Interval and Computer-Algebraic Methods in Science and Engineering, INTERVAL'94*, pages 241-242, St. Peterburg, Russia, 1994. International Journal Interval Computations.