Adriano Xavier Carvalho

SISTEMA DE REESCRITA DE TERMOS PARA INTERVALOS: EM DIREÇÃO A UM MODELO FORMAL PARA A COMPUTAÇÃO INTERVALAR

Natal
Outubro de 2005

Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada
Programa de Pós-Graduação em Sistemas e Computação

SISTEMA DE REESCRITA DE TERMOS PARA INTERVALOS: EM DIREÇÃO A UM MODELO FORMAL PARA A COMPUTAÇÃO INTERVALAR

Dissertação submetida ao Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte como parte dos requisitos para a obtenção do grau de Mestre em Sistemas e Computação (MSc.).

Adriano Xavier Carvalho

Natal, Outubro de 2005

SISTEMA DE REESCRITA DE TERMOS PARA INTERVALOS: EM DIREÇÃO A UM MODELO FORMAL PARA A COMPUTAÇÃO INTERVALAR

Adriano Xavier Carvalho

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Sistemas e Computação (MSc.), Área de Concentração em Teoria e Inteligência Computacional, e aprovada em sua forma final pelo Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte.

Regivan Hugo Nunes Santiago
Orientador

Regivan Hugo Nunes Santiago
Coordenador do Programa

Banca Examinadora:

Regivan Hugo Nunes Santiago
Presidente

Anamaria Martins Moreira

Benjamín René Callejas Bedregal

Edward Hermann Haeusler

CARVALHO, Adriano Xavier

Sistema de Reescrita de Termos para Intervalos: Em Direção a Um Modelo Formal para a Computação Intervalar / Adriano Xavier Carvalho. Natal: 2005.

107p. : 1.

Orientador: Regivan Hugo Nunes Santiago

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Norte. Centro de Ciências Exatas e da Terra. Departamento de Informática e Matemática Aplicada.

1. Teoria e Inteligência Computacional – Tese. 2. Sistemas de Reescrita. 3. Computação Intervalar. 4. Lógica Equacional. I. Sistema de Reescrita de Termos para Intervalos: Em Direção a Um Modelo Formal para a Computação Intervalar.

Demais, filho meu, atenta: Não há limite para fazer livros, e o muito estudar é enfado da carne. (Ec 12:12)

Aos meus pais, Serafim e Rosimar.
Aos meus irmãos, Alessiane, Alexley e Alessandro.
E aos amigos da Tecnomed, Eduardo Henrique, Ibsen e Marize.

AGRADECIMENTOS

Expresso aqui meu mais profundo sentimento de gratidão a todas as pessoas que colaboraram, de uma forma ou de outra, para a concretização desta dissertação. O apoio, quase sempre anônimo, destas pessoas foi certamente imprescindível. Como não sou capaz de citar a todas, nominalmente, prefiro deixar que elas continuem assim, anônimas. Apenas um nome eu não gostaria de deixar anônimo, o de Jesus Cristo, meu Senhor, porque estou certo de que o mestrado foi tão somente mais uma manifestação da sua graça em minha vida.

Resumo da Dissertação apresentada ao DIMAp/UFRN como parte dos requisitos necessários para obtenção do grau de Mestre em Sistemas e Computação (MSc.).

SISTEMA DE REESCRITA DE TERMOS PARA INTERVALOS: EM DIREÇÃO A UM MODELO FORMAL PARA A COMPUTAÇÃO INTERVALAR

Adriano Xavier Carvalho

Outubro de 2005

Orientador: Regivan Hugo Nunes Santiago
Área de Concentração: Teoria e Inteligência Computacional
Palavras-chave: Sistemas de Reescrita; Computação Intervalar; Ponto-Flutuante
Número de Páginas: 1 + 107

Apresenta-se um sistema de reescrita de termos para aritmética intervalar (adição, subtração e multiplicação), em direção a um modelo matemático para a computação intervalar. Primeiramente, é apresentado um sistema de reescrita de termos cujas regras (equações direcionadas) implementam aritmética de ponto-flutuante binária, baseada no Padrão IEEE 754. Em seguida, este sistema primitivo é estendido com regras para a aritmética intervalar. E, finalmente, a corretude e a terminação do sistema fornecido são ambas discutidas.

Abstract of Dissertation presented to DIMAp/UFRN as a partial fulfillment of the requirements for the degree of Master in Systems and Computation (MSc.).

INTERVAL TERM REWRITING SYSTEM: TOWARD A FORMAL MODEL FOR INTERVAL COMPUTATION

Adriano Xavier Carvalho

October, 2005

Advisor: Regivan Hugo Nunes Santiago
Area of Concentration: Formal Methods
Key words: Rewrite Systems; Interval Computation; Floating-Point
Number of Pages: 1 + 107

We present a term rewriting system for interval arithmetic (addition, subtraction and multiplication), toward a mathematical model for interval computation. We start presenting a term rewriting system whose rules (directed equations) perform binary floating-point arithmetic, which is based on IEEE-754 Standard. Next, this primitive system is extended with rules for interval arithmetic. Finally, correctness and termination of our system are both discussed.

Sumário

Lista de Figuras

Lista de Tabelas

Símbolos e Abreviaturas

Σ	Alfabeto;
	Assinatura (heterogênea ou homogênea)
$\mathcal{F}$	Assinatura (homogênea)
Σ^*	Fecho de Kleene
Λ	Palavra vazia;
	Função vazia
$\mathcal{G}$	Gramática
Φ	Linguagem formal
fbf	Fórmula bem formada
ibf	Indução bem fundada
sss	Se e somente se
$\forall$	Para todo
$\exists$	Existe
$\exists!$	Existe (único)
NaN	Ponto-flutuante não numérico (*"Not a Number"*)
INaN	NaN intervalar
$\mathbb{N}$	Conjunto dos números naturais
$\mathbb{Z}$	Conjunto dos números inteiros
$\mathbb{Q}$	Conjunto dos números racionais
$\mathbb{R}$	Conjunto dos números reais
$\mathbb{IR}$	Conjunto dos intervalos reais
$\mathbb{IF}$	Conjunto dos intervalos Padrão IEEE 754
$\mathbb{INAN}$	Conjunto dos NaN's intervalares
$\mathcal{A}$	Conjunto de axiomas de uma teoria
$\mathcal{R}$	Conjunto de predicados de uma teoria (regras de inferência)
$\vdash$	Conseqüência lógica (lê-se "deriva")
$\models$	Conseqüência semântica (lê-se "modela")
$\trianglelefteq_{sub}$	Relação de sub-termo
$\triangleleft_{sub}$	Relação de sub-termo estrita
$\bowtie$	Incomparável
$\hookleftarrow$	Aplicação de contexto
$\cup$	União
$\cap$	Interseção
$\subset$	Subconjunto (contido ou igual)
$\supset$	Super-conjunto (contido ou igual)
$\sqsubseteq$	Ordem de encompassamento
$\sqsubset$	Ordem de encompassamento estrita

Capítulo 1

Introdução

1.1 Contexto histórico e motivação

A precisão de cálculos no processamento de dados é um imprescindível pré-requisito da computação científica e, ao mesmo tempo, constitui ainda um dos grandes desafios para os sistemas modernos, especialmente aqueles que lidam com a manipulação de números reais. O modelo computacional predominante para representação interna de números reais é o modelo de ponto-flutuante, abordado no Capítulo 5. Tal modelo faz uso de arredondamentos para adequar os sistemas às limitações de memória e processamento, intrínsecas às arquiteturas de computadores. Como o número de ponto-flutuante não representa com exatidão o número real, mas sim uma aproximação deste, ocorre em geral que sucessivas operações aritméticas resultam em acumulativos erros de arredondamento. Uma análise mais detalhada deste problema pode ser encontrada em [Gol91, Wal90].

Desde o surgimento da Matemática Intervalar, proposta por Moore e Sunaga nos anos 60, vários esforços têm sido implementados no sentido de apresentar uma solução alternativa (e mais acurada) para representação dos números reais. A aritmética intervalar é uma aritmética definida sobre conjuntos de intervalos, ao invés de conjuntos de números reais. Neste caso, a representação de um dado número real é feita não por um arredondamento do mesmo, mas sim através de um intervalo que o contenha. A vantagem desta abordagem está na possibilidade de se mensurar os erros, permitindo o seu controle durante o processo computacional. Para um intervalo $[a, b]$ contendo a resposta ideal, o erro máximo será de $b - a$. O Capítulo 5 apresenta brevemente a aritmética intervalar, bem como suas propriedades; para uma introdução mais geral da computação intervalar e suas aplicações, veja [Kea96].

Paralelamente aos esforços destinados à computação intervalar, meios industriais e acadêmicos têm investido cada vez mais na área de métodos formais.

As linguagens de especificação e verificação formal de sistemas têm crescido significativamente em popularidade. Uma grande limitação destas linguagens, entretanto, continua sendo sua carência de soluções efetivas para aplicações que envolvam números reais, como será mostrado mais adiante. A contribuição desta dissertação reside justamente no campo da especificação e verificação formal de sistemas envolvendo números reais, cuja implementação adote a abordagem intervalar. Propõe-se desenvolver um sistema de reescrita de termos capaz de verificar de forma automática aplicações desta natureza, fazendo uso, para tanto, da aritmética intervalar. Esse sistema de reescrita, além de servir de base para a especificação de sistemas que utilizem a aritmética intervalar como abordagem de implementação, pretende fornecer um modelo computacional alternativo em relação às máquinas de Turing (veja, por exemplo, [Bed96]).

1.2 A importância do raciocínio equacional

Equações exercem papel fundamental nas ciências em geral, e em particular na Ciência da Computação. O raciocínio equacional é um importante componente na prova automática de teoremas, linguagens de programação de alto nível, especificação e verificação de programas e inteligência artificial [DV04]. Raciocinar com equações envolve derivar conseqüências de equações dadas e encontrar valores para variáveis que satisfazem uma dada equação.

Reescrita é um poderoso método para lidar com equações. Um sistema de reescrita de termos é basicamente um conjunto de equações orientadas, chamadas *regras de reescrita*, as quais são usadas para substituir iguais por iguais, mas somente na direção indicada [Rin98]; de fato, o que distingue a reescrita de termos da lógica equacional é que o lado esquerdo pode ser substituído pelo lado direito, mas não o contrário, constituindo um modelo computacional *Turing-completo* [BN98]. Deste modo, expressões complexas podem ser substituídas por expressões mais simples equivalentes, até que se obtenha a forma mais simples possível [DJ90].

A teoria de reescrita centraliza-se em torno do conceito da *forma normal*, uma expressão irredutível, i.e. uma expressão que não pode ser reescrita para nenhuma outra. A computação, neste caso, consiste na reescrita para uma forma normal; quando a forma normal é *canônica*, i.e. única, ela é tida como o valor da expressão inicial. Quando a reescrita de termos iguais sempre leva à mesma forma normal, diz-se que o conjunto de regras é convergente e a reescrita pode ser usada para verificar igualdade entre termos.

1.3 Organização do trabalho

O Capítulo 2 destaca alguns conceitos-chave da Ciência da Computação, particularmente relacionados ao estudo de linguagens formais: sistemas de dedução, Σ-álgebras, assinaturas, termos, substituição etc. Fornece, ainda, uma introdução das noções de sistemas de reescrita e dedução equacional. Os conceitos abordados neste capítulo, bem como as respectivas convenções notacionais, são pré-requisitos básicos para a compreensão do texto como um todo.

A seguir, no Capítulo 3, são estudadas três apresentações da lógica equacional: Lógica equacional homogênea, lógica equacional heterogênea e lógica equacional condicional. A lógica equacional em questão está intimamente relacionada com a respectiva lógica de reescrita, estudada no Capítulo 4. Para cada uma das apresentações da lógica equacional, são considerados aspectos sintáticos (sistemas de dedução) e semânticos (modelos).

O Capítulo 4 formaliza a noção de sistemas da reescrita apresentada no Capítulo 2, levando em consideração, também, as propriedades que tais sistemas devem satisfazer, a fim de prover um procedimento de decisão para teorias equacionais. Será mostrado que o *sine qua non* da reescrita de termos está nas propriedades de terminação (uma forma normal pode ser encontrada), confluência (as formas normais são únicas) e completude (resolve quaisquer conjuntos finitos de identidades).

O Capítulo 5, por sua vez, trata da representação computacional dos números reais, começando com um resumo do Padrão IEEE 754 para aritmética de ponto-flutuante, e terminando com uma introdução à Matemática Intervalar de forma sucinta, mas concisa, a fim de descrever formalmente o objeto que está sendo modelado pelo sistema de reescrita proposto, e fornecer o embasamento matemático para as regras do mesmo. Computação de pontos-flutuantes binários, algoritmos intervalares e a aritmética de Moore são tópicos essenciais deste capítulo.

Finalmente, no Capítulo 6, apresentam-se as contribuições particulares deste trabalho, a saber, dois sistemas: O Sistema de Reescrita de Termos para Ponto-flutuante (SRTF), proposto como um modelo formal para computações de números de ponto-flutuantes binários, e o Sistema de Reescrita de Termos para Intervalos (SRTI), o qual estende o SRTF e é proposto como base para a formulação de um modelo computacional para a aritmética intervalar.

As considerações finais da pesquisa, análise de trabalhos correlatos e sugestão de trabalhos futuros são feitas no Capítulo 7.

Capítulo 2

Linguagens, Σ-álgebras e Reescrita

2.1 O papel da reescrita de termos

Enquanto linguagens naturais (Português, Inglês etc.) são definidas informalmente, a maioria dos formalismos da computação (linguagens de programação, por exemplo) é definida por regras explicitamente declaradas em termos das cadeias de símbolos que podem ocorrer no mesmo, e, por isso, são chamados *linguagens formais*. Alguns aspectos básicos da teoria de linguagens formais serão destacados neste capítulo. Os conceitos introduzidos aqui seguem de [BA01, Dav89, Hen88, ABL02, BN98, Gog77, GM96, Bir35, KK01, Bar98, dM96, RS63, Bee85].

O estudo de linguagens formais pode ser abordado a partir de vários pontos de vista. Primeiramente, na Seção 2.2, será enfatizado o aspecto *dedutivo* de linguagens, ou seja, a linguagem em termos de seus símbolos e regras para geração de cadeias válidas de símbolos; esta abordagem é conhecida como *semântica axiomática*, e nota-se que ela é puramente sintática.

Em seguida, a Seção 2.3 apresentará outra abordagem, conhecida como *semântica denotacional*, que visa munir as linguagens formais de um significado, fornecendo mecanismos algébricos para *interpretar* aquilo que pode ser expresso sintaticamente por uma linguagem. A Seção 2.4, depois, traz o conceito de substituição.

Existe, ainda, um terceiro ponto de vista, conhecido como *semântica operacional*, que lida com o aspecto *computacional* de uma dada linguagem formal, e será tratado na Seção 2.5. Ressalta-se que aqui reside o papel da reescrita de termos, pois pode-se olhar para programas como termos, e, neste caso, um sistema de reescrita de termos proverá justamente a semântica operacional de tais programas. Tais sistemas serão melhor explorados no Capítulo 4.

2.2 Aspectos sintáticos

2.2.1 Um pouco de computabilidade

Em 1936, Alonzo Church e Alan Turing propuseram, respectivamente, dois modelos matemáticos com proposta de formalização do conceito de **procedimento efetivo**, i.e um programa que execute alguma operação matemática [Cut80], como, por exemplo, os procedimentos para calcular o máximo divisor comum de dois números naturais, ou a raiz quadrada de um número natural.

Desde os trabalhos de Church e Turing, surgiram inúmeros modelos de computação, tais como Máquinas de Acesso Aleatório (RAMs) [SB04, Smi94], Máquinas de Ilimitados Registradores (URMs) [Cut80], Sistemas de Post [BL74] etc. Todos estes modelos são equivalentes, no sentido de que formalizam a mesma classe de funções, a saber, as funções parciais recursivas [SB04, Smi94]. Dessa forma, diz-se que um determinado modelo de computação é **Turing-completo** se ele é equivalente a um dos modelos acima.

Definição 2.2.1 Dado um conjunto A, uma função parcial $f : A \to A$, f é **computável** se existe uma função $f^* : \mathbb{N} \to \mathbb{N}$ e um procedimento efetivo injetivo $\alpha : A \to \mathbb{N}$, chamado **codificação** de A em $\mathbb{N}$, tal que o seguinte diagrama comuta:

$$\begin{array}{ccc} A & \xrightarrow{f} & A \\ {\scriptstyle \alpha^{-1}}\uparrow & & \downarrow{\scriptstyle \alpha} \\ \mathbb{N} & \xrightarrow{f^*} & \mathbb{N} \end{array}$$

ou seja, $f^* = \alpha \circ f \circ \alpha^{-1}$.

Notação: A composição de duas funções f e g quaisquer será denotada por $g \circ f$, onde $(g \circ f)(x) = g(f(x))$.

Definição 2.2.2 Um conjunto A diz-se **recursivamente enumerável** se existe uma enumeração efetiva $f : \mathbb{N} \to A$ tal que $Imagem(f) = A$.

Definição 2.2.3 Um conjunto A diz-se **recursivo** se a sua função característica $C_A : \mathcal{U} \to \{0, 1\}$ é computável.

Nota: É possível provar que todo conjunto recursivo é um conjunto recursivamente enumerável.

Definição 2.2.4 Um predicado ou propriedade P diz-se **decidível** se a sua função característica $C_p : \mathcal{U} \to \{0,1\}$ é computável. Se a função parcial $f_p : \mathcal{U} \to \{0,1\}$ tal que

$$f_p(u) = \begin{cases} 1 & \text{se } P(u) \\ indef. & \text{caso contrário} \end{cases} \tag{2.1}$$

é computável, então P é chamado predicado **semi-decidível**.

2.2.2 Sistemas de dedução

A noção de sistemas de dedução é essencial para a definição de lógicas e teorias de prova.

Definição 2.2.5 Um **Sistema de dedução** [KK01] ou uma **Teoria formal** [Dav89] é uma quádrupla $(\Sigma, \Phi, \mathcal{A}, \mathcal{R})$, onde:

- Σ é um alfabeto,
- $\Phi = \{\phi_0, \phi_1, ...\}$ é uma linguagem formal sobre Σ, chamada conjunto de fórmulas bem-formadas ($fbfs$),
- $\mathcal{A} = \{a_0, a_1, ...\}$ é um subconjunto de Φ, chamado conjunto de axiomas,
- $\mathcal{R} = \{r_0, r_1, ..., r_n\}$ é um conjunto finito de predicados sobre Φ, chamados regras de inferência.

Conjuntos infinitos de axiomas podem ser especificados por esquemas axiomáticos. Uma regra de inferência é escrita como:

$$\textsc{Nome}\ \phi_0, ..., \phi_1 \vdash \phi_n \text{ se } \textit{Condição}$$

e significa:

"Dado $\phi_0, ..., \phi_1$ deduz-se ϕ_n se *Condição* é satisfeita."

Exemplo 2.2.1 Seja $(\Sigma, \Phi, \mathcal{A}, \mathcal{R})$ um sistema de dedução tal que:

- $\Sigma = \{p, q, u, \wedge, \to, (,)\}$ é o alfabeto.
- Φ é o conjunto de $fbfs$, onde: $p, q, u \in \Phi$ e, se ϕ_0 e ϕ_1 são $fbfs$, então $(\phi_0 \wedge \phi_1)$ e $(\phi_0 \to \phi_1)$ também são.

- $\mathcal{A} = \{(\phi_0 \rightarrow \phi_0)\}$ é o conjunto de esquemas axiomáticos.
- $\mathcal{R} = \{r_0, r_1, r_2\}$ é o conjunto de regras de inferência, onde
  r_0 é definida como

$$\phi_0, (\phi_0 \rightarrow \phi_1) \vdash \phi_1,$$

  r_1 é definida como

$$(\phi_0 \wedge \phi_1) \vdash \phi_0,$$

  e r_2 é definida como

$$(\phi_0 \wedge \phi_1) \vdash \phi_1.$$

Definição 2.2.6 Uma **derivação** d da **conclusão** $c \in \Phi$ das *premissas* $P = \{p_0, ..., p_{n-1}\} \subseteq \Phi$ é uma seqüência não-vazia finita $d_0, ..., d_m$ tal que $d_i \in \Phi, d_m = c$ e ou $d_i \in \mathcal{A}$ (d_i é um axioma), ou $d_i \in P$ (d_i é uma **premissa** ou **hipótese**), ou d_i foi obtido aplicando-se alguma regra de inferência de $\mathcal{R}$ para um conjunto $\{d_j, ...d_k\}$ de fórmulas tal que $d_j, ..., d_k \in d$ e $j, ..., k < i$. Neste caso, diz-se que c é **dedutível de** P naquele sistema de dedução, e isto se escreve como:

$$p_0, ..., p_{n-1} \vdash c. \tag{2.2}$$

Assim, seguindo o Exemplo 2.2.1, a partir do conjunto de premissas $P = \{p \wedge q, p \rightarrow u\}$, pode-se chegar à conclusão $c = u$, ou seja,

$$p \wedge q, p \rightarrow u \vdash u. \tag{2.3}$$

Neste caso, uma derivação d da conclusão u é dada pela seqüência

$$d \;=\; d_0, d_1, d_2, d_3 \;=\; p \wedge q, p, p \rightarrow u, u \tag{2.4}$$

onde:

$d_0 = p \wedge q$ pertence a P, ou seja, d_0 é uma hipótese (premissa),
$d_1 = p$ foi obtido pela aplicação de r_1 em $\{d_0\}$,
$d_2 = p \rightarrow u$ pertence a P, ou seja, d_2 é uma hipótese (premissa), e
$d_3 = u$ foi obtido pela aplicação de r_0 em $\{d_1, d_2\}$.

Definição 2.2.7 Uma fórmula $th \in \Phi$ que pode ser derivada a partir do conjunto vazio de premissas é chamada **teorema**. Diz-se que th é **provável** no sistema de dedução. Isto se escreve como:

$$\vdash th. \tag{2.5}$$

Uma derivação de um teorema é chamada **prova**. O conjunto de todos os teoremas é denotado por $\mathcal{TH}$.

Definição 2.2.8 Uma **interpretação** fornece um significado para cada símbolo do sistema de dedução, tal que toda fbf passa a ser entendida como uma sentença que é ou verdadeira ou não segundo essa interpretação.

A definição acima é apenas uma intuição da noção de interpretação, a qual será definida de forma mais precisa em termos de Σ-álgebras adiante.

Definição 2.2.9 Uma interpretação é um **modelo** para um conjunto de $fbfs$ P se P é verdadeiro naquela interpretação. Diz-se que uma interpretação é um modelo para um determinado sistema de dedução se ela é um modelo para o seu conjunto de teoremas.

A relação entre os universos sintático e semântico de um sistema de dedução é dada pelas propriedades de *completude* e *corretude*:

Definição 2.2.10 Um sistema de dedução é **completo** se toda sentença que é verdadeira em todas as interpretações é provável no sistema.

Definição 2.2.11 Um sistema de dedução é **correto** ou **seguro** se toda sentença provável no sistema é verdadeira em todas as interpretações.

O termo *completude* também é usado para relacionar o sistema dedutivo com o método computacional:

Definição 2.2.12 Um método computacional é **completo** se, para cada sentença ϕ da linguagem Φ, esse método termina sob entrada ϕ em uma quantidade finita de tempo, indicando se ϕ é verdadeira ou não em todas as interpretações.

Outros dois conceitos de particular interesse são os de *decidibilidade* e *consistência* de um sistema de dedução. Mas antes, segue uma definição informal de *decidibilidade*:

Definição 2.2.13 Uma propriedade é chamada **decidível** se existe um procedimento efetivo que irá determinar se a propriedade é satisfeita (em algum sentido) ou não.

A definição acima foi propositalmente vaga, pois deseja-se aplicá-la em diferentes contextos: propriedades de símbolos (x é um símbolo da teoria?), cadeias de símbolos (x é uma fbf?), seqüências de $fbfs$ (x é uma prova?), e assim por diante. No caso de um sistema de dedução:

Definição 2.2.14 Um sistema de dedução é **decidível** se existe um procedimento efetivo que irá determinar, para qualquer sentença da teoria, se aquela sentença é ou não provável no sistema.

Definição 2.2.15 Um sistema de dedução é **consistente** se ele não contém nenhuma fbf tal que ao mesmo tempo esta fbf e sua negação sejam prováveis no sistema.[1]

Definição 2.2.16 Um procedimento efetivo que implementa a função característica do conjunto de teoremas de um determinado sistema de dedução é chamado de um **procedimento de decisão** para aquele sistema. Observe-se que, se o conjunto de todos os teoremas $\mathcal{TH}$ é recursivo, então o sistema de dedução é chamado **decidível**. Se $\mathcal{TH}$ não é recursivo, mas é recursivamente enumerável, então o sistema de dedução é chamado **semi-decidível**. Se $\mathcal{TH}$ não é recursivamente enumerável, então o sistema de dedução é chamado **indecidível**.

2.2.3 Assinaturas

Termos são obtidos pela combinação de símbolos de funções, constantes ou variáveis. Por exemplo, para um símbolo de função binário $+$ e variáveis x, y, então "$+(x, y)$" é um termo (esta notação é chamada **prefixa**); este mesmo termo poderia também ser escrito como "$(x, y)+$" (notação **sufixa**), ou ainda, "$x+y$" (notação **infixa**). O conceito de *assinatura* possibilita esclarecer quais símbolos de função estão disponíveis, em um certo contexto, e qual a quantidade de argumentos que cada um tem.

Definição 2.2.17 Uma **assinatura** (funcional) é uma tripla $\Sigma = \langle S, TF, PF \rangle$, onde S é um conjunto de ***sorts*** (nomes de conjuntos), TF é um conjunto de símbolos de função que designam funções totais, e PF é um conjunto de símbolos de função que designam funções parciais. O **perfil** de um símbolo de função $f \in TF \cup PF$ é um par (w, s) onde $w \in S^*$ (S^* é o fecho de Kleene de S) e $s \in S$; escreve-se $f : w \mapsto s$ ou simplesmente $f_{w,s}$ para indicar que (w, s) é o perfil de f. A cada $f \in TF \cup PF$ é associado um número natural, chamado **aridade**, tal que para $w = s_1...s_n$, $aridade(f) = n$. Elementos de aridade zero são chamados **constantes**; sempre assume-se que há pelo menos uma constante. Uma assinatura é chamada de **heterogênea** quando ela possui mais de um *sort*, e **homogênea** quando ela possui apenas um *sort*.

Nota: A idéia de se ter na assinatura um conjunto PF de símbolos de funções parciais é de especial importância quando se trabalha com predicados de lógica equacional condicional, conforme será apresentado na Seção 3.4.

[1] Existem sistemas de dedução nos quais a consistência não é tida como um predicado fundamental, e.g. Lógica Paraconsistente, a qual admite a inconsistência e, em contrapartida, exige a não trivialidade, i.e. Φ tem que ser um subconjunto próprio de Σ^*.

Notação: Em todo o texto que se segue, será utilizada a notação Σ para assinaturas em geral, conforme o Exemplo 2.2.3 e o Exemplo 2.2.4 abaixo, e $\mathcal{F}$ para o caso particular de assinaturas homogêneas, conforme o Exemplo 2.2.2 abaixo. Por simplicidade, o *sort*, quando for único, pode ser omitido, ficando a assinatura definida em termos do conjunto $TF \cup PF$, conforme o Exemplo 2.2.2; analogamente, PF será omitido quando for vazio, conforme o Exemplo 2.2.3.

Seguem alguns exemplos de assinaturas válidas:

Exemplo 2.2.2 $\mathcal{F} = \{0, s, +, *\}$, onde $aridade(0) = 0$, $aridade(s) = 1$, $aridade(+) = 2$, e $aridade(*) = 2$.

Exemplo 2.2.3 $\Sigma = \langle\{Nat\}, \{Zero, Suc\}\rangle$, com $Zero_{\Lambda,Nat}$ e $Suc_{Nat,Nat}$.

Exemplo 2.2.4 $\Sigma = \langle\{Entrada, Estado, Saida\}, \{i, g\}, \{f\}\rangle$, com $i_{\Lambda,Estado}$, $g_{Estado,Saida}$ e $f_{EntradaEstado,Estado}$.

2.2.4 Termos

Em lógica, termos são expressões simbólicas que representam elementos de um conjunto. Por exemplo, os números 0 e $4 + 3 \in \mathbb{N}$ podem ser representados pelos termos "0" e "$s(s(s(s(0)))) + s(s(s(0)))$", respectivamente. Neste sentido, termos exercem o papel de substantivos próprios, i.e. são nomes que referenciam objetos de um determinado conjunto. Nesta seção serão apresentadas as várias representações dos termos de primeira ordem: como cadeias, árvores, mapeamentos, e funções. Mas antes, a noção de *ordem de sub-termo* é introduzida:

Definição 2.2.18 A propriedade de um termo ser sub-termo de outro induz uma relação de ordem sobre os termos, chamada **ordem de sub-termo**. Será usada a notação $s \trianglelefteq_{sub} t$ para denotar que s é um sub-termo de t e $s \triangleleft_{sub} t$ para denotar que s é um sub-termo *estrito* de t.

2.2.4.1 Termos como cadeias

Termos de primeira ordem são construídos sobre um vocabulário de símbolos de variáveis e símbolos de função especificados por uma assinatura. Aqui, eles serão considerados como cadeias (*strings*).

Definição 2.2.19 Seja $\mathcal{F}$ uma assinatura (homogênea), e $\mathcal{X}$ um conjunto (enumerável) de símbolos de **variáveis**, o **conjunto dos termos homogêneos** (de primeira ordem) $\mathcal{T}(\mathcal{F}, \mathcal{X})$ é o menor conjunto contendo $\mathcal{X}$ e tal que a cadeia $f(t_1, ..., t_n)$ esteja em $\mathcal{T}(\mathcal{F}, \mathcal{X})$ sempre que $aridade(f) = n$ e $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ para

$i \in [1..n]$. O conjunto dos termos sem variáveis, chamado **conjunto dos termos fechados**, é denotado por $\mathcal{T}(\mathcal{F})$. Termos que contém variáveis são chamados **abertos**. Um termo é **linear** se cada uma de suas variáveis ocorre somente uma vez no mesmo. Constantes são denotadas pelas letras $a, b, c, ...$ e variáveis são denotadas pelas letras x, y, z, podendo ser indexadas, i.e. $x_1, x_2, ...$, e termos pelas letras $l, r, g, d, p, q, s, t, u, v, w, l', r', g', ...$ Por exemplo, "0" e "$s(s(0)) + s(0)$" são termos fechados, mas "$s(0) + x$" e "$y + z$" são termos abertos; "$x + s(s(s(0))) + y$" é linear, mas "$x + s(s(0)) + x + s(0)$" não é linear.

Exemplo 2.2.5 Dada uma assinatura $\mathcal{F} = \{Zero, Suc, Soma, Div\}$, com $aridade(Zero) = 0$, $aridade(Suc) = 1$, $aridade(Soma) = 2$ e $aridade(Div) = 2$, então $\mathcal{T}(\mathcal{F})$ contém os termos:

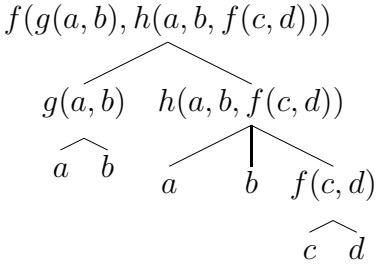
$Zero$
$Suc(Zero), Suc(Suc(Zero)), Suc(Suc(Suc(Zero))), ...$
$Soma(Zero, Zero), Soma(Zero, Suc(Zero)), ...$
$Div(Zero, Zero), Div(Zero, Suc(Zero)), Div(Suc(Zero), Suc(Zero)), ...$

2.2.4.2 Termos como árvores

Um termo t pode ser visto também como uma árvore rotulada finita, cujas folhas são rotuladas com variáveis ou constantes, e cujos nós internos são rotulados com símbolos de aridade positiva.

Definição 2.2.20 Tomando $\mathbb{N}$ como um alfabeto, então $\langle \mathbb{N}^*, . \rangle$ é o fecho de Kleene munido da operação de concatenação. Uma **posição** (também chamada **ocorrência**) em um termo t é representada como uma cadeia $\omega \in \mathbb{N}^*$, que descreve o caminho da **raiz** de t para a raiz do sub-termo $t|_\omega$, naquela posição. Um termo u tem uma ocorrência em t se $u = t|_\omega$ para alguma posição ω em t.

Exemplo 2.2.6 O termo $t = f(g(a, b), h(a, b, f(c, d)))$ é representado pela seguinte árvore:



onde $t|_1 = g(a, b)$, $t|_{11} = a$, $t|_{12} = b$, $t|_2 = h(a, b, f(c, d))$, $t|_{21} = a$, $t|_{22} = b$, $t|_{23} = f(c, d)$, $t|_{231} = c$ e $t|_{232} = d$.

Usa-se $t[s]_\omega$ para enfatizar que o termo t contém s como sub-termo na posição ω. Em alguns casos, ω pode ser omitido. Seguindo o exemplo acima, $t[g(a,b)]_1$, $t[f(c,d)]_{23}$, $t[d]_{232}$ etc.

A palavra vazia Λ denota, portanto, o caminho vazio para a raiz da árvore e $\upsilon, \omega \in \mathbb{N}^* - \Lambda$ denotam as demais sub-árvores. Assim, posições são ordenadas da seguinte maneira: $\omega_1 \leq \omega_2$ se existe uma ω_3 tal que $\omega_1\omega_3 = \omega_2$. Se duas posições ω_1 e ω_2 são incomparáveis, então isto é denotado por $\omega_1 \bowtie \omega_2$.

2.2.4.3 Termos como mapeamentos

Um termo também pode ser visto como um mapeamento parcial de $\langle \mathbb{N}^*, . \rangle$ para uma assinatura $\mathcal{F}$. O domínio deste mapeamento, denotado por $\mathcal{D}om(t)$, é o conjunto das posições dos símbolos de $\mathcal{F}$ que ocorrem em t, também chamado *domínio* de t; ele deve ser não vazio e fechado sob prefixo, ou seja, se $\omega \in \mathcal{D}om(t)$, então todos os prefixos de ω pertencem ao $\mathcal{D}om(t)$. O tamanho $|t|$ do termo t é a cardinalidade de $\mathcal{D}om(t)$. A quantidade de nós do termo t rotulados com o símbolo f é denotada por $|t|_f$. $\mathcal{V}ar(t)$ denota o conjunto de variáveis em t. $\mathcal{G}rd(t)$ é o conjunto de posições cujos termos associados não contém variáveis. Por extensão, o mapeamento cujo domínio é vazio é chamado **termo vazio** e denotado por Λ[2].

Exemplo 2.2.7 O termo t, do Exemplo 2.2.6, é representado pelo seguinte mapeamento:

$$\begin{array}{lcl} \Lambda & \mapsto & f \\ 1 & \mapsto & g \\ 11 & \mapsto & a \\ 12 & \mapsto & b \\ 2 & \mapsto & h \\ 21 & \mapsto & a \\ 22 & \mapsto & b \\ 23 & \mapsto & f \\ 231 & \mapsto & c \\ 232 & \mapsto & d \end{array}$$

Definição 2.2.21 Quando um termo t contém um sub-termo u na posição $\omega \in \mathbb{N}^*$, i.e. $t[u]_\omega$, e esse sub-termo u é trocado por um termo s, dando origem a um novo termo t', onde $t'[s]_\omega$, usa-se a notação $t'[\omega \hookleftarrow s]$ e chama-se esta operação de **aplicação de contexto**.

[2] Portanto, o mapeamento em questão trata-se da função vazia [SB04], aqui denotada por Λ; assim, usa-se Λ para designar a posição da raiz e a função vazia.

Exemplo 2.2.8 Seguindo o Exemplo 2.2.6, ao substituir-se o sub-termo $u = f(c,d)$, que está na posição 23 do termo t, pelo termo $s = c$, obtém-se o termo $t' = f(g(a,b),h(a,b,c))$, e usa-se $t'[23 \hookleftarrow c]$ para denotar esta aplicação de contexto.

2.2.4.4 Termos como operações

Para qualquer termo t e para qualquer subconjunto $V = \{x_1, ..., x_n\}$ de $\mathcal{V}ar(t)$, pode-se associar uma função denotada por $t(x_1, ..., x_n)$ de $\mathcal{T}(\mathcal{F},\mathcal{X})^n$ para $\mathcal{T}(\mathcal{F},\mathcal{X})$ tal que:

$$\begin{array}{lccc} t(x_1,...,x_n): & \mathcal{T}(\mathcal{F},\mathcal{X})^n & \rightarrow & \mathcal{T}(\mathcal{F},\mathcal{X}) \\ & t_1,...,t_n & \mapsto & t[x_i \hookleftarrow t_i]_{i=1,...,n} \end{array}$$

Portanto, um termo pode ser visto como um novo operador $op : \mathcal{T}(\mathcal{F},\mathcal{X})^n \rightarrow \mathcal{T}(\mathcal{F},\mathcal{X})$, que é derivado das operações primitivas de $f \in \Sigma_{\mathcal{T}(\mathcal{F},\mathcal{X})}$.

Em particular, tomando $V = \mathcal{V}ar(t)$, qualquer termo t pode ser considerado como um operador derivado.

2.2.4.5 Termos heterogêneos

Uma extensão bastante útil da noção de termos de primeira ordem apresentada na Definição 2.2.19 é o conceito de termos heterogêneos. Termos heterogêneos são obtidos quando símbolos de função, variáveis e termos são categorizados em classes, chamadas *sorts*. A seguir, a Definição 2.2.19 será generalizada para assinaturas heterogêneas.

Definição 2.2.22 Seja $\Sigma = \langle S, TF, PF\rangle$ uma assinatura (heterogênea), e $\mathcal{X} = \bigcup_{s\in S} \mathcal{X}_s$ um conjunto de variáveis heterogêneas, onde cada $\mathcal{X}_s$ indica o conjunto de variáveis de *sort* s, o conjunto dos termos de *sort* s, denotado por $\mathcal{T}(\Sigma,\mathcal{X})_s$, é o menor conjunto contendo $\mathcal{X}_s$ e qualquer constante a de *sort* s tal que $f(t_1,...,t_n)$ esteja em $\mathcal{T}(\Sigma,\mathcal{X})_s$ sempre que $f_{s_1...s_n,s}$ e $t_i \in \mathcal{T}(\Sigma,\mathcal{X})_{s_i}$ para $i \in [1..n]$. O **conjunto dos termos heterogêneos** (de primeira ordem) $\mathcal{T}(\Sigma,\mathcal{X})$ é a família $\{\mathcal{T}(\Sigma,\mathcal{X})_s | s \in S\}$. Analogamente, o conjunto dos termos heterogêneos fechados será denotado por $\mathcal{T}(\Sigma)$, e o conjunto dos termos heterogêneos fechados de *sort* s será denotado por $\mathcal{T}(\Sigma)_s$.

Exemplo 2.2.9 Dada uma assinatura $\Sigma = \langle\{Nat, Bool\}, \{Zero, Suc, Soma, V, F, Maior, Menor\}, \{Div\}\rangle$, com $Zero_{\Lambda,Nat}$, $Suc_{Nat,Nat}$, $Soma_{NatNat,Nat}$, $V_{\Lambda,Bool}$, $F_{\Lambda,Bool}$, $Maior_{NatNat,Bool}$, $Menor_{NatNat,Bool}$ e $Div_{NatNat,Nat}$, então $\mathcal{T}(\Sigma) = \{\mathcal{T}(\Sigma)_{Nat}, \mathcal{T}(\Sigma)_{Bool}\}$, onde

$\mathcal{T}(\Sigma)_{Nat}$ contém os termos:

$Zero$
$Suc(Zero), Suc(Suc(Zero)), Suc(Suc(Suc(Zero))), ...$
$Soma(Zero, Zero), Soma(Zero, Suc(Zero)), ...$
$Div(Zero, Zero), Div(Zero, Suc(Zero)), Div(Suc(Zero), Suc(Zero)), ...$

e $\mathcal{T}(\Sigma)_{Bool}$ contém os termos:

V
F
$Maior(Zero, Zero), Maior(Suc(Zero), Zero), ...$
$Menor(Zero, Zero), Menor(Suc(Zero), Zero), ...$

2.3 Σ-álgebras

2.3.1 Álgebras homogêneas

Em termos gerais, uma **álgebra** pode ser vista simplesmente como um conjunto munido de operações sobre seus elementos. Aqui será abordado um caso de particular interesse, que é o conceito de Σ-álgebra, primeiramente para assinaturas homogêneas, e na seção seguinte, para assinaturas heterogêneas.

Definição 2.3.1 Dada uma assinatura homogênea Σ, uma **Σ-álgebra (homogênea)** é um par $\langle A, \Sigma_A \rangle$, onde

i) A é um conjunto, chamado conjunto suporte (em inglês, *carrier*);

ii) Σ_A é um conjunto de funções $\{f_A | f \in \Sigma\}$, tal que se $aridade(f) = n$ então f_A é uma função de $A^n \to A$. A função f_A é chamada de *interpretação* de f, também denotada por $\iota(f)$. Diz-se que A é fechado sob as funções $f_A \in \Sigma_A$.

Desta forma, uma Σ-álgebra é simplesmente uma interpretação da assinatura Σ, consistindo de um conjunto A e uma interpretação sobre A para cada símbolo de função em Σ. Naturalmente, uma dada assinatura pode ter várias interpretações em diferentes conjuntos suporte, e ainda, interpretações diferentes sobre o mesmo conjunto suporte.[3]

[3]Para um mesmo símbolo $f \in \Sigma$, pode-se ter diferentes operações que interpretam f.

Exemplo 2.3.1 Seja a assinatura $\Sigma = \{Zero, Suc, Pred, Soma\}$, onde $aridade(Zero) = 0$, $aridade(Suc) = 1$, $aridade(Pred) = 1$ e $aridade(Soma) = 2$. Então $\langle N, \Sigma_N \rangle$ é uma Σ-álgebra onde

i) N é o conjunto dos números naturais;

ii) Σ_N é dado por:

a) $Zero_N$ é o número natural 0;
b) $Suc_N : N \to N$ é definido por: $Suc_N(n) = n + 1$;
c) $Pred_N : N \to N$ é definido por: $Pred_N(0) = 0, Pred_N(n) = n - 1$, onde $n \neq 0$;
d) $Soma_N : N^2 \to N$ é definido por: $Soma_N(m, n) = m + n$;
onde $+$ e $-$ denotam as operações aritméticas de soma e subtração usuais, respectivamente.

Exemplo 2.3.2 Outra interpretação para a assinatura Σ descrita no Exemplo 2.3.1 pode ser dada pela Σ-álgebra $\langle Z, \Sigma_Z \rangle$ onde

i) Z é o conjunto dos números inteiros;

ii) Σ_Z é dado por:

a) $Zero_Z$ é o número inteiro 0;
b) $Suc_Z : Z \to Z$ é definido por: $Suc_Z(n) = n + 1$;
c) $Pred_Z : Z \to Z$ é definido por: $Pred_Z(n) = n - 1$;
d) $Soma_Z : Z^2 \to Z$ é definido por: $Soma_Z(m, n) = m + n$;
onde $+$ e $-$ denotam as operações aritméticas de soma e subtração usuais, respectivamente.

Para cada assinatura Σ existe uma importante Σ-álgebra chamada de Σ-*álgebra dos termos*. Tais álgebras são puramente objetos formais, em que o conjunto suporte é o próprio conjunto $\mathcal{T}(\Sigma)$ dos *termos* construídos usando os símbolos de função em Σ (conforme a Definição 2.2.19 e a Definição 2.2.22), e as funções são meras manipulações destes termos. Formalmente:

Definição 2.3.2 Seja a assinatura Σ. A **Σ-álgebra dos termos** é o par $\langle \mathcal{T}(\Sigma), \Sigma_{\mathcal{T}(\Sigma)} \rangle$, onde:

i) $\mathcal{T}(\Sigma)$ é o conjunto dos termos para Σ;

ii) $\Sigma_{\mathcal{T}(\Sigma)}$ é um conjunto de funções $\{f_{\mathcal{T}(\Sigma)}, f \in \Sigma\}$, tal que

a) se $aridade(f) = 0$ então $f_{\mathcal{T}(\Sigma)} = f$, senão
b) se $aridade(f) = n$ então $f_{\mathcal{T}(\Sigma)}$ é uma função de $\mathcal{T}(\Sigma)^n \to \mathcal{T}(\Sigma)$, definida como $f_{\mathcal{T}(\Sigma)}(t_1, ..., t_n) = f(t_1, ..., t_n)$.

Notação: Seguindo a notação $\mathcal{F}$ adotada na Seção 2.2.3 para assinaturas homogêneas, a $\mathcal{F}$-álgebra dos termos será explicitamente denotada pelo par $\langle T(\mathcal{F}), \mathcal{F}_{T(\mathcal{F})} \rangle$ quando o contexto exigir tal diferenciação, como é o caso dos capítulos 3 e 4.

Deve-se notar que a álgebra dos termos é sintática por natureza. De fato, a sintaxe de muitas linguagens de programação ou linguagens formais em geral pode ser dada pela álgebra dos termos para alguma assinatura Σ em particular e, neste caso, a semântica (denotacional) será dada por alguma Σ-álgebra que satisfaça certas restrições.

Para que se compreenda a enorme importância matemática e computacional da álgebra dos termos, como será visto mais adiante, é necessário que se introduza, primeiramente, o conceito de Σ-homomorfismos, que são simplesmente funções entre conjuntos suporte de Σ-álgebras que preservam as operações. Formalmente:

Definição 2.3.3 Sejam $\mathcal{A} = \langle A, \Sigma_A \rangle$ e $\mathcal{B} = \langle B, \Sigma_B \rangle$ duas Σ-álgebras. Uma função $\theta : A \to B$ é um **Σ-homomorfismo** de A em B se, para todo $f \in \Sigma$, com $aridade(f) = n$,

$$\theta(f_A(a_1, ..., a_n)) = f_B(\theta(a_1), ..., \theta(a_n)). \tag{2.6}$$

Diz-se que θ é um **endomorfismo** se $A = B$; θ é um **monomorfismo** se a função é injetora; θ é um **epimorfismo** se a função é sobrejetora; e θ é um **isomorfismo** se θ é uma bijeção.

Nota: Substituições em $\mathcal{T}(\mathcal{F}, \mathcal{X})$, tal como serão definidas na Seção 2.4, são, de fato, endomorfismos de $\mathcal{T}(\mathcal{F}, \mathcal{X})$, o que justifica sua propriedade fundamental: para quaisquer termos $t_1, ..., t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ e símbolo $f \in \mathcal{F}$:

$$\sigma(f(t1, ..., tn)) = f(\sigma(t_1), ..., \sigma(t_n)). \tag{2.7}$$

Exemplo 2.3.3 Seja a assinatura Σ, tal como no Exemplo 2.3.1 e no Exemplo 2.3.2, e seja $i : N \to Z$ a função de injeção tal que $i(n) = n$ para todo $n \in N$. Então i *não* é um Σ-homomorfismo entre $\langle N, \Sigma_N \rangle$ e $\langle Z, \Sigma_Z \rangle$, visto que a equação 2.6 não é satisfeita para o símbolo de função $Pred$, pois $i(Pred_N(0)) = i(0) = 0$, mas $Pred_Z(i(0)) = Pred_Z(0) = -1$.

Exemplo 2.3.4 Seja a assinatura Σ, tal como no Exemplo 2.3.1 e no Exemplo 2.3.2, seja P o conjunto dos números naturais pares, i.e. $P = \{0, 2, 4, ...\}$ e seja Σ_P definido como:

a) $Zero_P = 0$

b) $Suc_P : P \to P$ é definido como $Suc_P(n) = n + 2$

c) $Pred_P : P \to P$ é definido como $Pred_P(0) = 0$ e $Pred_P(n) = n - 2$, se $n > 0$

d) $Soma_P : P^2 \to P$ é definido como $Soma_P(m, n) = m + n$.

Então $\langle P, \Sigma_P \rangle$ é uma Σ-álgebra.

Seja $i : N \to P$ a função de injeção tal que $i(n) = 2n$ para todo $n \in N$. Então i é um Σ-homomorfismo. Para provar isto, basta mostrar que todo símbolo $f \in \Sigma$ satisfaz a equação 2.6:

i) $i(Zero_N) = i(0) = 0$ e $Zero_P = 0$

ii) $i(Suc_N(n)) = i(n + 1) = 2n + 2$ e $Suc_P(i(n)) = Suc_P(2n) = 2n + 2$

iii) $i(Pred_N(0)) = i(0) = 0$ e $Pred_P(i(0)) = Pred_P(0) = 0$;
$i(Pred_N(n)) = i(n - 1) = 2n - 2$ e $Pred_P(i(n)) = Pred_P(2n) = 2n - 2$

iv) $i(Soma_N(m, n)) = i(m+n) = 2m+2n$ e $Soma_P(i(m), i(n)) = Soma_P(2m, 2n) = 2m + 2n$. □

Nota: Em geral, Σ-homomorfismos se comportam da mesma maneira que funções usuais sobre conjuntos. Por exemplo, se $f : A \to B$ e $g : B \to C$ são Σ-homomorfismos então a composição $g \circ f : A \to C$ também o é. Além disso, a função identidade sobre um dado conjunto A também é um Σ-homomorfismo para qualquer Σ-álgebra $\langle A, \Sigma_A \rangle$.

A propriedade mais importante da álgebra dos termos é expressa em termos de homomorfismos:

Teorema 2.3.1 (Teorema da Inicialidade) *Para toda Σ-álgebra $\langle A, \Sigma_A \rangle$, existe um único Σ-homomorfismo $\theta_A : \mathcal{T}(\Sigma) \to A$.*

Nota: Fixando-se uma Σ-álgebra $\langle A, \Sigma_A \rangle$, e entendendo um Σ-homomorfismo da álgebra dos termos em A como uma interpretação ou tradução dos termos de $\mathcal{T}(\Sigma)$ em elementos de A, o Teorema da Inicialidade assegura que essa interpretação/tradução é única.

A definição do conjunto suporte da álgebra dos termos $\langle \mathcal{T}(\Sigma), \Sigma_{\mathcal{T}(\Sigma)} \rangle$ dada anteriormente é indutiva. $T(\Sigma)$ é o menor conjunto de cadeias que contém os símbolos de constante e que são fechadas sob as operações da álgebra dos termos, i.e. as operações $f_{\mathcal{T}(\Sigma)}$, para cada $f \in \Sigma$. A natureza indutiva de $\mathcal{T}(\Sigma)$ fornece um poderoso método de prova para derivar propriedades de termos. Para mostrar que a propriedade P vale para todos os termos $\mathcal{T}(\Sigma)$ é suficiente

i) provar que P vale para todos os símbolos de constantes em Σ;

ii) assumindo que P vale para os termos $t_1, ..., t_n$, provar que P vale para o termo $f(t_1, ..., t_n)$ para todo $f \in \Sigma$ de aridade n, com $n > 0$.

Isto é o que Henessy [Hen88] chama de **indução estrutural**, visto que a indução é realmente na estrutura sintática dos termos. Pode-se usar a indução estrutural também para definir relações ou novas funções sobre $\mathcal{T}(\Sigma)$. Por exemplo, para definir uma função g sobre $\mathcal{T}(\Sigma)$, é suficiente

i) definir o resultado da aplicação de g nos símbolos de constantes;

ii) definir o resultado da aplicação de g em $f(t_1, ..., t_n)$ em termos de $g(t_1), ...,$ $g(t_n)$ para todo $f \in \Sigma$ de aridade n, com $n > 0$.

Prova: Uma prova para o Teorema 2.3.1 pode ser encontrada, por exemplo, em [Hen88], pg. 26, fazendo uso da indução estrutural do conjunto $\mathcal{T}(\Sigma)$. □

2.3.2 Atribuição, variedade e álgebra livre

Os conceitos fornecidos a seguir fazem-se necessários para a formalização de modelos para as lógicas equacionais discutidas no Capítulo 3.

Definição 2.3.4 Seja $\mathcal{A} = \langle A, \mathcal{F}_A \rangle$ uma $\mathcal{F}$-álgebra com conjunto suporte A, e $\mathcal{X}$ um conjunto cujos elementos são chamados variáveis. Uma **atribuição** de $\mathcal{X}$ em $\mathcal{A}$ é um mapeamento total ρ de $\mathcal{X}$ em A.

Uma atribuição ρ de $\mathcal{X}$ em $\mathcal{A}$ se estende para um único[4] Σ-homomorfismo ν de $\mathcal{T}(\mathcal{F}, \mathcal{X})$ em A definido indutivamente da seguinte maneira:

$$\begin{array}{rcll} \nu(x) & = & \rho(x) & \text{para } x \in \mathcal{X}; \\ \nu(c) & = & c_A & \text{para } c \in \mathcal{F} \text{ e } aridade(c) = 0; \\ \nu(f(t_1, ..., t_n)) & = & f_A(\nu(t_1), ..., \nu(t_n)) & \text{para } f \in \mathcal{F} \text{ e } aridade(f) \neq 0. \end{array}$$

Antes da definição de variedade, serão formalizadas as noções de imagem homomórfica, sub-álgebra e produto direto:

Definição 2.3.5 Dada uma $\mathcal{F}$-álgebra $\mathcal{A} = \langle A, \mathcal{F}_A \rangle$, uma $\mathcal{F}$-álgebra $\mathcal{B} = \langle B, \mathcal{F}_B \rangle$ é **imagem homomórfica** de $\mathcal{A}$ se existe um Σ-homomorfismo de A em B.

[4] A unicidade de ν é garantida pelo Teorema 2.3.1.

Definição 2.3.6 Sejam $\mathcal{A} = \langle A, \mathcal{F}_A \rangle$ e $\mathcal{B} = \langle B, \mathcal{F}_B \rangle$ duas $\mathcal{F}$-álgebras. Diz-se que $\mathcal{A}$ é uma **sub-álgebra** de $\mathcal{B}$ se as seguintes condições são satisfeitas:

1. $A \subseteq B$;
2. se $a_1, ..., a_n \in A$ então $f_A(a_1, ..., a_n) = f_B(a_1, ...a_n)$.

Em outras palavras, $\mathcal{A}$ é uma sub-álgebra de $\mathcal{B}$ se B é fechado sob todas as operações em $\mathcal{F}_A$.

Definição 2.3.7 Sejam $\mathcal{A} = \langle A, \mathcal{F}_A \rangle$ e $\mathcal{B} = \langle B, \mathcal{F}_B \rangle$ duas $\mathcal{F}$-álgebras. O **produto direto** de $\mathcal{A}$ e $\mathcal{B}$, denotado por $\mathcal{A} \times \mathcal{B}$ é definido como sendo a $\mathcal{F}$-álgebra $\mathcal{P} = \langle P, \mathcal{F}_P \rangle$, satisfazendo as seguintes condições:

1. $P = A \times B$[5];
2. se $(a_1, b_1), ..., (a_n, b_n) \in P$ então
   $f_P((a_1, b_1), ..., (a_n, b_n)) = (f_A(a_1, ..., a_n), f_B(b_1, ...b_n))$.

É possível agora, introduzir a definição de variedade:

Definição 2.3.8 Uma classe não-vazia $\mathcal{C}$ de $\mathcal{F}$-álgebras é chamada uma **variedade** quando é fechada sobre as operações de sub-álgebra, imagem homomórfica, e produto direto. Uma variedade tem a interessante propriedade de ter uma representante canônica chamada álgebra livre.

Nota: Conforme Beeson [Bee85], pode-se afirmar que:

1. Toda categoria equacional[6] é uma variedade e toda variedade pode ser definida equacionalmente;
2. Seja $\langle \mathcal{T}(\mathcal{F}, \mathcal{X}), \mathcal{F}_{\mathcal{T}(\mathcal{F},\mathcal{X})} \rangle$ a $\mathcal{F}$-álgebra dos termos. Seja E um conjunto de equações sobre $\langle \mathcal{T}(\mathcal{F}, \mathcal{X}), \mathcal{F}_{\mathcal{T}(\mathcal{F},\mathcal{X})} \rangle$. E seja $\mathcal{C}$ uma variedade de $\mathcal{F}$-álgebras. Então E é correto e completo com respeito a $\mathcal{C}$ sss $\langle \mathcal{T}(\mathcal{F}, \mathcal{X}), \mathcal{F}_{\mathcal{T}(\mathcal{F},\mathcal{X})} \rangle /_{=_E}$ é inicial em $\mathcal{C}$.

Definição 2.3.9 Seja $\mathcal{C}$ uma classe não-vazia de $\mathcal{F}$-álgebras e $\mathcal{X}$ um conjunto de variáveis. Uma **$\mathcal{F}$-álgebra $\mathcal{C}$-livre** (também chamada simplesmente de **álgebra livre** sobre $\mathcal{X}$ é qualquer $\mathcal{F}$-álgebra $\mathcal{L} = (L, \mathcal{F}_L)$ tal que:

[5] *P é o produto cartesiano de A e B.*

[6] Uma categoria equacional é uma categoria de Σ-álgebras que satisfazem um determinado conjunto de equações, como por exemplo, as regras de dedução equacional apresentadas no capítulo 3, Figura 3.1.

- $\mathcal{L} \in \mathcal{C}$,
- $\mathcal{X} \subseteq L$,
- para qualquer $\mathcal{F}$-álgebra $\mathcal{A} = \langle A, \mathcal{F}_A \rangle$ em $\mathcal{C}$ e qualquer atribuição $\nu : \mathcal{X} \to A$, existe um homomorfismo $\phi : L \to A$ tal que ϕ e ν concordam sobre $\mathcal{X}$, ou seja, são tais que $\forall x \in \mathcal{X}, \phi(x) = \nu(x)$.

  Isto pode ser ilustrado pelo diagrama:

$$\begin{array}{ccc} \mathcal{X} & \xrightarrow{\subseteq} & L \\ {\scriptstyle \nu}\downarrow & & \downarrow{\scriptstyle \theta} \\ A & \xrightarrow[Id_A]{} & A \end{array}$$

É fácil perceber que, quando existe uma álgebra livre sobre um conjunto $\mathcal{X}$, ela é única, a menos de isomorfismo.

Definição 2.3.10 Uma álgebra $\mathcal{I} = \langle I, \mathcal{F}_I \rangle$ numa classe $\mathcal{C}$ de $\mathcal{F}$-álgebras é **$\mathcal{C}$-inicial** (também chamada **álgebra inicial**) se para qualquer álgebra $\mathcal{A} = \langle A, \mathcal{F}_A \rangle$ em $\mathcal{C}$, há somente um único homomorfismo $\theta : I \to A$.

Nota: Por definição, a álgebra inicial também pode ser vista como a álgebra livre sobre o conjunto vazio de variáveis. Quando ela existe, evidentemente é única, a menos de isomorfismo. Note-se, ainda, que o Teorema 2.3.1 poderia ser reformulado como: *A álgebra dos termos $\mathcal{T}(\Sigma)$ é inicial na classe de todas as Σ-álgebras.*

Um conhecido e importante teorema, com respeito às definições dadas acima, foi enunciado por Birkhoff [Bir35]:

Teorema 2.3.2 *Para qualquer conjunto de variáveis $\mathcal{X}$, a álgebra livre (então a inicial também) de qualquer variedade $\mathcal{C}$ sempre existe.*

2.3.3 Álgebras heterogêneas

A noção de álgebra foi apresentada na seção anterior simplesmente como um conjunto, chamado conjunto suporte da álgebra, munido de uma família de operações sobre esse conjunto. Uma álgebra heterogênea consiste de uma família de conjuntos suporte, indexada por um conjunto de *sorts*, e uma família de operações sobre tais conjuntos, também convenientemente indexada. Formalmente:

Definição 2.3.11 Dada uma assinatura heterogênea $\Sigma = \langle S, TF, PF\rangle$, uma **Σ-álgebra (heterogênea)** é um par $\langle A, \Sigma_A\rangle$, onde

i) A é uma família de conjuntos suporte $\{A_s | s \in S\}$;

ii) Σ_A é um conjunto de funções $\{f_A | f \in TF \cup PF\}$, tal que se o perfil de f é dado por $f_{s_1 \ldots s_n, s}$, então f_A é uma função de $A_{s_1} \times \ldots \times A_{s_n} \to A_s$. A função f_A é chamada de **interpretação** de f, também podendo ser denotada por $\iota(f)$.

A generalização para álgebras heterogêneas é muito natural para a ciência da computação. Por exemplo, considerando o conjunto de *sorts* $S = \{real, int, bool\}$, uma álgebra $\mathcal{A}$ para esta assinatura teria como conjuntos suportes os elementos da família $\{A_{real}, A_{int}, A_{bool}\}$, munida com algumas operações tais como $exp : A_{real} \times A_{int} \to A_{real}$ ou $cond : A_{bool} \times A_{real} \times A_{real} \to A_{real}$, que poderiam ser a exponenciação ou condicional, respectivamente.

Definição 2.3.12 Seja a assinatura $\Sigma = \langle S, TF, PF\rangle$, e sejam as Σ-álgebras $\mathcal{A} = \langle A, \mathcal{F}_A\rangle$ e $\mathcal{B} = \langle B, \mathcal{F}_B\rangle$. Um **Σ-homomorfismo** θ de A em B é uma família de funções $\{\theta_s : A_s \to B_s | s \in S\}$ tal que:

$$\theta_s(f_A(a_1, ..., a_n)) = f_B(\theta_{s_1}(a_1), ..., \theta_{s_n}(a_n)). \tag{2.8}$$

para $\langle a_1, ..., a_n\rangle \in A_{s_1} \times \ldots \times A_{s_n}$ e $f_{s_1 \ldots s_n, s} \in TF \cup PF$.

2.4 Substituições

2.4.1 Definições e propriedades elementares

Definição 2.4.1 A aplicação de contexto (*replacement*), vista na Definição 2.2.21, possui um caso particular, de especial importância, que é a **substituição** (*substitution*) univocamente definida pelo mapeamento de variáveis a termos, i.e. $\rho : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$. Deve-se notar que, devido à existência de uma única extensão homomórfica de $\rho : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ para $\nu : \mathcal{T}(\mathcal{F}, \mathcal{X}) \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ (ver [Gog77]), uma *substituição* é uma aplicação de $\mathcal{T}(\mathcal{F}, \mathcal{X})$ em $\mathcal{T}(\mathcal{F}, \mathcal{X})$ determinada univocamente pela imagem de suas variáveis. Escreve-se $\{x_1 \mapsto t_1, ..., x_n \mapsto t_n\}$ quando há somente variáveis finitamente diversas não mapeadas em si mesmas. Além disso, é comum na literatura utilizar-se σ no lugar de ρ e ν, ou seja, é comum o abuso de linguagem $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ e $\sigma : \mathcal{T}(\mathcal{F}, \mathcal{X}) \to \mathcal{T}(\mathcal{F}, \mathcal{X})$. Mantendo essa notação, a aplicação de uma substituição $\sigma = \{x_1 \mapsto t_1, ..., x_n \mapsto t_n\}$ a um termo é recursivamente definida como segue:

se t é uma variável x_i para algum $i = 1, .., n$, então

$$\sigma(t) = t_i,$$

se t é uma variável $x \neq x_i$ para todo $i = 1, ..., n$, então

$$\sigma(t) = t,$$

se t é um termo $f(u_1, ..., u_k)$ com $u_1, ..., u_k \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ e $f \in \mathcal{F}$, então

$$\sigma(t) = f(\sigma(u_1), ..., \sigma(u_k)).$$

Exemplo 2.4.1 Aplicando $\sigma = \{x \mapsto f(a, z)\}$ ao termo $t = g(a, g(x, x))$, resulta no termo $\sigma(t) = g(a, g(f(a, z), f(a, z)))$.

Definição 2.4.2 O **domínio de uma substituição** σ é o conjunto de variáveis que não são trivialmente mapeadas em si mesmas:

$$\mathcal{D}om(\sigma) = \{x | x \in \mathcal{X} \ e \ \sigma(x) \neq x\}$$

e o conjunto de variáveis introduzidas por σ é chamado de **imagem**, definido por:

$$\mathcal{R}an(\sigma) = \bigcup_{x \in \mathcal{D}om(\sigma)} \mathcal{V}ar(\sigma(x))$$

Quando $\mathcal{R}an(\sigma) = \emptyset$, σ é chamada **substituição fechada**. A substituição cujo domínio é vazio é denotada por Id.

$\sigma_{|W}$ denota a restrição da substituição σ ao subconjunto $W \subseteq \mathcal{X}$, definida como:

$$\begin{array}{rcll} \sigma_{|W} & = & \sigma(x) & \text{se } x \in W \\ & = & x & \text{senão.} \end{array}$$

Restrições se estendem para conjuntos: se S é um conjunto de substituições, então:

$$S_{|W} = \{\sigma_{|W} | \sigma \in S\}.$$

A **composição de substituições** α e β é denotada por "$\beta \circ \alpha$" ou "$\beta.\alpha$" ou simplesmente pela justaposição "$\beta\alpha$" e definida usualmente como $\beta\alpha(x) = \beta(\alpha(x))$. O conjunto de todas as substituições de $\mathcal{T}(\mathcal{F}, \mathcal{X})$ é denotado por $\mathcal{S}ubst(\mathcal{F}, \mathcal{X})$ ou $\mathcal{S}ubst$ quando $\mathcal{F}$ e $\mathcal{X}$ estão claros no contexto.

A **adição** (união) de duas substituições σ e ρ também pode ser definida quando seus domínios são disjuntos:

$$(\sigma + \rho)(x) = \begin{array}{ll} \sigma(x) & \text{se } x \in \mathcal{D}om(\sigma) \\ \rho(x) & \text{se } x \in \mathcal{D}om(\rho) \\ x & \text{se } x \notin \mathcal{D}om(\sigma) \cup \mathcal{D}om(\rho) \end{array}$$

No que segue, define-se o que são substituições idempotentes. Tais substituições são bastante importantes, uma vez que possuem propriedades que tornam a definição de conceitos muito mais simples e facilitam a prova de propriedades.

Definição 2.4.3 Uma substituição σ é **idempotente** se $\sigma.\sigma = \sigma$.

As substituições idempotentes também podem ser caracterizadas pelos seus domínios, sendo possível provar que σ é idempotente sss $\mathcal{R}an(\sigma) \cap \mathcal{D}om(\sigma) = \emptyset$.

Definição 2.4.4 Seja ξ a substituição finita $\{x_1 \mapsto y_1, ..., x_n \mapsto y_n\}$, onde $y_1, ..., y_n$ são variáveis distintas[7]. Então ξ é chamada uma **permutação** se $\mathcal{R}an(\xi) = \mathcal{D}om(\xi)$, e ξ é **renomeação** se $\mathcal{R}an(\xi) \cap \mathcal{D}om(\xi) = \emptyset$; Seja uma substituição ξ^{-1}, diz-se que ξ^{-1} é **inversa** de ξ se $\xi.\xi^{-1} = Id = \xi^{-1}\xi$.

Exemplo 2.4.2 [KK01] *A substituição $\sigma = \{x \mapsto f(a, z)\}$ não é uma renomeação, mas $\xi = \{x \mapsto y_1, z \mapsto y_2\}$ sim. A substituição $\mu = \{x \mapsto y, y \mapsto x\}$ é uma permutação, mas não uma renomeação. Note-se que $\xi' = \{y_1 \mapsto x, y_2 \mapsto z\}$ não é a inversa de ξ uma vez que $\xi.\xi'(x) = \xi(x) = y_1$.*

2.4.2 Inclusão de termos

Definição 2.4.5 Um termo t' é uma **instância** de um termo t se $t' = \sigma(t)$ para alguma substituição σ; neste caso, escreve-se $t \leq t'$ e diz-se que t é **mais geral** do que t' ou que t **inclui** t', ou ainda, que t é **menos específico** do que t'. A substituição σ é um **casamento** de t a t'. A relação "$\leq$" é uma quasi-ordem[8] sobre o conjunto dos termos chamada **inclusão**, onde a equivalência associada "$\equiv$" e a ordem estrita "$<$" são respectivamente chamadas **equivalência de inclusão** e **inclusão estrita**.

Deve-se notar que a ordem de inclusão não é preservada pela aplicação de contexto, ou seja:

$$t \leq t' \not\Rightarrow f(t_1, ..., t, ..., t_n) \leq f(t_1, ..., t', ..., t_n),$$

[7] Ou seja, de acordo com a definição de substituição, $\xi : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ é uma função injetora.

[8] A rigor, o termo quasi-ordem indica uma relação transitiva e anti-reflexiva (veja [RW03]). Ao longo desta pesquisa, porém, o termo quasi-ordem será usado como sinônimo de pré-ordem, i.e. uma relação transitiva e reflexiva, o que é comum na literatura de sistemas de reescrita (veja [KK01] e [BN98]).

conforme o seguinte exemplo: $x \leq a$ mas $f(x,x) \not\leq f(x,a)$.

Também não é preservada pela substituição, ou seja:

$$t \leq t' \nRightarrow \sigma(t) \leq \sigma(t'),$$

conforme o seguinte exemplo: $x \leq a$ mas $(x \mapsto b)x \not\leq (x \mapsto b)a$, onde $(x \mapsto b)x$ e $(x \mapsto b)a$ abreviam $\sigma(x)$ e $\sigma(a)$, respectivamente, com $\sigma = \{(x \mapsto b)\}$.

Definição 2.4.6 A **ordem de encompassamento** denotada por $\sqsubseteq$ é definida por $t \sqsubseteq t'$ se um sub-termo de t' é uma instância de t[9], ou seja, para $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ e algum sub-termo s de t':

$$t \sqsubseteq t' \Leftrightarrow \exists \sigma \in \mathcal{S}ubst, \sigma(t) = s$$

A ordem estrita associada é denotada por $\sqsubset$.

Neste sentido, a *inclusão* é um caso particular da ordem de *encompassamento*.

2.4.3 Inclusão de substituição

A inclusão se estende para substituições da seguinte forma:

Definição 2.4.7 Uma substituição τ é uma **instância** em $V \subseteq \mathcal{X}$ de uma substituição σ, denotada por $\sigma \leq_V \tau$, lê-se σ é **mais geral sobre** V **do que** τ, quando:

$$\sigma \leq_V \tau \Leftrightarrow \exists \rho, \forall x \in V, \tau(x) = \rho(\sigma(x)).$$

Outra notação utilizada é $\tau =_V \rho\sigma$. A relação "$\leq_V$" é uma quasi-ordem[10] em $\mathcal{S}ubst$, também chamada **inclusão**. V é omitido quando igual a $\mathcal{X}$.

Analogamente a termos, a equivalência $\equiv_V$ e a ordem estrita $<_V$ em substituições são respectivamente chamadas **equivalência de inclusão** e **inclusão estrita**. É fácil observar que σ e τ são equivalentes na inclusão em V sss ρ é um mapeamento um para um de $\mathcal{R}an(\sigma_{|V})$ para $\mathcal{R}an(\tau_{|V})$.

A menos de renomeação, a ordem de inclusão em substituições é bem-fundada, ou seja, para qualquer subconjunto $S \subseteq \mathcal{S}ubst$, sempre haverá uma substituição mais geral σ.

[9] Pode-se interpretar ingenuamente $t \sqsubseteq t'$ como "t é instanciado 'dentro' de t'", ou ainda, como "t abstrai um sub-termo de t'".

[10] Ou seja, "$\leq_V$" é uma relação transitiva e reflexiva.

2.5 Reescrita e dedução equacional

Dedução equacional e reescrita baseiam-se na substituição de iguais por iguais. Começa-se com um certo termo t_0 sobre uma dada assinatura Σ, e então aplica-se equações $e_1, ..., e_k$. Cada aplicação destas equações gera novos termos $t_1, t_2, ..., t_k$, onde cada termo é igual a todos os termos anteriores. Isto pode ser escrito como

$$\begin{array}{ll} t_0 & \\ & = \quad \{e_1\} \\ t_1 & \\ & = \quad \{e_2\} \\ \ldots & \\ & = \quad \{e_k\} \\ t_k & \end{array}$$

De fato, uma equação da forma $t = t'$ é uma *regra de reescrita* sss o conjunto de variáveis que ocorre em t' é um subconjunto do conjunto de variáveis que ocorre em t. Um conjunto finito de regras de reescrita sobre uma assinatura Σ é chamado *sistema de reescrita de ($\Sigma-$)termos*. Tais noções serão formalizadas posteriormente, no Capítulo 4.

Esta seção será finalizada com um exemplo, ilustrando o que vem a ser "aplicar" uma equação a um termo:

Exemplo 2.5.1 [GM96] Dada uma assinatura $\Sigma = \langle \{Nat\}, \{0, s, +, *\} \rangle$, onde $0_{w,s} = (\Lambda, Nat)$, $s_{w,s} = (Nat, Nat)$, $+_{w,s} = (NatNat, Nat)$ e $*_{w,s} = (NatNat, Nat)$, seja t_0 o termo "$s(0) + (s(0) * s(0))$" e seja e_1 a equação $(\forall X)s(0) * X = X$. Então, aplica-se e_1 ao sub-termo "$(s(0) * s(0))$" associando-se à variável X no lado esquerdo da equação a constante "$s(0)$" no sub-termo, obtendo "$s(0)$" para o valor correspondente do lado direito. Este deve ser agora colocado no contexto do termo original "$s(0)+(s(0)*s(0))$", obtendo "$s(0)+s(0)$" como resultado final. Assim, o procedimento geral é associar o lado esquerdo da equação a um sub-termo do termo dado, e então substituir aquele sub-termo pela correspondente instância da substituição do lado direito.

Neste capítulo, foram abordados conceitos básicos de linguagens formais, sob três pontos de vista: axiomático, denotacional e computacional. Os dois primeiros serão diretamente aplicados no capítulo a seguir, onde serão estudados aspectos sintáticos (axiomáticos) e semânticos (denotacionais) da lógica equacional. O terceiro, por sua vez, possui singular importância para esta pesquisa, e será enfatizado (implícita ou explicitamente) ao longo de todo o Capítulo 4.

Capítulo 3

Apresentações da Lógica Equacional

Neste capítulo, os conceitos básicos de linguagens formais e Σ-álgebras definidos no capítulo anterior, tais como modelos e sistemas de dedução, serão aplicados no contexto da lógica equacional, em três apresentações diferentes: Lógica equacional homogênea, lógica equacional heterogênea e lógica equacional condicional.

A abordagem apresentada aqui segue de [KK01]. Veja também [Bra03], que faz uso de uma generalização da lógica equacional ordenada heterogênea, chamada lógica equacional de pertinência, para descrever uma lógica de reescrita.

3.1 Lógica equacional homogênea

Na lógica equacional, as fórmulas são construídas a partir de termos de primeira ordem e do predicado de igualdade. Em outras palavras, as fórmulas atômicas possuem a estrutura "$t = t'$", onde t e t' são os termos mencionados.

Definição 3.1.1 Um par de termos $\{l, r\}$ é chamado **equação** ou **axioma equacional** ou **igualdade**, e denotado por $(l = r)$. Assume-se que as variáveis de um axioma equacional são quantificadas universalmente. Quando a quantificação deve ser explícita, escreve-se $(\forall X, l = r)$, onde $\mathcal{V}ar(l) \cup \mathcal{V}ar(r) \subseteq X$.

3.1.1 Sintaxe

A partir de um conjunto de axiomas, novas igualdades podem ser deduzidas através de regras de inferência.

Um sistema de dedução para equações é dado na Figura 3.1. O axioma da reflexividade indica que $(t = t)$ não precisa de premissa para ser deduzido.

1. Reflexividade		$\vdash$	$(t = t)$
2. Simetria	$(t = t')$	$\vdash$	$(t' = t)$
3. Transitividade	$(t = t'), (t' = t'')$	$\vdash$	$(t = t'')$
4. Congruência	$\{(t_i = t'_i) \mid i = 1, ..., n\}$	$\vdash$	$(f(t_1, ..., t_n) = f(t'_1, ..., t'_n))$ $\forall f \in \mathcal{F}_n$
5. Substitutividade	$(t_1 = t_2)$	$\vdash$	$(\sigma(t_1) = \sigma(t_2))$ se $\sigma \in \mathcal{S}ubst$

Figura 3.1: Regras de dedução equacional [KK01]

Definição 3.1.2 Dado um conjunto de axiomas equacionais $E = \{\{l, r\} | l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})\}$, sobre o conjunto de termos $\mathcal{T}(\mathcal{F}, \mathcal{X})$, a **teoria equacional de** E, denotada $\mathcal{TH}(E)$, é o conjunto de igualdades que podem ser obtidas, partindo de E, aplicando-se as regras de inferência dadas na Figura 3.1.

Notação: Escreve-se $E \vdash s = t$, se $(s = t) \in \mathcal{TH}(E)$.

Um sistema de inferência mais compacto é a assim chamada **substituição de iguais por iguais**:

Definição 3.1.3 Dado um conjunto E de axiomas, escreve-se $s \leftrightarrow_E t$ se $s_{|\omega} = \sigma(l)$ e $t = s[\sigma(r)]_\omega$ para alguma posição ω em $\mathcal{D}om(s)$, substituição σ e igualdade $l = r$ (ou $r = l$) em E.

Exemplo 3.1.1 Se $E = \{(f(x, x) = x)\}$, então $f(f(a, a), b) \leftrightarrow_E f(a, b)$, onde $\sigma(x) = a$.

Definição 3.1.4 A **relação de provabilidade** de E é o fecho transitivo reflexivo da relação simétrica $\leftrightarrow_E$, e é denotada por $\overset{*}{\leftrightarrow}_E$. Tal relação é uma relação de congruência.

Notação: A álgebra quociente[1] do conjunto de termos $\mathcal{T}(\mathcal{F}, \mathcal{X})$ por $\overset{*}{\leftrightarrow}_E$ é denotada por $\mathcal{T}(\mathcal{F}, \mathcal{X})/E$.

O teorema seguinte declara a equivalência entre os dois sistemas de inferência para dedução de igualdade.

Teorema 3.1.1 (Birkhoff) [Bir35]

$$E \vdash s = t \; sss \; s \overset{*}{\leftrightarrow}_E t.$$

[1]Sobre álgebras quocientes, veja [Hen88].

3.1.2 Semântica

Os modelos das teorias equacionais homogêneas são as Σ-álgebras homogêneas, ou $\mathcal{F}$-álgebras, que satisfazem um conjunto de equações E.

A noção de modelo e validade introduzida a seguir vale para o caso particular dos axiomas de igualdade. Esses axiomas são igualdades quantificadas universalmente, da forma $\forall \mathcal{V}ar(l,r).l = r$ [2] e igualdades quantificadas existencialmente, da forma $\forall(\mathcal{V}ar(l,r)\backslash V), \exists V, l = r$ [3] onde $V \subseteq \mathcal{V}ar(l,r)$. Note-se que, por *skolemização*, o último pode sempre ser reduzido ao primeiro. O quantificador universal, em geral, não é mencionado explicitamente.

Definição 3.1.5 Uma $\mathcal{F}$-álgebra $\mathcal{A}$ é um **modelo** de um axioma $s = t$, se para qualquer atribuição ν das variáveis em s e t, $\nu(s) = \nu(t)$. Diz-se também que a igualdade $s = t$ é **válida** em $\mathcal{A}$, denotado por $\mathcal{A} \models s = t$.

Definição 3.1.6 Um $\mathcal{F}$-álgebra $\mathcal{A}$ é um **modelo** de um axioma $\exists V.s = t$, se para qualquer atribuição ν das variáveis em s e t exceto aquelas em V, há uma atribuição μ tal que $\mu\nu(s) = \mu\nu(t)$.

Definição 3.1.7 Seja E um conjunto de axiomas de igualdade. Uma $\mathcal{F}$-álgebra $\mathcal{A}$ é um **modelo** de E se ela é um modelo de todos os axiomas em E.

$\mathcal{M}od(E)$ denota o conjunto dos modelos de E. A classe desses modelos é caracterizada pela noção de variedade, como segue:

Teorema 3.1.2 [KK01] *Uma classe $\mathcal{C}$ de álgebras é a classe de modelos de um conjunto de igualdades E, i.e. $\mathcal{M}od(E)$, sss ela for uma variedade, ou seja, quando for fechada sobre produto direto, imagem homomórfica e sub-álgebra.*

Pode-se provar que $\mathcal{T}(\mathcal{F},\mathcal{X})/E$ é um modelo de E. E mais:

Teorema 3.1.3 [Bir35, KK01, BS81] *Para qualquer conjunto de axiomas E, para quaisquer termos $s,t \in \mathcal{T}(\mathcal{F},\mathcal{X})$, e para qualquer álgebra $\mathcal{A} \in \mathcal{M}od(E)$,*

$$\mathcal{A} \models s = t \text{ sss } \mathcal{T}(\mathcal{F},\mathcal{X})/E \models s = t \text{ sss } E \vdash s = t$$

[2] $\forall \mathcal{V}ar(l,r).l = r$ é uma abreviação para $\forall x \in \mathcal{V}ar(l).\forall y \in \mathcal{V}ar(r).l[x]_\omega = r[y]_{\omega'}$, para alguma posição ω de x e alguma posição ω' de y.

[3] Por exemplo, a equação que expressa a propriedade de elemento inverso aditivo, da forma $\forall x.\exists y.x + y = 0$, pode ser reescrita como $\forall(\mathcal{V}ar(l,r)\backslash V), \exists V, l = r$, onde $l = x + y$, $r = 0$, $\mathcal{V}ar(l,r) = \{x,y\}$ e $V = \{y\}$.

Notação: Escreve-se $s =_E t$ sss $\mathcal{A} \models s = t$, para todo $\mathcal{A} \in \mathcal{M}od(E)$. Graças aos teoremas 3.1.1 e 3.1.3, as notações $s =_E t$ e $s \overset{*}{\leftrightarrow}_E t$ podem ser usadas permutavelmente.

Na classe de álgebras que são modelos de E, a álgebra livre sobre $\mathcal{X}$ é (isomorfa a) a álgebra $\mathcal{T}(\mathcal{F}, \mathcal{X})/E$. Na classe de álgebras que são modelos de E, a álgebra inicial é (isomorfa a) a álgebra $\mathcal{T}(\mathcal{F})/E$.

Álgebras de termos são especialmente interessantes por causa da sua larga representatividade mas esta ainda é limitada à classe de álgebras geradas de termos.

Definição 3.1.8 Uma $\mathcal{F}$-álgebra $\mathcal{A} = (A, \iota(\mathcal{F}))$ é **gerada a partir de termos** quando para todo elemento $a \in A$ existe um termo $t \in \mathcal{T}(\mathcal{F})$ tal que $\iota(t) = a$.

Exemplo 3.1.2 Tomando-se $\mathcal{F} = \{+, *, suc, 0\}$, então a $\mathcal{F}$-álgebra dos naturais $\mathcal{L}$, onde $+, *, suc, 0$ são interpretados com seu significado usual e cujo domínio é o conjunto de todos os naturais $\mathbb{N}$, é gerada a partir de termos.

Nem todas as álgebras são geradas de termos, conforme o exemplo a seguir:

Exemplo 3.1.3 Tomando-se $\mathcal{F} = \{+, *, suc, 0\}$, então a $\mathcal{F}$-álgebra dos reais $\mathcal{R}$, onde $+, *, suc, 0$ são interpretados com seu significado usual e cujo domínio é o conjunto de todos os reais $\mathbb{R}$, não é gerada a partir de termos uma vez que, por exemplo, $\sqrt{2}$ não pode ser expresso finitamente usando somente os operadores $+, *, suc, 0$.

3.2 Satisfatibilidade

Definição 3.2.1 Uma equação $(s = t)$ é **satisfatível** numa álgebra $\mathcal{A}$ se existe uma atribuição ν de valores para as variáveis de s e t para os quais $\nu(s) = \nu(t)$.

Esta definição é um caso particular da definição mais geral de um *problema equacional*, que será dada a seguir:

Definição 3.2.2 Sejam $\mathcal{F}$ uma assinatura, $\mathcal{X}$ um conjunto de variáveis, e $\mathcal{A}$ uma $\mathcal{F}$-álgebra . Um $(< \mathcal{F}, \mathcal{X}, \mathcal{A} >$-$)$**problema equacional** é qualquer conjunto $P = \{(s_i =_{\mathcal{A}} t_i)\}_{i \in I}$ de equações, tal que s_i e t_i são termos em $\mathcal{T}(\mathcal{F}, \mathcal{X})$. Uma **solução** de P é qualquer homomorfismo h de $\mathcal{T}(\mathcal{F}, \mathcal{X})$ para $\mathcal{A}$ tal que $\forall i \in I, h(s_i) = h(t_i)$, ou seja, s_i e t_i são mapeados ao mesmo valor em $\mathcal{A}$ pelo homomorfismo h. Dois problemas equacionais P e P' são chamados **equivalentes** se eles tem o mesmo conjunto de soluções.

O processo de se resolver um problema equacional é chamado de **unificação**: dado um conjunto E de axiomas equacionais, e $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, deseja-se encontrar uma substituição σ tal que $\sigma(s) =_E \sigma(t)$. Existem dois casos especiais: quando a álgebra $\mathcal{A}$ é a própria álgebra de termos $\mathcal{T}(\mathcal{F}, \mathcal{X})$, i.e. $E = \emptyset$ (**unificação sintática**) e quando $\mathcal{A}$ é a álgebra quociente $\mathcal{T}(\mathcal{F}, \mathcal{X})/E$ (**unificação semântica**). Como usual, reserva-se a palavra "unificação" para estes dois casos, e fala-se de "resolução de equações" para os demais.

Um dos mais simples problemas da lógica equacional é decidir, para uma dada teoria equacional $\mathcal{TH}(E)$, se dois termos são E-iguais ou não.

Definição 3.2.3 Sejam $\mathcal{TH}(E)$ uma teoria equacional construída sobre a álgebra dos termos $\mathcal{T}(\mathcal{F}, \mathcal{X})$ e t, t' dois termos. O **problema da palavra** consiste em decidir se a igualdade $t = t'$ vale em $\mathcal{TH}(E)$, isto é, se $E \models t = t'$.

Se a teoria equacional é recursiva[4], então o problema da palavra é semi-decidível [KK01].

3.3 Lógica equacional heterogênea

Na lógica equacional heterogênea, as igualdades são construídas a partir de termos heterogêneos e quantificadas universalmente. Nesta seção, o sistema dedutivo e os modelos, apresentados anteriormente para o caso de assinaturas homogêneas, serão generalizados para assinaturas heterogêneas (veja Seção 2.2.3).

3.3.1 Sintaxe

Em primeiro lugar, segue um conceito de fundamental importância, tanto para o caso homogêneo quanto o heterogêneo, a saber, o conceito de *apresentação* ou *especificação*:

Definição 3.3.1 Uma **apresentação**, também chamada **especificação**, e denotada $SP = (\Sigma, E)$, é dada por uma assinatura Σ, e um conjunto E de igualdades quantificadas universalmente $(\forall X, t = t')$ onde $\mathcal{V}ar(t) \cup \mathcal{V}ar(t') \subseteq X$. (A quantificação pode ser omitida quando $X = \mathcal{V}ar(t) \cup \mathcal{V}ar(t')$).

Exemplo 3.3.1 *:* Seja uma especificação de listas com dois *sorts List* para listas e *Elt* para elementos. Listas são construídas com dois construtores *nil* para a lista vazia e *push* que concatena um elemento a uma lista. Seja também uma operação *conc*, sobre listas, que concatena duas listas e produz uma terceira. A estrutura *List* e a operação *conc* são descritas pela seguinte especificação:

[4] i.e. é possível decidir se um dado axioma equacional pertence ou não à teoria.

$$\begin{array}{rcl} sorts: & & Elt, List \\ nil: & \mapsto & List \\ push: Elt\ List & \mapsto & List \\ conc: List\ List & \mapsto & List \end{array}$$

e as igualdades definem a operação *conc*:

$$\begin{array}{rcl} \forall z : List \quad conc(nil, z) & = & z \\ \forall x : Elt, y : List, z : List \quad conc(push(x, y), z) & = & push(x, conc(z, y)) \end{array}$$

Desta forma, a especificação de listas é dada por $SP = (\Sigma, E)$, onde $\Sigma = \langle \{List, Elt\}, \{nil, push, conc\} \rangle$, cujos símbolos funcionais possuem os perfis $nil_{\lambda,List}$, $push_{EltList,List}$ e $conc_{ListList,List}$, respectivamente; e, ainda, $E = \{(conc(nil, z) = z), (conc(push(x, y), z) = push(x, conc(z, y)))\}$.

As regras de dedução equacional, dadas na Figura 3.1 generalizam para o *framework* heterogêneo desde que não exista nenhum *sort* vazio. Uma análise precisa dos possíveis problemas que podem surgir quando esta hipótese não é satisfeita pode ser encontrada em [MG85].

3.3.2 Semântica

Os modelos para a lógica equacional heterogênea são Σ-álgebras, tal como definido na seção 2.3.3.

Substituições, definidas como mapeamentos σ de variáveis com *sort* em termos heterogêneos tal que se $x : s$ então $\sigma(x) \in \mathcal{T}(\Sigma, \mathcal{X})_s$, induzem Σ-homomorfismos sobre a Σ-álgebra de termos heterogêneos.

A seguir, somente modelos com sorts não-vazios serão considerados. Isto significa que para qualquer modelo $\mathcal{A}$ e qualquer $s \in S$, $\mathcal{A}_s$ é um conjunto não-vazio.

Uma especificação $SP = (\Sigma, E)$ realmente descreve uma classe de álgebras, nominalmente a classe de Σ-álgebras satisfazendo as igualdades E, denotada por $\mathcal{M}od(E)$ ou $ALG(SP)$. $ALG(SP)$ com SP-homomorfismos[5] é uma categoria[6], também denotada por $ALG(SP)$.

É possível demonstrar que a substituição de iguais por iguais $\overset{*}{\leftrightarrow}_E$ em $\mathcal{T}(\Sigma, \mathcal{X})$ é correta e completa para dedução em $ALG(SP)$, i.e.:

[5] A notação SP-homomorfismos denota os Σ-homomorfismos entre os elementos da classe de álgebras $\mathcal{M}od(E)$.

[6] Simplificadamente, uma categoria é uma estrutura algébrica constituída de um conjunto de objetos, e transformações entre os mesmos.

$$t \overset{*}{\leftrightarrow}_E t' \; sss \; \forall \mathcal{A} \in ALG(SP), \mathcal{A} \models (\forall X, t = t').$$

Esta classe $ALG(SP)$ tem uma álgebra inicial denotada por $\mathcal{T}(\Sigma)/E$ ou por $\mathcal{T}_{SP}$-$\mathcal{T}(\Sigma)/E$ construída como a álgebra quociente da álgebra de termos fechados $\mathcal{T}(\Sigma)$ pela congruência $\overset{*}{\leftrightarrow}_E$ gerada por E.

3.4 Lógica equacional condicional

3.4.1 Sintaxe

As fórmulas em questão são as, assim chamadas, *igualdades condicionais*. Uma igualdade condicional não é nada mais do que uma cláusula de Horn[7]. Formalmente:

Definição 3.4.1 Uma **cláusula equacional** é uma cláusula construída apenas com o predicado de igualdade, e denotada por $\Gamma \Rightarrow \Delta$, onde Γ é o antecedente e Δ é o sucedente. Uma **cláusula de Horn equacional** é uma cláusula equacional cujo sucedente é reduzido a uma igualdade. Uma **igualdade condicional** é uma cláusula de Horn equacional, denotada por "$l = r$, se Γ", onde Γ é um conjunto de igualdades. Γ e $l = r$ são respectivamente chamados de **condição** e **conclusão** da igualdade condicional.

O significado de "$l = r$, se Γ" é que l e r são iguais se a condição Γ é satisfeita. Quando é necessário tornar a condição explícita, a igualdade condicional é denotada por "$l = r$ se $(s_1 = t_1 \wedge ... \wedge s_n = t_n)$".

Um **sistema condicional** é um conjunto de igualdades condicionais.

Para qualquer conjunto de axiomas condicionais E, igualdades condicionais, da forma "$s = t$ se Γ", podem ser deduzidas via regras de inferência.

Um sistema de dedução para dedução equacional condicional é dado na Figura 3.2. Neste sistema, quando $\Gamma = \bigwedge_{i=1,...,n} u_i = v_i$ com $n \geq 0, \sigma(\Gamma) = \bigwedge_{i=1,...,n} \sigma(u_i) = \sigma(v_i)$.

Definição 3.4.2 Dado um conjunto de axiomas condicionais E e um conjunto de termos $\mathcal{T}(\Sigma, \mathcal{X})$, a **teoria condicional** de E, denotada $\mathcal{TH}(E)$, é o conjunto de igualdades condicionais que podem ser obtidas, a partir de E, aplicando-se as regras de inferência dadas na Figura 3.2.

[7] Uma cláusula (i.e. uma disjunção de literais) é chamada *cláusula de Horn* se ela contém, no máximo, um literal positivo. Na lógica apresentada nesta seção, uma cláusula será vista como uma conjunção (e não uma disjunção) de literais, seguindo o padrão de [KK01].

1. Reflexividade		$\vdash$	$(t = t)$
2. Simetria	$(t = t')$	$\vdash$	$(t' = t)$
3. Transitividade	$(t = t'), (t' = t'')$	$\vdash$	$(t = t'')$
4. Congruência	$\{(t_i = t'_i) \mid i = 1, ..., n\}$	$\vdash$	$(f(t_1, ..., t_n) = f(t'_1, ..., t'_n))$ se $f \in \mathcal{F}_n$
5. Substitutividade	$(t_1 = t_2)$ se Γ	$\vdash$	$\sigma(t_1) = \sigma(t_2)$ se $\sigma(\Gamma)$ se $\sigma \in \mathcal{S}ubst$
6. Corte	$t_1 = t_2$ se $(\Gamma \wedge t'_1 = t'_2)$, $t'_1 = t'_2$ se Γ'	$\vdash$	$t_1 = t_2$ se $(\Gamma \wedge \Gamma')$

Figura 3.2: Regras de dedução equacional condicional [KK01]

3.4.2 Semântica

Do ponto de vista algébrico, sistemas condicionais e sistemas equacionais têm resultados muito similares[8].

Definição 3.4.3 Seja E um conjunto de axiomas condicionais. Uma álgebra $\mathcal{A}$ é um **modelo** de E se, para qualquer axioma "$l = r,\ se\ (s_1 = t_1 \wedge ... \wedge s_n = t_n)$" em E, e para qualquer atribuição ν de variáveis em $\mathcal{A}$,

$$se\ \forall i \in [1, ..., n], \nu(s_i) = \nu(t_i)\ \ então\ \nu(l) = \nu(r).$$

Diz-se também que o axioma "$l = r,\ se\ (s_1 = t_1 \wedge ... \wedge s_n = t_n)$" é **válido** em $\mathcal{A}$, ou que $\mathcal{A}$ **satisfaz** "$l = r,\ se\ (s_1 = t_1 \wedge ... \wedge s_n = t_n)$", isto é $\mathcal{A} \models s = t$.

Dado um conjunto de axiomas condicionais E, $\mathcal{A}$ é um modelo de E se $\mathcal{A}$ satisfaz todos os axiomas condicionais em E.

Definição 3.4.4 Um conjunto de axiomas condicionais E é chamado **consistente** se ele tem um modelo, e **inconsistente** ou **insatisfatível** caso contrário. E **implica** C (escreve-se $E \models C$) se todo modelo de E satisfaz C.

$\mathcal{M}od(E)$ é o conjunto de modelos de E. Pode-se provar que existe uma menor congruência sobre $\mathcal{T}(\Sigma, \mathcal{X})$ gerada por E. Esta congruência é obtida como o menor ponto-fixo de uma função contínua definida sobre o reticulado completo de congruências construídas sobre $\mathcal{T}(\Sigma, \mathcal{X})$.

[8] Vendo um axioma condicional como uma cláusula de Horn equacional, tem-se também um outro tipo de semântica, via *interpretações de Herbrand*. Veja [Rin04] pg. 16 e [KK01] pg. 45.

A álgebra quociente $\mathcal{T}(\Sigma, \mathcal{X})/E$ de $\mathcal{T}(\Sigma, \mathcal{X})$ pela congruência gerada por E é um modelo de E. Além disso, o modelo inicial é a álgebra quociente $\mathcal{T}(\Sigma)/E$ de termos fechados pela congruência gerada por E.

Seguindo trabalhos anteriores, um teorema de Birkhoff estabelece a completude de substituição condicional de iguais por iguais. O sistema de dedução para dedução equacional condicional dado na Figura 3.2 é correto e completo com respeito a esta noção de modelos.

Teorema 3.4.1 [KK01] *Para qualquer conjunto de axiomas condicionais E, para qualquer igualdade condicional $s = t$ se Γ,*

$$\mathcal{M}od(E) \models s = t \text{ se } \Gamma \text{ sss } E \vdash s = t \text{ se } \Gamma.$$

Neste capítulo, aspectos axiomáticos e denotacionais foram considerados para a lógica equacional, em suas diversas apresentações. Agora, o foco será direcionado para o aspecto computacional de teorias equacionais. Será apresentada, no capítulo a seguir, uma lógica, chamada *lógica de reescrita*, que impõe certas restrições aos sistemas dedutivos (e seus respectivos modelos) apresentados até aqui, a fim de prover um procedimento de decisão para as teorias em questão.

Capítulo 4

Sistemas de Reescrita de Termos

4.1 Introdução a sistemas de reescrita

Este capítulo introduz os principais conceitos relacionados a sistemas de reescrita de termos, além de descrever a lógica de reescrita cujas regras servirão de base para a teoria formal proposta no Capítulo 6. Além disso, são discutidas as propriedades fundamentais que tais sistemas devem satisfazer, a fim de prover um procedimento de decisão para teorias equacionais, em especial as propriedades de terminação e confluência.

4.1.1 Visão geral

Antes de apresentar uma definição formal, a noção de reescrita será introduzida com duas charadas clássicas, adaptadas de Dershowitz [Der93].

Charada 1: Ilha do Camaleão. Os camaleões nesta estranha ilha vêm em três cores — vermelho, amarelo e verde — e vagueiam continuamente. Sempre que dois camaleões de cores diferentes se encontram, ambos mudam para a terceira cor. Supondo que haja 15 camaleões vermelhos, 14 amarelos, e 13 verdes, podem os seus encontros casuais levarem a um estado estável, todos compartilhando a mesma cor?

Charada 2: Urna Grega. Uma urna contém 150 feijões pretos e 75 brancos. Dois feijões são removidos de cada vez. Se são da mesma cor, um preto é colocado na urna; se são diferentes, o branco é devolvido à urna. O processo é repetido tanto quanto possível. A cor do último feijão na urna é predeterminada? Se for, qual é a cor?

Ambas as charadas fornecem regras para se partir de um estado para outro. A primeira questiona a possibilidade de se chegar a um estado final a partir de

um estado inicial; a última questiona a forma pela qual o estado final se relaciona com o estado inicial.

Um *sistema de reescrita* consiste, em geral, de um conjunto de *regras* para transformar termos.

A *Ilha do Camaleão* pode ser expressa através de seis regras, uma para cada par de camaleões possível:

$$\begin{aligned}
\textit{vermelho amarelo} &\rightarrow \textit{verde verde}\\
\textit{amarelo vermelho} &\rightarrow \textit{verde verde}\\
\textit{verde amarelo} &\rightarrow \textit{vermelho vermelho}\\
\textit{amarelo verde} &\rightarrow \textit{vermelho vermelho}\\
\textit{vermelho verde} &\rightarrow \textit{amarelo amarelo},\\
\textit{verde vermelho} &\rightarrow \textit{amarelo amarelo},
\end{aligned}$$

A *Urna Grega*, por sua vez, pode ser expressa com o seguinte sistema de quatro regras, uma para cada par de feijões possível:

$$\begin{aligned}
\textit{preto preto} &\rightarrow \textit{preto}\\
\textit{branco branco} &\rightarrow \textit{preto}\\
\textit{preto branco} &\rightarrow \textit{branco}\\
\textit{branco preto} &\rightarrow \textit{branco}.
\end{aligned}$$

A noção de sistemas de reescrita será formalizada na seções seguintes. As definições fornecidas foram adaptadas de [BN98, DJ90, Der93, KK01]. A notação utilizada segue de Kirchner [KK01], com pequenas variações.

4.1.2 Raciocínio equacional

Na tentativa de automatizar tanto quanto possível o raciocínio equacional, depara-se com o problema de encontrar um procedimento de decisão para teorias equacionais. À semelhança de Baader e Nipkow [BN98], Kirchner [KK01] ilustra tal problema através do exemplo da teoria dos grupos, definida pelos axiomas abaixo, onde "$*$" é uma operação binária, "e" representa o elemento identidade dessa operação, e $i(x)$ representa o inverso de x:

$$\begin{aligned}
x * e &= x\\
x * i(x) &= e\\
(x * y) * z &= x * (y * z)
\end{aligned}$$

A prova de que "e" também é identidade à esquerda pode ser feita por substituição equacional como segue:

$$\begin{aligned} e * x &= e * (x * e) = e * (x * (i(x) * i(i(x)))) \\ &= e * ((x * i(x)) * i(i(x))) = e * (e * i(i(x))) \\ &= (e * e) * i(i(x)) = e * i(i(x)) = (x * i(x)) * i(i(x)) \\ &= x * (i(x) * i(i(x)) = x * e = x \end{aligned}$$

Automatizar esta prova requer conjecturar qual o axioma correto a ser aplicado em cada passo, em qual direção, e a possibilidade de retrocesso. A idéia de reescrita é suprimir a necessidade de retrocesso, primeiro pelo uso de axiomas (orientados) da esquerda para a direita somente; segundo, fornecendo suficientes axiomas (orientados) para ter o mesmo poder de dedução que originalmente. Por exemplo, decidir a igualdade na teoria dos grupos requer dez axiomas orientados (equivalentes aos três previamente apresentados).

$$\begin{aligned} x * e &\rightarrow x \\ e * x &\rightarrow x \\ x * i(x) &\rightarrow e \\ i(x) * x &\rightarrow e \\ i(e) &\rightarrow e \\ i(i(x)) &\rightarrow x \\ i(x * y) &\rightarrow i(y) * i(x) \\ (x * y) * z &\rightarrow x * (y * z) \\ x * (i(x) * y) &\rightarrow y \\ i(x) * (x * y) &\rightarrow y \end{aligned}$$

Naturalmente com este sistema, provar o teorema anterior se torna óbvio, por causa da existência do axioma orientado "$e * x \rightarrow x$".

4.1.3 Sistemas de reescrita

A idéia central de reescrita consiste em impor direcionalidade no uso de equações.

Definição 4.1.1 Uma **regra de reescrita** é um par ordenado de termos denotado $l \rightarrow r$. Os termos l e r são respectivamente chamados **lado esquerdo** (*left-hand side*) e **lado direito** (*right-hand side*) da regra. Um **sistema de reescrita** ou **sistema de reescrita de termos** é um conjunto (finito ou infinito) de regras de reescrita.

Exemplo 4.1.1 A Lógica Combinatória é um exemplo de uma teoria com um símbolo funcional binário (aplicação), símbolos de constantes S, K, I (combinadores) e variáveis. A teoria é definida por três axiomas que podem ser aplicados como regras de reescrita, usando os seguinte sistema de reescrita LC:

$$\begin{array}{rcl} ((S \cdot x) \cdot y) \cdot z & \rightarrow & (x \cdot z) \cdot (y \cdot z) \\ (K \cdot x) \cdot y & \rightarrow & x \\ (I \cdot x) & \rightarrow & x. \end{array}$$

Geralmente o símbolo "$\cdot$" é omitido para melhor legibilidade.

Uma regra é aplicada substituindo-se uma instância do lado esquerdo pela mesma instância de seu lado direito, mas nunca o inverso, como acontece com a relação de igualdade. Note-se que duas regras são consideradas a mesma se elas somente diferem por uma renomeação de suas variáveis. Um sistema de reescrita R induz uma relação binária nos termos chamada de **relação de reescrita** ou **relação de derivabilidade**.

Definição 4.1.2 Dado um sistema de reescrita R, um termo t **reescreve** para um termo t', o que é denotado por $t \rightarrow_R t'$ se existe:

- uma regra $l \rightarrow r$ de R,
- uma posição ω em t, e
- uma substituição σ, satisfazendo $t_{|\omega} = \sigma(l)$, chamada de um **casamento** (*match*) de l para $t_{|\omega}$, tal que $t' = t[\omega \hookleftarrow \sigma(r)]$.

Exemplo 4.1.2 Sejam $R = \{x+y \rightarrow y+x\}$, $t = 4 \times (0+z)$ e $t' = 4 \times (z+0)$, e tomando $\sigma = \{x \mapsto 0, y \mapsto z\}$. Verifica-se que $t \rightarrow_R t'$, pois $t_{|2} = 0+z = \sigma(x+y)$ e $t[2 \hookleftarrow \sigma(y+x)] = 4 \times (z+0) = t'$.

Quando t reescreve para t' com uma regra $l \rightarrow r$ e uma substituição σ, será sempre assumido que variáveis de l e t são disjuntas. Assim $\mathcal{D}om(\sigma) \cap \mathcal{R}an(\sigma) = \emptyset$. Isto não é uma restrição, já que variáveis de regras podem ser renomeadas sem perda de generalidade.

Notação: Quando a regra, a substituição e a posição necessitam ser explicitadas, denota-se um passo de reescrita por

$$t \rightarrow_R^{\omega,\sigma,l\rightarrow r} t'.$$

Definição 4.1.3 O sub-termo $t_{|\omega}$ onde o passo de reescrita é aplicado é chamado de **redex**. Um termo que não tem redex é chamado de **irredutível** para R ou **forma normal** de R. A forma irredutível de t é denotada por $t \downarrow_R$. Por fim, diz-se que um termo t **possui** uma forma normal quando t se reduz para uma forma normal, em um número finito ou infinito de passos de reescrita.

Notação: No que segue, com relação a $\rightarrow_R$:

$\overset{=}{\rightarrow}_R$ indica o fecho reflexivo;
$\overset{+}{\rightarrow}_R$ indica o fecho transitivo;
$\leftrightarrow_R$ indica o fecho simétrico;
$\overset{*}{\rightarrow}_R$ indica o fecho reflexivo-transitivo;
$\overset{*}{\leftrightarrow}_R$ indica o fecho reflexivo-simétrico-transitivo.

Definição 4.1.4 Uma **derivação de reescrita** é qualquer sequência de passos de reescrita

$$t_1 \rightarrow_R t_2 \rightarrow_R ...$$

A **relação de derivabilidade** $\overset{*}{\leftrightarrow}_R$ é definida nos termos por $t \overset{*}{\leftrightarrow}_R t'$ se existe uma derivação de reescrita de t em t' ou vice-versa. Se a derivação contém ao menos um passo, é denotada por $\overset{+}{\rightarrow}_R$.

Continuando o Exemplo 4.1.1, da Lógica Combinatória, pode-se derivar

$$S(KS)Kxyz \rightarrow_{LC} KSx(Kx)yz \rightarrow_{LC} S(Kx)yz \rightarrow_{LC} Kxz(yz) \rightarrow_{LC} x(yz)$$

onde o operador "$\cdot$" foi omitido. Abreviar $S(KS)K$ por B, estabelece que $Bxyz \overset{*}{\rightarrow}_{LC} x(yz)$ e define Bxy como a composição $x \circ y$ onde x e y são variáveis representando funções.

As várias relações definidas a partir de R têm em comum serem fechadas sob a aplicação de contexto[1] e a substituição:

Definição 4.1.5 Uma relação binária $\rightarrow$ sobre um conjunto de termos $\mathcal{T}(\Sigma, \mathcal{X})$ é **termo-fechada** ou **monotônica** se ela é fechada sob aplicações de contexto:

$$\forall s, t, u \in \mathcal{T}(\Sigma, \mathcal{X}) \forall \omega \in \mathcal{D}om(u).s \rightarrow t \Rightarrow u[s]_\omega \rightarrow u[t]_\omega,$$

e sob substituições:

$$\forall s, t \in \mathcal{T}(\Sigma, \mathcal{X}), \forall \sigma \in \mathcal{S}ubst.s \rightarrow t \Rightarrow \sigma(s) \rightarrow \sigma(t).$$

Em geral, as principais propriedades de um sistema de reescrita de termos são definidas a partir das propriedades das relações binárias (relações de reescrita) induzidas por suas regras. Tais propriedades serão resumidas na definição abaixo, e serão particularmente discutidas nas seções 4.2 e 4.3.

[1] o que implica serem fechadas sob os símbolos $f \in \Sigma$.

Definição 4.1.6 Uma relação de reescrita $\rightarrow_R$, sobre $\mathcal{T}(\Sigma, \mathcal{X})$, é chamada

- **Church-Rosser**[2] sss
  $\forall t, t' \; . \; t \overset{*}{\leftrightarrow}_R t'$ sss $\exists s \; . \; t \overset{*}{\rightarrow}_R s \overset{*}{\leftarrow}_R t'$;
- **Confluente** sss
  $\forall t, t', t'' \; . \; t' \overset{*}{\leftarrow}_R t \overset{*}{\rightarrow}_R t'' \Rightarrow \exists s \; . \; t' \overset{*}{\rightarrow}_R s \overset{*}{\leftarrow}_R t''$;
- **Terminante** sss
  não existe nenhuma cadeia descendente infinita $t_0 \rightarrow_R t_1 \rightarrow_R ...$;
- **Convergente** sss
  ela é confluente e terminante;
- **Normalizante** sss
  todo elemento possui uma forma normal.

Nota: Deve-se chamar a atenção para o fato de que as propriedades acima não são especificamente características de relações de reescrita, mas de relações binárias em geral. É possível definir-se abstratamente a noção de "redução" e, a partir daí, definirem-se as propriedades acima para sistemas de redução abstratos, de forma genérica.

A seguir, será formalizada a importante noção de *ordem de reescrita*, a qual será usada, posteriormente, como base para a definição de *ordem de redução* e *ordem de simplificação*, ambas utilizadas na prova da terminação de sistemas de reescrita, como será visto na Seção 4.2.

Definição 4.1.7 Uma relação binária $\rightarrow$ sobre um conjunto de termos $\mathcal{T}(\Sigma, \mathcal{X})$ é uma **ordem de reescrita** se ela é transitiva, irreflexiva e termo-fechada.

Exemplo 4.1.3 A relação binária induzida pela regra $f(f(x)) \rightarrow x$ é claramente termo-fechada, visto que a mesma é fechada sob aplicações de contexto (por exemplo, $g(a, f(f(x)), b) \rightarrow g(a, x, b)$), e sob substituições (por exemplo, aplicando $\sigma = \{x \mapsto a\}$, então $f(f(a)) \rightarrow a$.

Lema 4.1.1 [KK01] As relações $\rightarrow_R, \leftarrow_R, \leftrightarrow_R, \overset{*}{\rightarrow}_R, \overset{*}{\leftarrow}_R, \overset{*}{\leftrightarrow}_R, \overset{+}{\rightarrow}_R, \overset{+}{\leftarrow}_R$ e $\overset{+}{\leftrightarrow}_R$ são fechadas sob a aplicação de contexto e substituições sobre $\mathcal{T}(\Sigma, \mathcal{X})$. A relação $\overset{+}{\rightarrow}_R$ é uma ordem de reescrita.

Definição 4.1.8 Para um sistema de reescrita $R = \{l_i \rightarrow r_i\}_{i \in I}$, denota-se $=_R$ a teoria equacional gerada pelo conjunto de axiomas equacionais $E = \{l_i = r_i\}_{i \in I}$.

[2] A. Church e J. Rosser provaram que o λ-cálculo tem esta propriedade.

Uma fácil caracterização de dois termos equivalentes módulo $=_R$ pode ser dada usando $\rightarrow_R$:

Lema 4.1.2 [KK01] Seja $R = \{l_i \rightarrow r_i\}_{i \in I}$ um sistema de reescrita de termos, então para quaisquer termos t, t', tem-se $t =_R t'$ sss existe uma sequência finita de termos $t = t_0, t_1, ..., t_{n-1}, t_n = t'$ tal que para todo $i \in [1..n-1]$, "$t_i \rightarrow_R t_{i+1}$ ou $t_{i+1} \rightarrow_R t_i$".

4.1.4 Uma lógica de reescrita

Esta introdução geral a sistemas de reescrita será finalizada, agora, com a ilustração de uma lógica de reescrita, a qual permite construir todos os elementos de uma relação de reescrita gerada por um sistema de reescrita de termos R.

Definição 4.1.9 Para um dado conjunto de termos $\mathcal{T}(\Sigma, \mathcal{X})$ e um conjunto de regras R, definem-se os **seqüentes** como as sentenças da forma $t \rightarrow_R t'$ onde t e t' são termos de $\mathcal{T}(\Sigma, \mathcal{X})$.

Uma teoria de reescrita é um conjunto R de regras de reescrita. Cada regra $(l \rightarrow r)$ tem um conjunto finito de variáveis $\mathcal{V}ar(l) \cup \mathcal{V}ar(r) = \{x_1, ..., x_n\}$ que são registradas na notação $(l(x_1, ..., x_n) \rightarrow r(x_1, ..., x_n))$. Uma teoria R acarreta o seqüente $(t \rightarrow t')$, se ele for obtido pela aplicação finita das regras de dedução apresentadas na Figura 4.1.

Lema 4.1.3 [KK01] A regra **Substituição** pode ser derivada das outras quatro regras da lógica de reescrita. Se a regra:

$$\textbf{Simetria} \quad \begin{array}{l} t_1 \rightarrow t_2 \\ \twoheadrightarrow \\ t_2 \rightarrow t_1 \end{array}$$

é adicionada às quatro regras **Reflexividade, Congruência, Troca** e **Transitividade**, claramente se obtém a lógica equacional definida anteriormente, resumida na Figura 3.1. A distância entre a lógica equacional e a lógica de reescrita é realmente bastante grande por causa do requisito muito forte da simetria.

Dependendo do uso de tais regras, podem-se construir diversas relações de reescrita.

Definição 4.1.10 Dado um sistema de reescrita R, um seqüente $t \rightarrow t'$ é:

- uma **R-reescrita de zero passos**, se ele pode ser derivado de R pela aplicação das regras **Reflexividade** e **Congruência** (quando t e t' coincidem).

Reflexividade	$\twoheadrightarrow$
	$(t \to t)$
	se $t \in \mathcal{T}(\Sigma, \mathcal{X})$
Congruência	$t_1 \to t'_1, ..., t_n \to t'_n$
	$\twoheadrightarrow$
	$f(t_1, ..., t_n) \to f(t'_1, ..., t'_n)$
	se $f \in \mathcal{F}_n$
Troca	$t_1 \to t'_1, ..., t_n \to t'_n$
	$\twoheadrightarrow$
	$l(t_1, ..., t_n) \to r(t'_1, ..., t'_n)$
	se $l(x_1, ..., x_n) \to r(x_1, ..., x_n) \in R$
Transitividade	$t_1 \to t_2, t_2 \to t_3$
	$\twoheadrightarrow$
	$t_1 \to t_3$
Substituição	$u(x_1, ..., x_n) \to v(x_1, ..., x_n), t_1 \to t'_1, ..., t_n \to t'_n$
	$\twoheadrightarrow$
	$u(t_1, ..., t_n) \to v(t'_1, ..., t'_n)$

Figura 4.1: Regras de dedução de reescrita [KK01]

- uma **R-reescrita concorrente de um passo**, se ele pode ser derivado de R pela aplicação das regras **Reflexividade**, **Congruência** e no mínimo uma aplicação de **Troca**. Quando **Troca** é aplicada exatamente uma vez, então o seqüente é chamado **R-reescrita sequencial de um passo**.
- uma **R-reescrita concorrente**, se ele pode ser derivado de R pela aplicação finita das regras apresentadas na Figura 4.1.

A relação entre uma R-reescrita sequencial de um passo e a relação de reescrita previamente definida nos termos é apresentada no seguinte lema.

Lema 4.1.4 [KK01] Um seqüente $t \to t'$ é uma R-reescrita sequencial de um passo sss existe uma regra $l \to r$ em R, uma substituição σ e uma ocorrência ω de t tal que $t \to_R^{\omega,\sigma,l\to r} t'$.

Prova: Indução imediata na forma da prova do seqüente $t \to t'$. □

Este resultado se estende a qualquer derivação de reescrita concorrente:

Lema 4.1.5 [KK01] Para cada seqüente $t \to t'$ computado usando a lógica de reescrita relativa a um conjunto de regras R:

- ou $t = t'$,
- ou existe uma corrente de R-reescrita concorrente de um passo:

$$t = t_0 \to t_1 \to ... \to t_{n-1} \to t_n = t'.$$

  Além disso, nesta corrente, cada passo $t_i \to t_{i+1}$ pode ser escolhido sequencialmente.

Prova: Esta também é uma conseqüência da forma da prova do seqüente $t \to t'$. □

Estes últimos resultados mostram a equivalência entre a definição operacional de reescrita e a relação de reescrita sequencial obtida a partir da lógica de reescrita. A principal vantagem do último é precisamente escapar de uma definição operacional de reescrita que necessita manipular a noção de ocorrência, associação e substituição. Isto permite, em particular, simplificar as provas.

4.2 Terminação de sistemas de reescrita

Esta seção resume os principais resultados [BN98, DJ90, Der93, KK01], na prova da terminação de sistema de reescrita de termos, fazendo uso explícito da estrutura de termo, substituições e propriedades de termo. Primeiramente, apresenta-se a noção de *indução bem fundada*, bem como sua relação com a terminação de sistemas de reescrita. Em seguida, o problema da terminação é descrito de forma genérica. Finalmente, dois métodos para prova da terminação são considerados: o primeiro baseado em ordens de redução e o segundo baseado em ordens de simplificação; será mostrado que a segunda abordagem é bastante sintática e mais restritiva (porém, mais poderosa) do que a primeira.

4.2.1 Indução bem fundada

A indução bem fundada é um importante princípio de prova, satisfeito por todas as relações terminantes, como será mostrado mais adiante. Inicialmente, destaque-se o princípio de indução para os números naturais: uma propriedade $P(n)$ é satisfeita por todo número natural n se é possível mostrar que $P(n)$ é satisfeita sob a hipótese de que $P(m)$ é satisfeita para todo número natural m tal que $m \leq n$. A segurança deste princípio de prova está no fato de que não existe nenhuma cadeia infinitamente descendente $m_0 > m_1 > ...$ de números naturais.

O princípio de *indução bem fundada* é uma generalização da indução de $(\mathbb{N}, >)$ para qualquer sistema de reescrita[3]. Formalmente, uma indução bem fundada para sistemas de reescrita é dada por:

Definição 4.2.1 Seja o sistema de reescrita R, e sua relação de reescrita associada $\rightarrow_R$. A **indução bem fundada** de R, segundo uma dada propriedade P, é expressa pela seguinte regra de inferência:

$$(\forall t \in \mathcal{T}(\Sigma, \mathcal{X}) \; . \; (\forall t' \in \mathcal{T}(\Sigma, \mathcal{X}) \; . \; (t \stackrel{+}{\rightarrow}_R t' \Rightarrow P(t'))) \Rightarrow \; P(t)) \Rightarrow \\ \forall t \in \mathcal{T}(\Sigma, \mathcal{X}) \; . \; P(t),$$

denotada por ibf.

Em outras palavras, para provar $P(t)$ para todo $t \in \mathcal{T}(\Sigma, \mathcal{X})$, é suficiente provar $P(t)$ sob a hipótese de que todo "sucessor" t' de t satisfaz P.

Nota: Nesse esquema de indução, o caso base está implícito na própria premissa de ibf. Se $\rightarrow_R$ é terminante, o "caso base" da indução consiste em mostrar que todo termo t, que não possui sucessor, satisfaz $P(t)$, i.e. as formas normais satisfazem P. Fazendo isto, a hipótese $\forall t' \in \mathcal{T}(\Sigma, \mathcal{X}) \; . \; (t \stackrel{+}{\rightarrow}_R t' \Rightarrow P(t'))$ torna-se verdadeira, por vacuidade, e a premissa de ibf degenera para $P(x)$.

Nipkow e Baader [BN98] mostram que a ibf não é correta para qualquer relação $\rightarrow_R$, mas o é para as relações terminantes, através dos seguintes teoremas[4]:

Teorema 4.2.1 [BN98] *Se $\rightarrow_R$ é terminante, então $\rightarrow_R$ satisfaz ibf.*

Prova: Assumindo que $\rightarrow_R$ não satisfaz ibf, ou seja, existe algum P tal que a premissa de ibf é satisfeita mas a conclusão não, i.e. $\neg P(t_0)$ para algum $t_0 \in \mathcal{T}(\Sigma, \mathcal{X})$. Mas então a premissa de ibf implica que deve existir algum t_1 tal que $t_0 \stackrel{+}{\rightarrow} t_1$ e $\neg P(t_1)$. Pelo mesmo argumento, deve existir algum t_2 tal que $t_1 \stackrel{+}{\rightarrow} t_2$ e $\neg P(t_2)$. Portanto, existe uma cadeia infinita $t_0 \stackrel{+}{\rightarrow} t_1 \stackrel{+}{\rightarrow} t_2 ...$, i.e. $\rightarrow_R$ não é terminante. □

Teorema 4.2.2 [BN98] *Se $\rightarrow_R$ satisfaz ibf, então $\rightarrow_R$ é terminante.*

[3] De fato, a indução de $(\mathbb{N}, >)$ se generaliza para quaisquer sistemas de redução (ver [BN98], Seção 2.2), mas, em favor da objetividade, sistemas de redução abstratos não serão tratados nesta pesquisa.

[4] Os teoremas originais, enunciados em [BN98], foram modificados, aqui, visando a particularização para sistemas de reescrita de termos.

Prova: Defina-se $P(t)$:= "não existe nenhuma cadeia infinita começando de t". O passo de indução é simples: se não existe nenhuma cadeia infinita começando de nenhum sucessor de t, então não existe nenhuma cadeia infinita começando de t. Portanto, a premissa de ibf é satisfeita, e pode-se concluir que todo $t \in \mathcal{T}(\Sigma, \mathcal{X})$, i.e. $\rightarrow_R$ é terminante (pela aplicação do teorema anterior). □

4.2.2 Terminação

A propriedade *normalizante* de uma relação de reescrita, apresentada na Definição 4.1.6, não é suficiente para evitar que um termo, mesmo tendo uma forma normal, tenha uma derivação de reescrita infinita. Fazendo uso da noção de indução bem fundada, descrita na seção anterior, a propriedade *normalizante* pode ser "especializada" da seguinte forma:

Definição 4.2.2 Uma relação de reescrita é **fracamente normalizante** se cada termo tem uma forma normal (embora reduções infinitas possam existir).

Definição 4.2.3 Uma relação de reescrita é **fortemente normalizante** ou **Noetheriana**[5] ou **bem fundada** ou **terminante** se nenhum termo tem reduções infinitas, i.e. se ela satisfaz ibf.

Esta propriedade mais forte (infelizmente indecidível), de terminação, geralmente se faz necessária. O requisito de terminação proíbe regras de reescrita como $f(x, a) \rightarrow g(y)$ com uma variável no lado direito que não ocorre no esquerdo, pois $f(x, a) \rightarrow g(f(x, a)) \rightarrow g(g(f(x, a))) \rightarrow ...$ Além disso, uma regra como $x \rightarrow g(x)$ cujo lado esquerdo é uma variável que ocorre no lado direito, não pode pertencer a um sistema de reescrita terminante, caso contrário $x \rightarrow g(x) \rightarrow g(g(x)) \rightarrow ...$

Exemplo 4.2.1 [KK01] Um sistema de reescrita terminante: O sistema de reescrita restrito à regra

$$f(f(x)) \rightarrow f(g(f(x)))$$

é terminante, visto que a reescrita faz o número de f's adjacentes em qualquer termo decrescer.

Em geral, determinar a terminação é indecidível[6]. A idéia da prova é a seguinte: Dada qualquer máquina de Turing $\mathcal{M}$, existe um sistema de reescrita $R_{\mathcal{M}}$ tal que $R_{\mathcal{M}}$ termina para todos os termos sss $\mathcal{M}$ pára seja qual for a fita de entrada. Como a parada uniforme de uma máquina de Turing é indecidível, a terminação de sistemas de reescrita também o é.

[5] O termo "Noetheriana" é uma homenagem à matemática alemã Emmy Noether.

[6] No caso restrito de *sistemas de reescrita fechados*, i.e. sistemas de reescrita cujas regras não contêm variáveis, a terminação se torna decidível [BN98].

Exemplo 4.2.2 [KK01] Um sistema de reescrita não terminante: O sistema de reescrita dado pela regra

$$-(x+y) \to (--x+y)+y$$

não é terminante, visto que existe uma derivação infinita:

$$--(x+y) \to -((--x+y)+y) \to (--(--x+y)+y)+y \to \\ (-((----x+y)+y)+y)+y \to ...$$

Nota: [BN98] A terminação é uma importante propriedade de sistemas de reescrita de termos. Para um sistema de reescrita terminante finito, uma forma normal de um dado termo pode ser encontrada por uma simples busca de "dentro para fora" (*depth-first*). Caso o sistema seja também confluente, as formas normais são únicas, o que torna o problema da palavra decidível, para a correspondente teoria equacional.

4.2.3 Ordens de redução

Provar a terminação de um sistema de reescrita requer tratar de um conjunto infinito de possíveis contextos para cada instância de uma regra. Isto motiva a definição de *ordem de redução*, a qual incorpora as propriedades de fecho sob a substituição e sob a aplicação de contexto. Aqui serão consideradas apenas as ordens de redução construídas diretamente sobre termos, mas também é possível a sua construção através de interpretações (veja [BN98, KK01]).

Definição 4.2.4 Uma **ordem de redução** "$>$" é uma ordem bem-fundada fechada sob aplicação de contexto e substituição, a qual é tal que para qualquer contexto $C[\_]$ e qualquer substituição σ, se $s > t$ então $C[s] > C[t]$ e $\sigma(s) > \sigma(t)$.

Ou seja, uma ordem de redução é uma ordem de reescrita bem fundada.

Exemplo 4.2.3 [BN98] A ordem estrita "$>$" em $\mathcal{T}(\Sigma, \mathcal{X})$ definida por[7]

$$s > t \text{ sss } |s| > |t|$$

é bem fundada e fechada sob aplicações de contexto. Por exemplo,

$$|f(f(x,x),y)| = 5 > 3 = |f(y,y)|.$$

[7] Vale lembrar que $|t|$ denota o tamanho de t.

Entretanto, "$>$" não é uma ordem de redução, pois não é fechada sob substituições. Por exemplo, para a substituição $\sigma = \{y \mapsto f(x,x)\}$,

$$\begin{array}{rcl} |\sigma(f(f(x,x),y))| & = & |f(f(x,x),f(x,x))| \;=\; 7, \\ |\sigma(f(y,y))| & = & |f(f(x,x),f(x,x))| \;=\; 7. \end{array}$$

Note-se que "$>$" não é fechada sob substituições simplesmente porque uma variável qualquer pode ter mais ocorrências no termo menor do que no maior. Se isto for proibido, obtém-se naturalmente uma ordem de redução:[8]

$$s > t \text{ sss } |s| > |t| \text{ e } \forall x \in X, |s|_x \geq |t|_x.$$

A seguir, dois importantes lemas para ordens de redução serão enunciados:

Lema 4.2.1 [KK01] *Em uma ordem de redução "$>$" um termo nunca é menor do que um de seus sub-termos, i.e. nunca acontece que $t_{|p} > t$ para $p \in \mathcal{D}om(t), p \neq \Lambda$.*

Prova: Assumindo que $t_{|p} > t$, então $t = t[t_{|p}]_p > t[t]_p$ e, portanto, pode-se construir uma sequência de termos infinitamente decrescente:

$$t_{|p} > t > t[t]_p > t[t[t]_p]_p > t[t[t[t]_p]_p]_p > ...$$

contradizendo o fato de a ordem de redução ser bem-fundada. □

Lema 4.2.2 [KK01] *Quando uma ordem de redução é total então ela contém a ordem de sub-termo.*

Prova: Como um termo e qualquer de seus sub-termos são comparáveis, e considerando o lema anterior, então sempre $t > t_{|p}$. Caso contrário, se $t_{|p} > t$ para algum termo t e alguma posição p, então existe uma sequência decrescente infinita

$$t > t[t]_p > t[t[t]_p]_p > ...$$

contradizendo o fato de a ordem de redução ser bem fundada. □

Usando uma ordem de redução, a terminação de reescrita pode ser provada por comparação dos lados esquerdo e direito das regras, graças ao seguinte teorema:

[8] $|t|_x$ denota a quantidade de ocorrências de x em t.

Teorema 4.2.3 [KK01] *Um sistema de reescrita R sobre o conjunto de termos $\mathcal{T}(\Sigma, \mathcal{X})$ é terminante sss existe uma ordem de redução "$>$" tal que toda regra $l \to r \in R$ satisfaz $l > r$.*

Prova: Se R é terminante sobre $\mathcal{T}(\Sigma, \mathcal{X})$, defina-se "$s > t$" se $s \xrightarrow{+}_R t$. Então "$>$" é claramente uma ordem de redução. Por outro lado, se existe uma ordem de redução "$>$" tal que toda regra $l \to r \in R$ satisfaz "$l > r$", qualquer cadeia descendente $t_1 \to_R t_2 \to_R ...$ corresponde a uma cadeia descendente $t_1 > t_2 > ...$ e assim, R deve ser terminante, caso contrário $>$ não seria bem-fundada. □

Assim, provar a terminação de um sistema de reescrita de termos finito $R \subseteq \mathcal{T}(\Sigma, \mathcal{X}) \times \mathcal{T}(\Sigma, \mathcal{X})$ recai em empregar-se uma ordem Noetheriana[9] "$>$", segundo a ibf definida na seção anterior, tal que R é terminante se, para quaisquer termos $s, t \in \mathcal{T}(\Sigma, \mathcal{X})$:

$$s \to_R t \Rightarrow s > t$$

Ao invés de decidir $s > t$ para os (infinitamente diversos) pares s, t com $s \to_R t$, é preferível checar $l > r$ para as (finitamente diversas) regras $l \to r \in R$.

4.2.4 Ordens de simplificação

Uma outra abordagem para provar terminação é baseada no seguinte fato: dada uma ordem de reescrita bem fundada (i.e. uma ordem de redução) "$>$", se ela é total, então ela contém a relação de sub-termo próprio "$\trianglerighteq_{sub}$", conforme foi demonstrado no Lema 4.2.2. Por outro lado, pode-se provar que para um $\mathcal{F}$ finito, se a ordem de reescrita contém a relação de sub-termo $\trianglerighteq_{sub}$, então ela é bem-fundada. Por simplicidade, o conceito de *ordem de simplificação* será definido sobre os termos homogêneos $\mathcal{T}(\mathcal{F}, \mathcal{X})$:

Definição 4.2.5 Uma ordem estrita "$>$" sobre $\mathcal{T}(\mathcal{F}, \mathcal{X})$ é chamada **ordem de simplificação** sss ela é uma ordem de reescrita e satisfaz a seguinte **propriedade de sub-termo**:

$t > t_{|p}$, para todo termo $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ e toda posição $p \in \mathcal{P}os(t) - \{\Lambda\}$.

Uma vez que "$>$" é uma ordem de reescrita (e, portanto, fechada sob a aplicação de contexto), a propriedade de sub-termo já segue da seguinte propriedade (mais simples):

$f(x_1, ..., x_i, ..., x_n) > x_i$, para todo $n \geq 1$, todo símbolo $f \in \mathcal{F}_n$, toda variável $x_1, ..., x_n \in \mathcal{X}$, e todo $1 \leq i \leq n$.

[9] i.e. bem fundada ou terminante.

Nota: [BN98] A definição de ordem de simplificação substitui o requisito de "$>$" ser bem fundada, na definição de ordens de redução, pela propriedade de subtermo. A fim de mostrar que este é um requisito mais forte, i.e. que toda ordem de simplificação é uma ordem de redução, faz-se necessária a definição da *imersão homeomórfica* (*homeomorphic embedding*) de termos e, ainda, a enunciação do Teorema de Kruskal para a relação de imersão.

Definição 4.2.6 A **imersão homeomórfica** "$\trianglerighteq_{emb}$" é uma relação binária sobre $\mathcal{T}(\mathcal{F}, \mathcal{X})$, definida como: Seja $x \in \mathcal{X}$ e $f \in \mathcal{F}_n$, então $s \trianglerighteq_{emb} t$ sss

1. $s = x = t$, ou
2. $s = f(s_1, ..., s_n)$, $t = f(t_1, ..., t_n)$ e $s_1 \trianglerighteq_{emb} t_1, ..., s_n \trianglerighteq_{emb} t_n$, ou
3. $s = f(s_1, ..., s_n)$ e $s_j \trianglerighteq_{emb} t$, para algum $1 \leq j \leq n$.

Exemplo 4.2.4 [BN98]

$$f(f(h(a), h(x)), f(h(x), a)) \trianglerighteq_{emb} f(f(a, x), x).$$

Olhando para a representação em árvore dos termos acima, a imersão homeomórfica é dado pelo mapeamento da Figura 4.2.

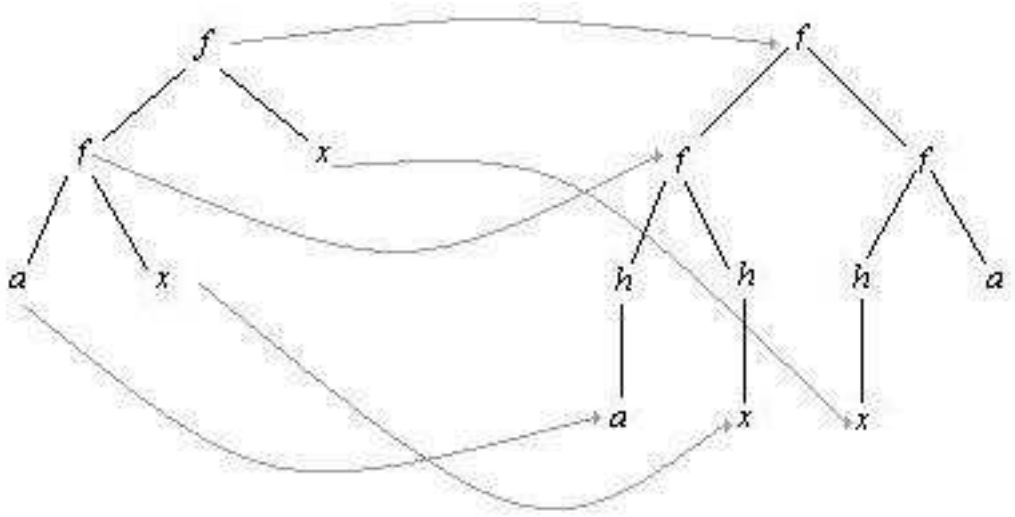


Figura 4.2: Imersão $f(f(a, x), x) \trianglelefteq_{emb} f(f(h(a), h(x)), f(h(x), a))$ [BN98]

A definição de *boa-ordem-parcial* será usada para fornecer uma definição alternativa de imersão homeomórfica:

Definição 4.2.7 Uma ordem parcial "$\geq$" sobre um dado conjunto $\mathcal{T}$ é uma **boa-ordem-parcial** sss se toda sequência infinita $t_1, t_2, ...$ de elementos de $\mathcal{T}$ contém dois elementos t_i e t_j tal que $i < j$ e $t_i \leq t_j$.

O sistema de reescrita[10]

$$R_{emb} = \{f(x_1, ..., x_n) \rightarrow x_i | n \geq 1, f \in \mathcal{F}_n, 1 \leq i \leq n\}.$$

induz uma imersão homeomórfica "$\trianglerighteq_{emb}$", dado pelo fecho transitivo reflexivo da relação de reescrita "$\rightarrow_{R_{emb}}$", i.e. $\trianglerighteq_{emb} = \overset{*}{\rightarrow}_{R_{emb}}$. Deve-se notar, ainda, que R_{emb} é claramente terminante e, portanto, "$\trianglerighteq_{emb}$" é uma ordem parcial bem fundada.

O importante Teorema de Kruskal afirma que, para $\mathcal{F}$ e $\mathcal{X}$ finitos, "$\trianglerighteq_{emb}$" é uma boa-ordem-parcial:

Teorema 4.2.4 (Kruskal) [BN98] *Seja $\mathcal{F}$ uma assinatura finita e $\mathcal{X}$ um conjunto finito de variáveis. Então a imersão homeomórfica "$\trianglerighteq_{emb}$" sobre $\mathcal{T}(\mathcal{F}, \mathcal{X})$ é uma boa-ordem-parcial.*

Prova: Veja [BN98], pg. 114. □

No que segue, o Teorema de Kruskal será usado para provar que toda ordem de simplificação é bem fundada. Para isto, o primeiro passo é mostrar que toda ordem de simplificação contém a relação de imersão homeomórfica:

Lema 4.2.3 [BN98, KK01] *Seja "$\geq$" uma ordem de simplificação sobre $\mathcal{T}(\mathcal{F}, \mathcal{X})$ e $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. Então $s \trianglerighteq_{emb} t \Rightarrow s \geq t$. Em outras palavras, a ordem de simplificação "$\geq$" contém a imersão homeomórfica "$\trianglerighteq_{emb}$".*

Prova: A imersão homeomórfica é a menor relação termo-fechada transitiva reflexiva satisfazendo a condição da Definição 4.2.6, e claramente contém a relação de sub-termo, sendo assim menor do que qualquer ordem de simplificação. Uma prova mais formal pode ser encontrada em [BN98], baseada na indução sobre $|s|$. □

Uma vez que toda ordem de reescrita que contenha a relação de sub-termo $\trianglerighteq_{sub}$ também contém a imersão homeomórfica, a propriedade de sub-termo é suficiente para garantir que uma ordem de simplificação seja bem-fundada, para $\mathcal{F}$ finito. Esta é a idéia que fundamenta a noção de ordem de simplificação. De fato, toda ordem de simplificação é uma ordem de redução. Formalmente:

Teorema 4.2.5 [BN98] *Seja $\mathcal{F}$ uma assinatura finita. Toda ordem de simplificação "$>$" sobre $\mathcal{T}(\mathcal{F}, \mathcal{X})$ é uma ordem de redução.*

[10] A regra $f(x_1, ..., x_n) \rightarrow x_i$ é chamada *remoção de contexto* e poderia ser generalizada para a regra $f(t_1, ..., t_n) \rightarrow t_i$, chamada *remoção de sub-termo*.

Prova: Pela Definição 4.2.5, resta mostrar que toda ordem de simplificação é bem fundada. Assim, assumindo que "$>$" é uma ordem de simplificação sobre $\mathcal{T}(\mathcal{F},\mathcal{X})$, e que $t_1 > t_2 > ...$ é uma cadeia infinita em $\mathcal{T}(\mathcal{F},\mathcal{X})$:

1. Verifica-se, por contradição, que "$>$" satisfaz $\mathcal{V}ar(t_1) \supset \mathcal{V}ar(t_2) \supset ...$ Assumindo que $x \in \mathcal{V}ar(t_{i+1}) - \mathcal{V}ar(t_i)$. Dada uma substituição $\sigma = \{x \mapsto t_i\}$, então, por um lado, $t_i = \sigma(t_i)$ (visto que x não ocorre em t_i) e $\sigma(t_i) > \sigma(t_{i+1})$ (visto que "$>$" é uma ordem de reescrita). Por outro lado, t_i é um sub-termo de $\sigma(t_{i+1})$, e assim, $\sigma(t_{i+1}) \geq t_i$, pela propriedade de sub-termo. Combinando as duas desigualdades, obtém-se $t_i > t_i$, o que é uma contradição.

2. A primeira parte da prova mostra que, para um conjunto finito $\mathcal{X} = \mathcal{V}ar(t_1)$, todo termo na sequência $t_1, t_2, ...$ pertence a $\mathcal{T}(\mathcal{F},\mathcal{X})$. Uma vez que $\mathcal{F}$ e $\mathcal{X}$ são finitos, o Teorema de Kruskal implica que esta sequência seja "boa", i.e. existe $i < j$ tal que $t_i \trianglerighteq_{emb} t_j$. Agora, pelo Lema 4.2.3, obtém-se $t_i \leq t_j$, o que contradiz o fato de que $t_i > t_{i+1} > ... > t_j$. □

Nota: A recíproca do teorema acima não é verdadeira, ou seja, existem ordens de redução que não são ordens de simplificação. Entretanto, é possível mostrar (veja [BN98], pg. 116) que a recíproca deste teorema é verdadeira para ordens de redução totais, sobre termos fechados.

A classe de sistemas de reescrita cuja terminação pode ser provada usando uma ordem de simplificação é importante o suficiente para justificar uma terminologia específica e estudar suas propriedades.

Definição 4.2.8 Um sistema de reescrita R é **terminante por simplificação** se existe uma ordem de simplificação "$\geq$" tal que para toda regra $l \to r$ em R, $l > r$.

Teorema 4.2.6 [KK01] *Seja $\mathcal{F}$ uma assinatura finita. Um sistema de reescrita R é terminante se R é terminante por simplificação.*

Prova: Se existe uma sequência infinita $t_1 \to_R t_2... \to_R t_i...$, então $t_1 > t_2... > t_i...$ Uma vez que a imersão homeomórfica é uma boa-ordem-parcial sobre $\mathcal{T}(\mathcal{F},\mathcal{X})$, existe um par de termos t_i e t_j tal que $i < j$ e $t_i \trianglerighteq_{emb} t_j$, o que implica que $t_i \leq t_j$, pelo Lema 4.2.3. Mas isto contradiz o fato de que $t_i > t_j$. □

Exemplo 4.2.5 [BN98] Seja $\mathcal{F} = \{f, g\}$, onde $aridade(f) = aridade(g) = 1$. Considere-se o seguinte sistema de reescrita de termos:

$$R = \{f(f(x)) \to f(g(f(x)))\}.$$

Pode-se provar que R é terminante. Neste caso, $\overset{+}{\rightarrow}_R$ é uma ordem de redução. $\overset{+}{\rightarrow}_R$ não pode ser uma ordem de simplificação, já que então $f(g(f(x))) \trianglerighteq_{emb} f(f(x))$ implicaria que $f(g(f(x))) \overset{+}{\rightarrow}_R f(f(x))$, o que, junto com a regra $f(f(x)) \rightarrow f(g(f(x)))$, contradiz a terminação de R.

Em particular, R é um exemplo de sistema de reescrita de termos terminante cuja terminação não pode ser provada com a ajuda de ordens de simplificação, já que $f(g(f(x))) > f(f(x))$ para qualquer ordem de simplificação "$>$".

Nesta seção, o conceito de ordem de simplificação foi abordado de forma genérica. Existem vários métodos para a definição de ordens de simplificação específicas, como, por exemplo:

- **Ordens de simplificação polinomiais**, induzidas por mapeamentos de símbolos de função da assinatura em polinômios;
- **Ordens de Knuth-Bendix**, determinadas por uma ordem estrita sobre a assinatura e uma função peso; e
- **Ordens de caminho recursivas**, em que dois termos são comparados, primeiramente, em função dos seus símbolos da raiz.

Os métodos acima têm em comum o fato de construírem uma ordem de simplificação a partir de alguma ordem bem fundada sobre os símbolos funcionais $f \in \mathcal{F}$, chamada **precedência**.

4.3 Confluência de sistemas de reescrita

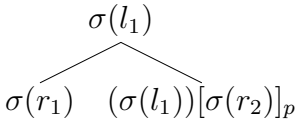
Nesta seção, outra importante propriedade de sistemas de reescrita será discutida: a propriedade de *confluência*, já apresentada sucintamente na Definição 4.1.6. No Capítulo 2 (ver seções 2.1 e 2.5), os sistemas de reescrita foram apresentados como uma ferramenta para definição da semântica operacional de linguagens formais. Neste contexto, olhando para um sistema de reescrita de termos como um programa, a confluência simplesmente significa que o programa é determinístico. A propriedade de confluência será analisada, primeiramente, para sistemas de reescrita terminantes, e, na Seção 4.3.4, será contemplado também o caso de sistemas de reescrita não terminantes.

4.3.1 Pares críticos

Quando duas regras de reescrita podem ser aplicadas a um mesmo termo, tem-se um *par crítico*, e isto representa um problema para a prova da confluência. Um par

crítico é o resultado de unificar-se o lado esquerdo de uma regra com um sub-termo (que não seja uma variável) do lado esquerdo de outra (potencialmente a mesma) regra, e reduzir-se o termo resultante usando ambas as regras. Formalmente:

Definição 4.3.1 Sejam $l_1 \to r_1$ e $l_2 \to r_2$ regras cujas variáveis foram renomeadas de tal modo que $\mathcal{V}ar(l_1, r_1) \cap \mathcal{V}ar(l_2, r_2) = \emptyset$. Seja a posição $p \in \mathcal{P}os(l_1)$ tal que $l_{1|p}$ não seja uma variável, e seja σ um unificador mais geral de $l_{1|p} =^? l_2$. Isto determina um **par crítico** $\langle \sigma(r_1), (\sigma(l_1))[\sigma(r_2)]_p \rangle$:

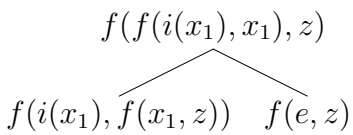


Quando duas regras produzem um par crítico, então diz-se que ocorreu uma **sobreposição** (*overlap*). Os pares críticos de um sistema de reescrita R são os pares críticos entre duas de suas regras (renomeadas) quaisquer, e são denotados por $CP(R)$. Note-se que, por definição, $u_1 =_R u_2$ para todo par crítico $\langle u_1, u_2 \rangle \in CP(R)$: pares críticos de R são conseqüências equacionais de R.

Exemplo 4.3.1 [BN98] Sejam as regras

$$\begin{array}{lrcl} (1) & f(f(x,y),z) & \to & f(x,f(y,z)), \\ (2) & f(i(x_1),x_1) & \to & e, \end{array}$$

as quais produzem um par crítico pela unificação do sub-termo $f(x, y)$ do lado esquerdo da regra (1) e o lado esquerdo $f(i(x_1), x_1)$ da regra (2). Um unificador mais geral $\{x \mapsto i(x_1), y \mapsto x_1\}$ e as setas correspondentes são



4.3.2 Confluência

O uso da noção de pares críticos na prova da confluência será visto a seguir. Mas antes, serão discutidas as propriedades Church-Rosser e confluência.

A propriedade Church-Rosser estabelece a relação entre a substituição de iguais por iguais e a reescrita:

Uma relação $\to_R$ sobre $\mathcal{T}(\Sigma, \mathcal{X})$ é **Church-Rosser** sss

$$\forall t, t' \ . \ t \overset{*}{\leftrightarrow}_R t' \text{ sss } \exists s \ . \ t \overset{*}{\rightarrow}_R s \overset{*}{\leftarrow}_R t',$$

conforme a Definição 4.1.6.

A propriedade de confluência, por sua vez, é o conceito chave para garantir o determinismo da relação de reescrita, vista como um processo computacional. Ela diz que duas computações, começando de um mesmo termo, darão efetivamente o mesmo resultado:

Uma relação $\rightarrow_R$ sobre $\mathcal{T}(\Sigma, \mathcal{X})$ é **confluente** sss

$$\forall t, t', t'' \ . \ t' \overset{*}{\leftarrow}_R t \overset{*}{\rightarrow}_R t'' \Rightarrow \exists s \ . \ t' \overset{*}{\rightarrow}_R s \overset{*}{\leftarrow}_R t'',$$

conforme a Definição 4.1.6.

Uma noção mais fraca de confluência é a propriedade de *confluência local*:

Definição 4.3.2 Uma relação $\rightarrow_R$ sobre $\mathcal{T}(\Sigma, \mathcal{X})$ é chamada **localmente confluente** sss

$$\forall t, t', t'' \ . \ t' \leftarrow_R t \rightarrow_R t'' \Rightarrow \exists s \ . \ t' \overset{*}{\rightarrow}_R s \overset{*}{\leftarrow}_R t'';$$

Analogamente, existe a propriedade de *confluência forte*:

Definição 4.3.3 Uma relação $\rightarrow_R$ sobre $\mathcal{T}(\Sigma, \mathcal{X})$ é chamada **fortemente confluente** sss

$$\forall t, t', t'' \ . \ t' \leftarrow_R t \rightarrow_R t'' \Rightarrow \exists s \ . \ t' \overset{=}{\rightarrow}_R s \overset{*}{\leftarrow}_R t'';$$

A Figura 4.3 ilustra a relação entre as propriedades acima definidas.

Nota: A necessidade de se definirem noções mais fracas de confluência surge da complexidade de se lidar com "flechas" de tamanho arbitrário em $t' \overset{*}{\rightarrow}_R t \overset{*}{\leftarrow}_R t''$. É possível demonstrar que, tratando-se de relações de reescrita terminantes, a *confluência local* implica a *confluência*; por outro lado, toda relação *fortemente confluente* é uma relação *confluente*.

Teorema 4.3.1 [BN98] *Um sistema de reescrita de termos é localmente confluente sss*

$$\forall \langle u_1, u_2 \rangle \in CP(R) \ . \ \exists t \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \ . \ u_1 \overset{*}{\rightarrow} t \overset{*}{\leftarrow} u_2.$$

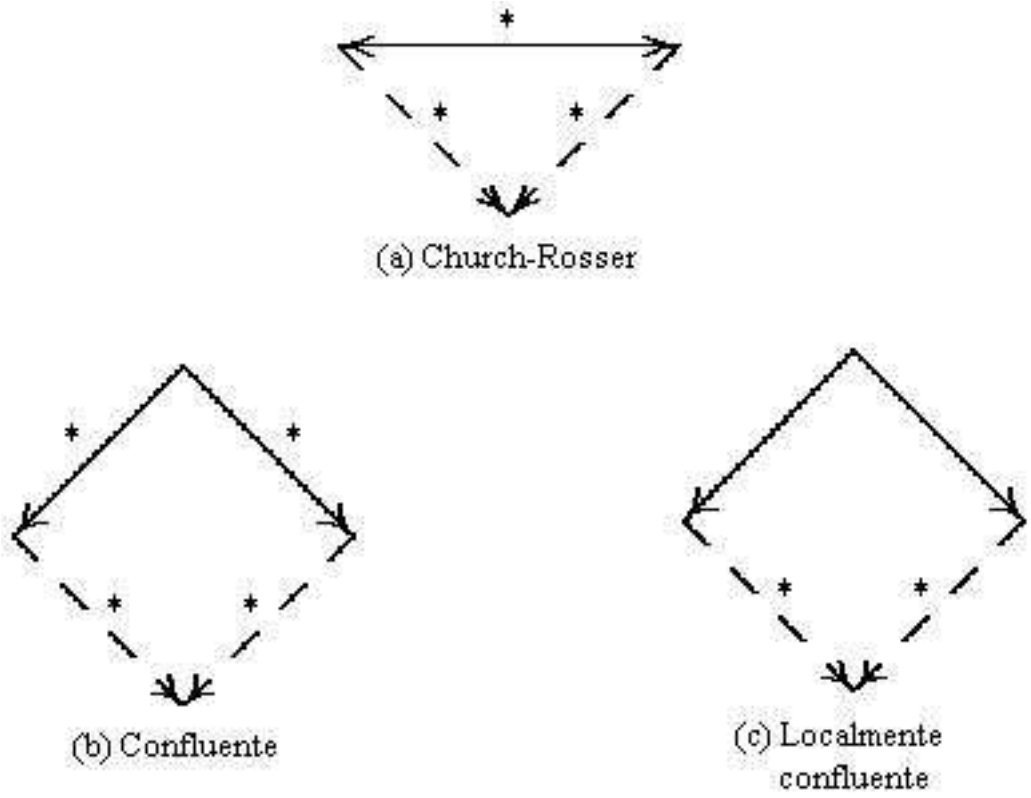


Figura 4.3: Propriedades de confluência

Prova: Veja [BN98], pg. 28. □

Exemplo 4.3.2 [BN98] Seja o sistema de reescrita de termos $R = \{f(f(x)) \rightarrow g(x)\}$, o qual possui exatamente um par crítico como resultado da sobreposição da regra com uma variante renomeada $f(f(y)) \rightarrow g(y)$. O lado esquerdo $f(f(x))$ unifica com o sub-termo $f(y)$ dos lados esquerdos renomeados, produzindo o unificador mais geral $\{y \mapsto f(x)\}$. Então, obtém-se o par crítico:

$$\begin{array}{ccc} & f(f(f(x))) & \\ \swarrow & & \searrow \\ g(f(x)) & & f(g(x)) \end{array}$$

Claramente R não é confluente porque o par crítico já está na forma normal. Este exemplo demonstra a necessidade de duas condições na definição do par crítico de um sistema de reescrita de termos.

- Regras precisam ser renomeadas. No exemplo, $f(f(x))$ e $f(x)$ não são unificáveis e não poderiam produzir par crítico.
- Os pares críticos entre uma regra e (uma cópia renomeada) dela mesma devem ser considerados. Caso contrário, todo sistema de uma regra aparentaria ser localmente confluente, o que claramente não é o caso.

Uma vez que, para sistemas de reescrita terminantes, a confluência local implica a confluência, o seguinte corolário do Teorema 4.3.1 é imediato:

Corolário 4.3.1 [BN98] *Um sistema de reescrita de termos terminante é confluente sss*

$$\forall \langle u_1, u_2 \rangle \in CP(R) \ . \ \exists t \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \ . \ u_1 \xrightarrow{*} t \xleftarrow{*} u_2.$$

E, ainda, como um sistema de reescrita finito possui somente pares críticos finitamente diversos, então:

Corolário 4.3.2 [BN98] *A confluência de um sistema de reescrita de termos finito e terminante é decidível.*

Prova: Veja [BN98], pg. 140. □

A decidibilidade da confluência é um resultado fundamental na busca de teorias equacionais decidíveis: se E constitui um sistema de reescrita terminante R, pode-se decidir se R é também confluente. Neste caso (R é convergente), é possível demonstrar que $=_E$ é decidível [BN98]. Complementando, um sistema de reescrita R fornece um *procedimento de decisão* para uma teoria equacional E se R é finito, convergente e $=_R$ coincide com $=_E$ [KK01].

Como foi visto na Seção 3.2, o problema da palavra consiste no problema de decidir se uma igualdade $s = t$ entre dois termos fechados segue de E ou não. Isto é um caso particular de provabilidade em E para termos arbitrários. Se R é convergente fechado, o problema da palavra é decidível reduzindo-se ambos os termos s e t a suas formas normais e testando-se a igualdade sintática dos resultados [KK01].

Evidentemente, nem toda teoria equacional pode ser decidida por reescrita. Algumas teorias não são finitamente apresentadas. Além disso algumas teorias finitamente apresentadas são indecidíveis (uma lógica combinatória, por exemplo). Mesmo certas teorias finitamente apresentadas e decidíveis não são decidíveis por reescrita. [KK01]

Exemplo 4.3.3 O sistema finito Thue, encontrado por Kapur e Narendran [KN85], composto pelo único axioma:

$$a(b(a(x))) = b(a(b(x))),$$

para o qual o problema da palavra é decidível, mas não existe um sistema convergente finito equivalente.

4.3.3 Sistemas reduzidos

Reduzindo os lados direitos das regras e eliminando as regras cujos lados esquerdos são redutíveis pela relação de reescrita, um sistema de reescrita convergente pode sempre ser convertido em um sistema reduzido.

Definição 4.3.4 Um sistema de reescrita R é **reduzido** ou **inter-reduzido** se, para cada regra $l \to r \in R$, o lado direito r é irredutível por R e nenhum termo s, estritamente menor do que l na ordem de encompassamento $\sqsubseteq$, é redutível.

Uma das propriedades interessantes dos sistemas de reescrita reduzidos é que, para uma dada teoria equacional, só existe um sistema (denominado *canônico*) contido em uma ordem de redução particular [DJ90, KK01], qualquer que seja esta teoria.

Definição 4.3.5 Seja R um conjunto de regras de reescrita e "$>$" qualquer ordem de redução. "$>$" **contém** R se para qualquer regra de reescrita $l \to r$ em R, $l > r$.

4.3.4 Sistemas ortogonais

Embora a confluência de sistemas de reescrita não-terminantes seja indecidível, é possível verificar tal propriedade para uma grande classe de sistemas não-terminantes, adicionando-se condições sintáticas fortes nos lados esquerdos.

Definição 4.3.6 Uma regra de reescrita $l \mapsto r$ é chamada **linear à esquerda** se nenhuma variável ocorre duas vezes em l. Analogamente, $l \mapsto r$ é chamada **linear à direita** se nenhuma variável ocorre duas vezes em r. Uma regra $l \mapsto r$ é chamada **linear** se ela é, ao mesmo tempo, linear à esquerda e linear à direita. Um sistema de reescrita de termos é chamado linear à esquerda se todas as suas regras são lineares à esquerda, linear à direita se todas as suas regras são lineares à direita, e linear se todas as suas regras são lineares.

Definição 4.3.7 Um sistema de reescrita de termos que é linear à esquerda e não possui pares críticos é chamado **ortogonal**.

Por exemplo, o sistema da Lógica Combinatória dado anteriormente é ortogonal.

Teorema 4.3.2 [KK01] *Se um sistema de reescrita R é ortogonal, então ele é confluente.*

Prova: Esta prova é dada por [Hue80], usando o fato de que a redução paralela (i.e., aplicação paralela de regras de R em redex's disjuntos) é fortemente confluente. □

Nota: A fim de se obter o resultado acima, a hipótese de ortogonalidade não pode ser enfraquecida, a não-sobreposição (i.e. a não existência de pares críticos) é claramente necessária, bem como a linearidade à esquerda.

O seguinte sistema de reescrita (obviamente, não-terminante) mostra que a linearidade à esquerda é essencial:

Exemplo 4.3.4 [BN98] As regras $f(x,x) \to a$, $f(x,g(x)) \to b$ e $c \to g(c)$ não se sobrepõem ($f(x,x)$ e $f(x',g(x'))$ *não* unificam!), ainda que o termo $f(c,c)$ tenha duas formas normais a e b.

A confluência de sistemas ortogonais estabelece a corretude da semântica operacional de linguagens de programação recursivas, garantindo, desta forma, que a computação de dois programas diferentes (mas equivalentes do ponto de vista lógico) produza o mesmo resultado.

Diversos teoremas interessantes podem ser provados por sistemas de reescrita ortogonais. Deve-se lembrar, primeiro, que um sistema de reescrita pode ser *fracamente normalizante* ou *fortemente normalizante*, conforme a Definição 4.2.2 e a Definição 4.2.3, respectivamente. Além disso,

Definição 4.3.8 Um sistema de reescrita é **fracamente normalizante interno** (*weakly innermost normalizing*) se todo termo tem uma forma normal que pode ser alcançada por **redução interna** (*innermost reduction*), i.e. uma redução na qual são selecionados somente redex's que não contém propriamente outros redex's.

As seguintes propriedades são equivalentes se o sistema de reescrita é ortogonal:

Teorema 4.3.3 [O'D77] *Seja R um sistema de reescrita ortogonal, R é fortemente normalizante sss R é fracamente normalizante interno.*

Teorema 4.3.4 [Ken89] *Todo sistema de reescrita ortogonal tem uma estratégia de redução computável, sequencial, normalizante.*

Nota: [KK01] Em geral, entretanto, o problema de encontrar uma estratégia ótima que evite todas as reescritas desnecessárias é indecidível, ressaltando, ainda, que a maioria das propriedades de sistemas de reescrita é indecidível.

4.4 Completude

Este capítulo introdutório de reescrita será fechado com uma discussão sucinta da completude de sistemas de reescrita de termos, no contexto da lógica equacional.

Até aqui, os principais conceitos relacionados a sistemas de reescrita foram apresentados, e suas propriedades fundamentais (i.e. terminação e confluência) foram discutidas. Entretanto, uma importante questão esteve sempre implícita nas seções anteriores deste capítulo: Dado um conjunto finito de axiomas equacionais E, como construir um procedimento de decisão para o problema da palavra para E?

Como o problema da palavra é geralmente indecidível, qualquer que seja o método para a construção de um procedimento de decisão, ele será necessariamente incompleto, ou seja, ele não vai ter sucesso para todo conjunto finito E. O teorema a seguir afirma que o problema da palavra para E é decidível se $\rightarrow_E$ é convergente:

Teorema 4.4.1 [BN98] *Se E é finito e $\rightarrow_E$ é convergente, então $=_E$ é decidível.*

Prova: Veja [BN98], pg. 59. □

Conforme [BN98], este teorema sugere um primeiro "algoritmo" (o qual não tem muita probabilidade de sucesso), formado pelos seguintes passos:

- **Verifique terminação:** Tente encontrar uma ordem de redução "$>$" tal que $s > t$ seja satisfeito por todos os axiomas equacionais $(s = t) \in E$. No caso de sucesso, considere o sistema de reescrita de termos $R = \{s \rightarrow t | (s = t) \in E\}$, e continue com este sistema no próximo passo; caso contrário, acuse erro.
- **Verifique confluência:** Decida a confluência do sistema de reescrita de termos R, cuja terminação é conhecida, computando todos os pares críticos entre as regras de R e testando se para todo par crítico (t, t') de R existe um s tal que $t \stackrel{*}{\rightarrow} s \stackrel{*}{\leftarrow} t'$. No caso de sucesso, a relação de reescrita $\rightarrow_R$ gera um procedimento de decisão para problema da palavra para E; caso contrário, acuse erro.

Exemplo 4.4.1 [BN98] Seja $E = \{x + 0 = x, x + s(y) = s(x + y)\}$. No primeiro passo, a *terminação* de R pode ser mostrada usando a ordem de caminho lexicográfica[11] "$>_{lex}$" que é induzida pela ordem de precedência "$>$" satisfazendo

[11] Um caso particular das ordens de caminho recursivas.

$+ > s$. A *confluência* de R segue trivialmente porque R não possui pares críticos.

Conseqüentemente, R é canônico e, desta forma, pode ser usado para decidir o problema da palavra para E. Agora, por um lado, pode-se deduzir, por exemplo, que $s(s(0)) + s(0) =_E s(0) + s(s(0))$ é satisfeito porque estes dois termos se reduzem para a mesma forma normal $s(s(s(0)))$ com relação a R. Por outro lado, pode-se usar R para mostrar que um certo axioma equacional não é uma conseqüência de E: por exemplo, $x + y =_E y + x$ não é satisfeito porque os dois termos são R-irredutíveis e distintos.

Uma razão para este método simples falhar é a existência de algum axioma $s = t$ que precise ser transformado na regra $t \to s$ ao invés de $s \to t$. Por exemplo, $x = x + 0$ acarretaria uma regra de reescrita não terminante $x \to x + 0$, embora $x + 0 \to x$ seja terminante. Assim, um primeiro passo, mais sensível, seria tentar ordenar os axiomas equacionais de entrada de um modo apropriado:

- **Verifique terminação:** Seja $E = \{s_1 = t_1, ..., s_n = t_n\}$. Tente encontrar uma ordem de redução "$>$" tal que, para todo $1 \leq i \leq n$, existem termos l_i, r_i satisfazendo $\{l_i, r_i\} = \{s_i, t_i\}$ e $l_i > r_i$. No caso de sucesso, considere o sistema de reescrita de termo $R = \{l_1 \to r_1, ...l_n \to r_n\}$ no próximo passo; caso contrário, acuse falha.

Nota: [BN98] Este primeiro passo, mesmo modificado, pode ocasionar falhas freqüentes. Isto pode acontecer devido a um dos axiomas ser inerentemente não terminante, tal como a identidade $x + y = y + x$, ou por não se encontrar uma ordem de redução apropriada mesmo que tal ordem exista. Além disso, ainda que o primeiro passo tenha sucesso, o método poderia falhar devido ao sistema de reescrita terminante resultante não ser confluente.

Neste capítulo, foram fornecidos os principais conceitos relacionados a sistemas de reescrita de termos, assim como uma análise das suas propriedades de terminação e confluência. Por fim, o procedimento básico de completude foi brevemente apresentado[12]. Desta maneira, está descrita a natureza do modelo formal proposto nesta dissertação. Agora, no capítulo seguinte, será estudado o objeto matemático que se pretende modelar.

[12]Para uma análise detalhada deste procedimento, veja [BN98], cap. 7.

Capítulo 5

Representação Computacional dos Números Reais

Representar computacionalmente os números reais, bem como realizar operações aritméticas envolvendo tais números, constitui um problema central para esta pesquisa. Predominantemente, o modelo de ponto-flutuante tem sido adotado como solução para este problema, e será enfocado na Seção 5.1. A Matemática Intervalar fornece uma representação alternativa, e será descrita na Seção 5.2. Por fim, na Seção 5.3, será mostrado como estas duas abordagens podem ser combinadas, em direção a um modelo para a computação dos números reais.

5.1 Padrão IEEE 754

A representação computacional de números reais em máquina é feita, em geral, por elementos de um subconjunto finito F dos racionais $\mathbb{Q}$, chamados pontos-flutuantes, os quais são usados como aproximações dos números reais durante as computações. A definição abstrata de tais sistemas costuma ser dada como uma estrutura matemática da forma $\mathbb{F} = \langle F, b, p, e\_min, e\_max, O \rangle$, onde $F \subset \mathbb{Q}$ é o conjunto de todos os números da máquina, b é a base numérica (binária, decimal etc.), p é a precisão, e_min o mínimo expoente e e_max o máximo expoente do sistema, e, finalmente, $O : \mathbb{R} \to F$, uma função de arredondamento.

O Padrão IEEE 754 [IEE85], para aritmética de pontos-flutuantes binária, é o mais amplamente utilizado para representação dos números reais em computadores modernos, incluindo computadores pessoais baseados na plataforma Intel, Macintoshes, e a maioria das plataformas Unix. A seguir, serão descritos, de forma resumida, os requisitos estabelecidos pelo Padrão IEEE 754, requisitos estes refletidos nas regras do sistema de reescrita de termos proposto na Seção 6.2. Uma análise mais geral do modelo de ponto-flutuante, e das dificuldades inerentes a esta abordagem, pode ser encontrada em [Wal90, Gol91].

5.1.1 Definições básicas

Segue a descrição de algumas terminologias próprias do Padrão IEEE 754:

Definição 5.1.1 Um **número de ponto-flutuante binário** ou **ponto-flutuante Padrão IEEE 754** ou simplesmente **ponto-flutuante** é uma cadeia de bits formada por três componentes: um **sinal**, um **expoente** e um **significante**. O significante, por sua vez, é composto pelo **bit implícito**, o **ponto implícito** e a **fração**. O bit implícito designa a parte inteira[1] do significante e, assim como o ponto, não é armazenado, já que, no caso da base binária, seu valor é sempre 1.

O valor numérico de um ponto-flutuante de sinal "s", expoente "e" e fração "f", é dado por:

$$(-1)^s \times 1.f \times 2^e$$

Note-se que apenas o sinal do significante é informado. Mesmo assim, é possível representar expoentes negativos, na forma de *expoentes polarizados*:

Definição 5.1.2 Define-se como **expoente polarizado** (*biased exponent*) a soma do expoente e uma constante, denominada **bias**.

Neste caso, o valor numérico de um ponto-flutuante de sinal "s", expoente "e" e fração "f", passa a ser:

$$(-1)^s \times 1.f \times 2^{e-bias}$$

Por exemplo, fazendo bias $= 127$, o expoente 53 é representado pelo expoente polarizado $180 = 127 + 53$; o expoente -27, por sua vez, é representado pelo expoente polarizado $100 = 127 + (-27)$.

5.1.2 Formatos

O Padrão IEEE 754 define quatro formatos de pontos-flutuantes, em dois grupos, básico e estendido. Cada grupo tem duas precisões, simples (*single*) e dupla (*double*). As diversas implementações do padrão podem, naturalmente, usar combinações dos formatos suportados.

[1] a qual deve ser maior que zero e menor que a base, respeitando a notação científica.

Definição 5.1.3 Quando sucessivas computações resultam em um valor grande demais (em módulo) para ser representado dentro do formato selecionado, diz-se que houve um ***overflow***, podendo ser negativo ou positivo, conforme o sinal do resultado. Se, pelo contrário, o valor for pequeno demais (em módulo), diz-se ter ocorrido um ***underflow***, podendo ser, analogamente, negativo ou positivo.

O *underflow* é um problema menos sério, pois ele apenas indica uma perda de precisão, que, em geral, é garantidamente próxima de zero.

Os formatos do grupo estendido dependem da implementação, ao passo que os formatos do grupo básico são pré-definidos pelo padrão, e são ilustrados nas figuras 5.1 e 5.2. Os números entre colchetes indicam o tamanho do campo, sendo que os bits mais significativos de cada campo ficam à esquerda, e os menos significativos, à direita.

Figura 5.1: Precisão Simples — 32 bits

Figura 5.2: Precisão Dupla — 64 bits

5.1.3 Valores especiais

O Padrão IEEE 754 reserva certas combinações de bits para denotar valores especiais no esquema de pontos-flutuantes. Estes valores serão descritos a seguir (o esquema completo é resumido na Tabela 5.1):

ZERO – Como foi mencionado acima, a parte inteira do significante não é armazenada, e assume-se que seja sempre 1. Uma conseqüência imediata disto é que se torna impossível a representação direta do número zero no padrão adotado. Este problema é contornado convencionando-se a denotação do zero quando todos os bits do expoente e da fração são 0. Note-se que nenhuma restrição é feita quanto ao sinal, e, portanto, há duas representações possíveis para o zero.

DESNORMALIZADO – Quando todos os bits do expoente são 0, mas a fração possui algum bit 1, assume-se que o bit implícito é 0 e não 1. O uso de pontos-flutuantes desnormalizados tem em vista a representação de números pequenos demais (em módulo) para serem representados com expoente normalizado. O valor numérico de um ponto-flutuante desnormalizado de sinal "s", expoente "e" e fração "f", é dado por:

$$(-1)^s \times 0.f \times 2^{-bias+1}$$

Por exemplo, supondo que o expoente tenha 8 bits (indo de 0 a $255 = 2^8 - 1$) e fazendo bias = 127, então a faixa de expoentes normalizados representáveis varia de $-127 = 0 - bias$ a $128 = 255 - bias$, e, portanto, o número 1.100×2^{-128} não tem representação normalizada, mas pode ser representado de forma desnormalizada como $0.011 \times 2^{-127+1} = 0.011 \times 2^{-126}$.

Nota: Uma implementação do Padrão IEEE 754 deve detectar operações cujos resultados não podem ser expressos com expoente normalizado, e fazer a desnormalização, sempre que possível, ao invés de sinalizar *underflow*. No SRTF, isto é feito pela função $normsub_{\mathbb{F}}$, descrita na seção 6.2.3.3.

INFINITO – Os valores $+\infty$ e $-\infty$ são denotados fazendo-se todo bit do expoente igual a 1 e todo bit da fração igual a 0. O bit de sinal distingue entre o infinito positivo e o infinito negativo. A capacidade de representar o infinito é útil porque ela permite que a computação continue após situações de *overflow*.

NAN – O ponto-flutuante NaN (*Not a Number*) é usado para se representar um valor que não representa um número real. NaN's são denotados com um expoente formado somente por bits 1 e uma fração diferente de zero. Existem duas categorias de NaN: QNaN (*Quiet NaN*), quando o bit mais significativo da fração é 1; e SNaN (*Signalling NaN*), quando o bit mais significativo da fração é 0.

Semanticamente, os QNaN's denotam operações *indeterminadas*, em geral, aquelas cujo resultado não está matematicamente definido, como, por exemplo, a raíz quadrada de um operando menor que 0. Por outro lado, os SNaN's são usados para denotar operações *inválidas*, e que requerem a sinalização de uma exceção, como, por exemplo, em situações de *overflow*.

Nota: Quando alguma operação ocasiona *overflow*, o resultado deve assumir o valor ∞, sem mudança de sinal, e um SNaN deve então sinalizar a exceção. No SRTF, por simplificação, *overflows* não disparam uma exceção diretamente, e também não retornam o ∞. Ao invés disso, um QNaN reservado é gerado, através

Sinal	Expoente	Fração	Valor
0	00..00	00..00	+0
0	00..00	00..01 a 11..11	Real Positivo Desnormalizado
0	00..01 a 11..10	00..00 a 11..11	Real Positivo Normalizado
0	11..11	00..00	$+\infty$
0	11..11	00..01 a 01..11	SNaN
0	11..11	10..00 a 11..11	QNaN
1	00..00	00..00	-0
1	00..00	00..01 a 11..11	Real Negativo Desnormalizado
1	00..01 a 11..10	00..00 a 11..11	Real Negativo Normalizado
1	11..11	00..00	$-\infty$
1	11..11	00..01 a 01..11	SNaN
1	11..11	10..00 a 11..11	QNaN

Tabela 5.1: Esquema de pontos-flutuantes [Hol05]

das funções $normadd_{\mathbb{F}}$ e $normmult_{\mathbb{F}}$, descritas nas seções 6.2.2.3 e 6.2.4.3, respectivamente. Neste caso, o QNaN se propaga pelas operações subseqüentes, permitindo, desta forma, que as aplicações de nível superior percebam e tratem a exceção.

5.1.4 Operações aritméticas

Várias operações são especificadas pelo Padrão IEEE 754, incluindo operações aritméticas, conversão entre formatos, conversão entre pontos-flutuantes e inteiros, raíz quadrada, resto etc. Suas definições devem ser atendidas por qualquer implementação do padrão.

As operações aritméticas de adição, subtração e multiplicação são de particular interesse, pois são justamente aquelas modeladas pelo sistema SRTF, proposto na Seção 6.2. Tais operações são definidas de maneira usual no Padrão IEEE 754. As únicas peculiaridades são para operações envolvendo valores especiais: No caso mais simples, toda operação com um NaN, gera um NaN como resultado; nos demais casos, vale o que está definido na Tabela 5.2.

Quando o resultado de uma operação aritmética não possui representação no formato selecionado (ou seja, quando o resultado é *infinitamente preciso*, segundo a terminologia do padrão), ele deve ser arredondado, de acordo com o modo de arredondamento selecionado. O Padrão IEEE 754 prevê que toda implementação deve fornecer quatro modos de arredondamento selecionáveis pelo usuário (pessoa, hardware ou software).

Operação	Resultado
$\infty + \infty$	∞
$\infty - \infty$	NaN
$\infty \times \infty$	∞
$\infty \times 0$	NaN

Tabela 5.2: Operações especiais

Os modos de arredondamento especificados no Padrão IEEE 754 são:

- Arredondamento para o mais próximo: O resultado deve ser o valor, dentro do formato, mais próximo do resultado infinitamente preciso. Se dois valores são igualmente próximos, então deve ser considerado aquele que tem o bit menos significativo igual a 0.
- Arredondamento para $+\infty$ ("para cima"): O resultado deve ser o valor, dentro do formato e possivelmente $+\infty$, mais próximo e não menor que o resultado infinitamente preciso;
- Arredondamento para $-\infty$ ("para baixo"): O resultado deve ser o valor, dentro do formato e possivelmente $-\infty$, mais próximo e não maior que o resultado infinitamente preciso;
- Arredondamento para 0: O resultado deve ser o valor, dentro do formato e possivelmente 0, mais próximo e não maior em módulo que o resultado infinitamente preciso.

Nota: O modo de arredondamento para o mais próximo é o padrão. Entretanto, nesta primeira versão do SRTF, apenas o arredondamento para 0 foi implementado, no caso, através da função $rshift$, descrita na seção 6.2.2.3. As conseqüências desta limitação serão discutidas no capítulo de conclusão.

5.2 Matemática Intervalar

Originada na década de 60, a Matemática Intervalar se apresenta como uma teoria matemática de considerável aplicação na computação científica, sobretudo na resolução de problemas numéricos. Nesta seção, serão introduzidos os principais conceitos relacionados à aritmética intervalar, bem como suas propriedades. Aqui serão considerados apenas os intervalos de números reais, para fins teóricos, mas na Seção 5.3 será mostrada uma proposta de implementação de intervalos.

A aritmética intervalar foi criada por R. E. Moore nos anos 1960 [Moo62], visando resolver problemas relacionados à falta de precisão dos números de ponto-flutuante (acumulação de erros de arredondamento, erros de truncamento etc).

Na abordagem intervalar, um determinado número real é representado através de um intervalo, que consiste, na verdade, de uma faixa de números reais, definida por seus extremos, contendo obviamente o número real a ser representado.

A análise intervalar tem sido desenvolvida como um ramo da análise numérica, visando encontrar algoritmos intervalares que produzam resultados numéricos precisos (veja [Moo79], e também [Rum88]). Tais algoritmos, naturalmente, fazem uso da aritmética intervalar, que será formalmente apresentada a seguir.

5.2.1 Introdução à aritmética intervalar

Definição 5.2.1 Seja $\mathbb{R}$ o conjunto dos números reais, e sejam $x_1, x_2 \in \mathbb{R}$ tais que $x_1 \leq x_2$, o conjunto $\{x \in \mathbb{R} \mid x_1 \leq x \leq x_2\}$ é um **intervalo de números reais** ou simplesmente um **intervalo**, denotado por $[x_1, x_2]$.

Definição 5.2.2 Denota-se por $\mathbb{IR}$ o **conjunto dos intervalos de números reais**, isto é,

$$\mathbb{IR} = \{[x_1, x_2] \mid x_1, x_2 \in \mathbb{R} \; e \; x_1 \leq x_2\}$$

Os elementos do conjunto acima serão denotados por letras latinas maiúsculas. Por exemplo, $A = [3, 7]$ e $B = [-1, 4]$.

Definição 5.2.3 Todo número real $x \in \mathbb{R}$ pode ser visto como um intervalo $X \in \mathbb{IR}$, onde $X = [x, x]$. Neste caso, X é chamado de **intervalo degenerado** ou **pontual**.

Uma das simplicidades da abordagem intervalar é que tanto as relações como as funções envolvendo intervalos são definidas a partir dos extremos.

Definição 5.2.4 Vendo cada intervalo da reta como um conjunto, a noção de **igualdade entre dois intervalos** é dada pela noção de igualdade entre conjuntos, ou seja, $A = B \Leftrightarrow (\forall x \in A \Rightarrow x \in B \; e \; \forall x \in B \Rightarrow x \in A)$. Desta forma, dados dois intervalos $A = [a_1, a_2]$ e $B = [b_1, b_2]$, $A = B \Leftrightarrow a_1 = b_1 \; e \; a_2 = b_2$.

Definição 5.2.5 (Operações Aritméticas entre Intervalos) Sejam os intervalos $X = [x_1, x_2]$ e $Y = [y_1, y_2]$, então:

1. $X + Y = [x_1, x_2] + [y_1, y_2] = [x_1 + y_1, x_2 + y_2]$
2. $X - Y = [x_1, x_2] - [y_1, y_2] = [x_1 - y_2, x_2 - y_1]$
3. $X \times Y = [x_1, x_2] \times [y_1, y_2] = [min\{x_1 \times y_1, x_1 \times y_2, x_2 \times y_1, x_2 \times y_2\}, max\{x_1 \times y_1, x_1 \times y_2, x_2 \times y_1, x_2 \times y_2\}]$

Nota: Assim como na Seção 5.1, a operação aritmética de divisão será omitida aqui, já que a mesma não é contemplada pelo modelo formal que está sendo proposto neste texto.

Teorema 5.2.1 *Sejam os intervalos* $X = [x_1, x_2]$ *e* $Y = [y_1, y_2]$*, então:*

1. $X + Y = \{r + s \mid r \in X \text{ e } s \in Y\}$
2. $X - Y = \{r - s \mid r \in X \text{ e } s \in Y\}$
3. $X \times Y = \{r \times s \mid r \in X \text{ e } s \in Y\}$

Prova: Veja [Lyr03], pg. 13. □

Nota: Observe-se que, ao representar operações da aritmética real através de operações da aritmética intervalar, essas últimas produzem exatamente a imagem das operações reais. Ou seja, a aritmética intervalar é

1. Correta, i.e. se $x \in X$ então $f(x) \in F(X)$, e
2. Ótima, i.e. $\{f(x) \mid x \in X\} = F(X)$.

Um tratamento matemático das propriedades de corretude e otimalidade pode ser visto em [SBA05].

5.2.2 Propriedades da aritmética intervalar

A seguir, serão enunciados alguns teoremas que ilustram as principais propriedades da aritmética intervalar.

Teorema 5.2.2 *A aritmética intervalar é subdistributiva, i.e., se* X*,* Y *e* Z *são intervalos, então*

$$X \times (Y + Z) \subseteq X \times Y + X \times Z.$$

Observe-se que, apesar da adição e a multiplicação intervalares serem comutativas e associativas, não vale a lei da distributividade. Além disso, têm-se $[0, 0]$ como identidade da adição intervalar, e $[1, 1]$ como identidade da multiplicação, mas não existem inversos aditivos e multiplicativos. Por exemplo:

$$[2, 4] - [2, 4] = [2, 4] + [-4, -2] = [-2, 2]$$

A falta de inverso implica numa séria dificuldade em se modelar a aritmética intervalar, pensando na lógica apresentada no Capítulo 3, dificuldade esta relacionada ao tratamento da igualdade e, conseqüentemente, à resolução de equações. O mesmo não ocorre com os sistemas de reescrita de termos, usados neste trabalho como proposta de um modelo formal para a computação intervalar, onde, como foi mostrado no Capítulo 4, as regras de dedução são mais simples do que aquelas para a lógica equacional.

Teorema 5.2.3 (Propriedades Algébricas da Soma em $\mathbb{IR}$) *Para todo $A = [a_1, a_2]$, $B = [b_1, b_2]$ e $C = [c_1, c_2] \in \mathbb{IR}$, vale:*

1. *Fechamento:* $A + B \in \mathbb{IR}$
2. *Associatividade:* $A + (B + C) = (A + B) + C$
3. *Comutatividade:* $A + B = B + A$
4. *Elemento Neutro:* $\exists! \mathcal{O} \in \mathbb{IR} \mid A + \mathcal{O} = \mathcal{O} + A = A \quad (\mathcal{O} = [0, 0])$

Prova: Veja [Lyr03], pg. 15. □

Teorema 5.2.4 (Propriedades Algébricas da Multiplicação em $\mathbb{IR}$) *Para todo $A, B, C \in \mathbb{IR}$, vale:*

1. *Fechamento:* $A \times B \in \mathbb{IR}$
2. *Associatividade:* $A \times (B \times C) = (A \times B) \times C$
3. *Comutatividade:* $A \times B = B \times A$
4. *Elemento Neutro:* $\exists! \mathcal{E} \in \mathbb{IR} \mid A \times \mathcal{E} = \mathcal{E} \times A = A \quad (\mathcal{E} = [1, 1])$

Prova: Veja [Lyr03], pg. 16. □

Vale lembrar que o conjunto $\mathbb{IR}$ não possui inverso multiplicativo, conforme enunciado anteriormente.

Teorema 5.2.5 (Monotonicidade da Aritmética Intervalar) *Para todo A, $B, C, D \in \mathbb{IR}$, tais que $A \subseteq C$ e $B \subseteq D$, vale:*

1. $A + B \subseteq C + D$
2. $-A \subseteq -C$
3. $A - B \subseteq C - D$

4. $A \times B \subseteq C \times D$

Prova: Veja [Lyr03], pg. 19. □

Teorema 5.2.6 (Propriedades de Inclusão) *Para todo* $A, B, C \in \mathbb{IR}$*, vale:*

1. $A + B = A + C \Rightarrow B = C$

2. $B - A = C - A \Rightarrow B = C$

3. $A + B \subseteq A + C \Rightarrow B \subseteq C$

4. $\forall \alpha, \beta \in \mathbb{R},\ (\alpha \times \beta) \times A = \alpha \times (\beta \times A)$

5. $\forall \alpha, \beta \in \mathbb{R},\ (\alpha + \beta) \times A \subseteq (\alpha \times A) + (\beta \times A)$

6. $\forall \alpha \in \mathbb{R},\ \alpha \times (B + C) = (\alpha \times B) + (\alpha \times C)$

Prova: Veja [Lyr03], pg. 19. □

5.3 Intervalos na prática

Na seção anterior foi dada uma definição matemática de intervalos de números reais, e das operações aritméticas sobre o conjunto de tais intervalos. Agora, será fornecida uma definição de intervalos cujos extremos são representados por pontos-flutuantes Padrão IEEE 754, descritos na Seção 5.1, a fim de prover uma abordagem computacional para a aritmética intervalar, que servirá de base para o sistema SRTI, proposto na Seção 6.3.

O principal objetivo consiste em demonstrar como as operações aritméticas sobre intervalos de pontos-flutuantes podem fornecer aproximações para as operações sobre intervalos de reais, sem perder a corretude e a otimalidade da aritmética intervalar, conforme nota final da Seção 5.2.1.

5.3.1 Noções preliminares

Os conceitos e algoritmos apresentados nesta seção foram adaptados de [HJE01] (veja também [Kea96] e [Wal90]).

Definição 5.3.1 Denota-se por $\mathbb{F}$ o **conjunto de pontos-flutuantes Padrão IEEE 754** cujos valores numéricos representam números reais.

Definição 5.3.2 Um **intervalo Padrão IEEE 754** é um intervalo de números reais cujos extremos são representados por elementos de $\mathbb{F}$, com a condição de que -0 não apareça como extremo esquerdo e $+0$ não apareça como extremo direito. Denota-se por $\langle a, b\rangle$ o intervalo Padrão IEEE 754 com extremos a e b.

A restrição quanto ao sinal do zero é uma mera convenção, com o fim de evitar ambiguidades e simplificar as fórmulas para operações aritméticas. Em outras abordagens, -0 e $+0$ podem aparecer em qualquer dos extremos, tendo diferente interpretação em cada caso.

Definição 5.3.3 O **conjunto dos intervalos Padrão IEEE 754** será denotado por $\mathbb{IF}$.

Uma operação envolvendo pontos-flutuantes pode resultar em um número real que não é um ponto-flutuante. Em tais casos, o Padrão IEEE 754 especifica que o resultado deve ser arredondado para o valor mais próximo que tenha representação, de acordo com o modo de arredondamento selecionado, como foi mencionado na Seção 5.1.4. Isto leva à seguinte diferenciação:

Definição 5.3.4 As operações de adição, subtração e multiplicação do Padrão IEEE 754 com arredondamento para baixo ($-\infty$) serão denotadas por $+_{lo}$, $-_{lo}$ e $\times_{lo}$, respectivamente. As mesmas operações com arredondamento para cima ($+\infty$) serão denotadas por $+_{hi}$, $-_{hi}$ e $\times_{hi}$.

O padrão também exige que toda operação atribua um valor à uma variável booleana *exact*, cujo valor será verdadeiro sss o resultado computado é igual ao resultado matematicamente definido.

5.3.2 Uma classificação de intervalos

A classificação de intervalos fornecida na Tabela 5.3, de acordo com o sinal dos elementos do intervalo, será útil para a distinção entre casos, para a operação aritmética intervalar de multiplicação.

5.3.3 Aritmética intervalar com o Padrão IEEE 754

O primeiro aspecto a ser considerado quanto à implementação de intervalos Padrão IEEE 754 está relacionado a arredondamentos. Felizmente, na aritmética intervalar, a necessidade de arredondamentos não leva a erro, ou seja, a corretude é mantida. O único efeito do arredondamento é a inclusão de valores ("lixo") que não seriam incluídos em caso de resultado exato.

Teorema 5.3.1 *[HJE01] Para todo conjunto de números reais, o intervalo Padrão IEEE mais estreito, contendo aquele conjunto, existe e é único.*

Classe	Descrição
M	A classe M ("Misto") é definida como o conjunto de intervalos contendo pelo menos um real positivo e um real negativo: $a < 0 < b$, para todo intervalo $\langle a, b\rangle$ na classe M.
Z	A classe Z ("Zero") é definida como o conjunto de intervalos não contendo nenhum real positivo e nenhum real negativo: $Z = \{\langle 0, 0\rangle\}$.
P	A classe P ("Positivo") é definida como o conjunto de intervalos contendo pelo menos um real positivo, mas nenhum negativo: $0 \leq a \leq b$ e $0 < b$, para todo intervalo $\langle a, b\rangle$ na classe P.
N	A classe N ("Negativo") é simétrica em relação à classe de P: $a \leq b \leq 0$ e $a < 0$, para todo intervalo $\langle a, b\rangle$ na classe N.

Tabela 5.3: Classificação de intervalos segundo o sinal [HJE01]

Prova: Considerando que $-\infty$ e $+\infty$ são números de ponto-flutuantes Padrão IEEE, então existe um intervalo Padrão IEEE 754 tal que contenha o conjunto de reais dado. Como o número de tais intervalos é finito e fechado sob intersecção, existe um mínimo intervalo Padrão IEEE 754 contendo aquele conjunto. □

Hickey [HJE01] ressalta que a importância do teorema acima está no fato de que a existência de um intervalo mínimo permite a definição da seguinte função:

Definição 5.3.5 Para todo conjunto α de reais, $\Gamma(\alpha)$ é o menor intervalo contendo α.

Definição 5.3.6 Diz-se que um conjunto β é uma **aproximação correta** de um conjunto α de reais se $\beta \supset \alpha$. Define-se o intervalo Padrão IEEE 754 $\Gamma(\alpha)$ como a **ótima aproximação IEEE 754** de α.

No teorema seguinte, Hickey demonstra como obter aproximações corretas e ótimas para as operações de adição, subtração e multiplicação:

Teorema 5.3.2 *[HJE01] Sejam $X = \langle a, b\rangle$ e $Y = \langle c, d\rangle$ intervalos Padrão IEEE não-vazios, então*

1. $\Gamma(\langle a, b\rangle + \langle c, d\rangle) = \langle a +_{lo} c, b +_{hi} d\rangle$;
2. $\Gamma(\langle a, b\rangle - \langle c, d\rangle) = \langle a -_{lo} d, b -_{hi} c\rangle$;
3. *As fórmulas na Figura 5.3 fornecem aproximações para a multiplicação.*

Classe de $\langle a, b\rangle$	Classe de $\langle c, d\rangle$	$\Gamma(\langle a, b\rangle \times \langle c, d\rangle)$
P	P	$\langle a \times_{lo} c, b \times_{hi} d\rangle$
M	P	$\langle a \times_{lo} d, b \times_{hi} d\rangle$
N	P	$\langle a \times_{lo} d, b \times_{hi} c\rangle$
P	M	$\langle b \times_{lo} c, b \times_{hi} d\rangle$
M	M	$\langle min(a \times_{lo} d, b \times_{lo} c), max(b \times_{hi} d, a \times_{hi} c)\rangle$
N	M	$\langle a \times_{lo} d, a \times_{hi} c\rangle$
P	N	$\langle b \times_{lo} c, a \times_{hi} d\rangle$
M	N	$\langle b \times_{lo} c, a \times_{hi} c\rangle$
N	N	$\langle b \times_{lo} d, a \times_{hi} c\rangle$
Z	P, M, N, Z	$\langle 0, 0\rangle$
P, M, N, Z	Z	$\langle 0, 0\rangle$
qualquer	$\emptyset$	$\emptyset$
$\emptyset$	qualquer	$\emptyset$

Figura 5.3: Multiplicação de intervalos Padrão IEEE [HJE01]

Prova: Veja [HJE01] pg. 1059.

Verifica-se um problema crucial no teorema acima: Ao definir os algoritmos, Hickey ignora o fato de que as operações aritméticas entre extremos dos intervalos podem resultar em pontos-flutuantes não numéricos (NaN's), ferindo, neste caso, a Definição 5.3.2. Este problema pode ser contornado generalizando-se a definição de Intervalo Padrão IEEE 754:

Definição 5.3.7 Um **INaN** ou **NaN intervalar** é um par de pontos-flutuantes Padrão IEEE 754 a e b, denotado por $\langle a, b\rangle$, onde pelo menos um dos elementos do par é um NaN. Seja $\mathbb{INAN}$ o **conjunto dos NaN's intervalares**. Um **intervalo Padrão IEEE 754 estendido** é qualquer elemento do conjunto $\mathbb{IF} \cup \mathbb{INAN}$.

Nota: Matematicamente, um INaN pode ser visto como o $\emptyset$, ou seja, um INaN não aproxima nenhum número real.

Como as operações envolvendo NaN's estão bem definidas pelo Padrão IEEE 754, e são captadas pelas regras do sistema SRTF subjacente, para pontos-flutuantes, as operações aritméticas entre extremos de intervalos, no SRTI, podem ser expressas pelas mesmas fórmulas do Teorema 5.3.2, porém aplicadas sobre o conjunto $\mathbb{IF} \cup \mathbb{INAN}$, tornando as regras do SRTI consideravelmente simples.

Naturalmente, o macro sistema SRTI herda as limitações das operações subjacentes, particularmente aquelas relacionadas ao arredondamento para cima, conforme nota sobre a função $rshift$ na Seção 5.1.4. Estas considerações serão retomadas, posteriormente, no capítulo de conclusão.

Este capítulo forneceu uma descrição dos objetos matemáticos que são modelados pelo sistema proposto nesta dissertação, começando com pontos-flutuantes, cujo modelo será apresentado na Seção 6.2 do capítulo a seguir, e terminando com a abordagem intervalar, para a qual será apresentada a sugestão de um modelo formal, na Seção 6.3 do mesmo capítulo.

Capítulo 6

Delineando um Modelo para a Computação Intervalar

6.1 A concepção de um modelo

Neste capítulo, serão apresentadas as contribuições particulares desta dissertação. Uma versão resumida deste trabalho encontra-se em [CS05].

Primeiramente, na Seção 6.2, será fornecido um sistema de reescrita de termos para a aritmética de ponto-flutuante binária, baseado no Padrão IEEE 754 descrito na Seção 5.1. Como a terminação da operação aritmética de divisão permanece como um problema aberto, apenas as operações de adição, subtração e multiplicação serão consideradas. Este sistema, chamado SRTF, é proposto como modelo formal para a computação de ponto-flutuante, constituindo um trabalho inicial para pesquisas nas áreas de especificação e verificação formal de sistemas que adotem a representação de ponto-flutuante para os números reais.

Em seguida, na Seção 6.3, o SRTF é estendido com regras para a aritmética intervalar descrita na Seção 5.2. O sistema resultante, chamado SRTI, é proposto como a **concepção inicial** de um modelo formal para a computação intervalar, que venha a ser correto e ótimo, de acordo com o que foi discutido na Seção 5.3, a fim de prover uma semântica operacional para sistemas que façam uso da abordagem intervalar como estratégia para representação computacional dos números reais.

Dado que um sistema de reescrita de termos constitui um sistema Post (veja [BL74]), espera-se também construir uma base para o desenvolvimento de um modelo de computabilidade para os números reais, tanto para a abordagem de ponto-flutuante como para a abordagem intervalar.

6.2 SRTF

Nesta seção são fornecidas as regras do Sistema de Reescrita de Termos para Ponto-flutuante (SRTF). Operações envolvendo dígitos, inteiros (listas de dígitos) e pontos-flutuantes (listas de dígitos de tamanho fixo concatenadas) são indexadas por "$\mathbb{D}$", "$\mathbb{Z}$" e "$\mathbb{F}$", respectivamente. O construtor de lista ":", bem como a aritmética sobre inteiros, foram fortemente inspiradas em [Ken95]. Algumas equações foram simplificadas em relação ao sistema de Kennaway [Ken95], já que, aqui, a independência de base não é um requisito; outra diferença que merece ser ressaltada está no "sentido" de construção dos termos: as listas no SRTF são construídas da direita para esquerda (dígitos à esquerda são mais significantes que dígitos à direita), a fim de modelar listas com zeros extras à esquerda.

Pontos-flutuantes são definidos pelo construtor ";", o qual concatena o dígito de sinal, a lista do expoente e a lista da fração. Respeitando o Padrão IEEE 754, o bit implícito e o ponto implícito do significante não são representados. Por questões de legibilidade, será convencionada a precisão da Figura 6.1, mas o SRTF pode ser adaptado, sem restrição alguma, para qualquer formato descrito na Seção 5.1.2, visto que as equações para aritmética foram implementadas com independência de precisão.

Figura 6.1: Precisão Convencionada — 9 bits

A Seção 6.2.1 fornece a interpretação dos construtores. As seções 6.2.2, 6.2.3 e 6.2.4 apresentam as operações aritméticas de adição, subtração e multiplicação, respectivamente, para dígitos, inteiros e pontos-flutuantes. No caso da aritmética de ponto-flutuante, a grande quantidade de regras está relacionada à diversidade de construtores, e às peculiaridades das operações envolvendo valores especiais (NaN's, infinito, números desnormalizados etc.), obrigando, muitas vezes, que determinadas operações sejam avaliadas caso a caso, para cada construtor, como é o caso, por exemplo, da função de comparação *cmp*, definida na Seção 6.2.5.

As constantes na Tabela 6.1 designam inteiros (listas) especiais que devem ser informadas ao sistema, conforme a precisão desejada, neste caso, três bits para o expoente e cinco para a fração, conforme a Figura 6.1. Por causa da polarização do expoente (veja Definição 5.1.2), o sistema deve tratar de forma especial o expoente e_esp, o qual é interpretado como $1 - bias$, e, portanto, é semanticamente equivalente ao expoente desnormalizado e_min, o qual é interpretado como $-bias + 1$. A constante f_zero, por sua vez, representa uma lista de zeros estourando a precisão adotada.

Nome	Valor	Descrição
e_size	$(END : 1) : 1$	Bits para o expoente.
f_size	$((END : 1) : 0) : 1$	Bits para a fração.
e_min	$((END : 0) : 0) : 0$	Expoente mínimo.
e_max	$((END : 1) : 1) : 1$	Expoente máximo.
e_esp	$((END : 0) : 0) : 1$	Expoente especial.
f_min	$((((END : 0) : 0) : 0) : 0) : 0$	Fração mínima.
f_max	$((((END : 1) : 1) : 1) : 1) : 1$	Fração máxima.
f_zero	$(((((END : 0) : 0) : 0) : 0) : 0) : 0$	Lista de zeros.

Tabela 6.1: Constantes dependentes de precisão

É importante observar que o uso de tais constantes não deve ser visto como uma limitação do sistema. O próprio Padrão IEEE 754 prevê o uso de parâmetros para seleção de formato (precisão). Além das constantes, usam-se as variáveis de lista "e_any" e "f_any" como abreviações para expoentes arbitrários da forma "$((END : e_0) : e_1) : e_2$" e frações arbitrárias "$((((END : f_0) : f_1) : f_2) : f_3) : f_4$", respectivamente, onde e_i e f_i são variáveis de dígito.

6.2.1 Construtores

6.2.1.1 Dígitos

$$\|0\| = 0$$
$$\|1\| = 1$$
$$\|10\| = 2$$

Deve-se notar que adoção do dígito extra "10" constitui uma mera conveniência técnica adotada por [Ken95], visando simplificar a manipulação do "vai um" em resultados intermediários de operações aritméticas de adição.

6.2.1.2 Inteiros

Sejam d e l variáveis de dígito e lista de dígitos, respectivamente.

/* Inteiro Positivo */
$$\|l : d\| = \|d\| + 2 \times \|l\|$$

/* Inteiro Negativo */
$$\|MINUS(l)\| = -\|l\|$$

/* Fim de Lista */
$$\|END\| = 0$$

6.2.1.3 Pontos-flutuantes

Sejam $s \in \{0,1\}$, $f\_any = ((((END : f_0) : f_1) : f_2) : f_3) : f_4$, onde $f_i \in \{0,1\}$, e $e\_any = ((END : e_0) : e_1) : e_2$, onde $e_i \in \{0,1\}$. Os construtores de pontos-flutuantes, a seguir, refletem o esquema apresentado na Tabela 5.1.

$$\text{/* Zero */}$$
$$\|s; e\_min; f\_min\| = 0$$

$$\text{/* Número Desnormalizado */}$$
$$\|s; e\_min; f\_any\| = (-1)^{\|s\|} \times (0 + \|f\_any\| \times 2^{-f\_size}) \times 2^{-bias+1},$$
$$\text{se } (f\_any \neq f\_min)$$

$$\text{/* Número Normalizado */}$$
$$\|s; e\_any; f\_any\| = (-1)^{\|s\|} \times (1 + \|f\_any\| \times 2^{-f\_size}) \times 2^{\|e\_any\|-bias},$$
$$\text{se } (f\_any \neq f\_min) \text{ e } (f\_any \neq f\_max)$$

$$\text{/* Infinito */}$$
$$\|s; e\_max; f\_min\| = (-1)^{\|s\|} \times \infty$$

$$\text{/* Signalling NaN: “inválido” */}$$
$$\|s; e\_max; f\_any\| = SNaN,$$
$$\text{se } (f_0 = 0) \text{ e } (f\_any \neq f\_min)$$

$$\text{/* Quiet NaN: “indeterminado” */}$$
$$\|s; e\_max; f\_any\| = QNaN$$
$$\text{se } (f_0 = 1)$$

6.2.2 Adição

6.2.2.1 Dígitos

A adição de dígitos ($+_{\mathbb{D}}$) é definida apenas para os dígitos “0” e “1”, já que o dígito extra “10” é usado apenas como resultado intermediário e jamais aparece como operando de $+_{\mathbb{D}}$, como será visto nas regras subseqüentes (analogamente, o dígito “10” não será considerado para os casos da subtração e da multiplicação).

$$\text{/* Adição de dígitos */}$$
$$0 +_{\mathbb{D}} 0 \rightarrow 0$$
$$0 +_{\mathbb{D}} 1 \rightarrow 1$$
$$1 +_{\mathbb{D}} 0 \rightarrow 1$$
$$1 +_{\mathbb{D}} 1 \rightarrow 10$$

$$\|t +_{\mathbb{D}} t'\| = \|t\| + \|t'\|$$

6.2.2.2 Inteiros

Seguem as regras para adição de inteiros, algumas das quais fazem uso do símbolo funcional $-_{\mathbb{Z}}$ que será apresentado na Seção 6.2.3.

$$
\begin{gathered}
/^* \text{ Incremento de inteiros não-negativos } ^*/ \\
inc(END) \rightarrow END : 1 \\
inc(x : 0) \rightarrow x : 1 \\
inc(x : 1) \rightarrow inc(x) : 0
\end{gathered}
$$

$$\|inc(t)\| = \|t\| + 1$$

$$
\begin{gathered}
/^* \text{ Normalização de inteiros: remove o dígito impróprio “10” } ^*/ \\
normadd_{\mathbb{Z}}(x, 0) \rightarrow x : 0 \\
normadd_{\mathbb{Z}}(x, 1) \rightarrow x : 1 \\
normadd_{\mathbb{Z}}(x, 10) \rightarrow inc(x) : 0
\end{gathered}
$$

$$\|normadd_{\mathbb{Z}}(t, t')\| = \|t'\| + 2 \times \|t\|$$

$$
\begin{gathered}
/^* \text{ Adição de inteiros } ^*/ \\
END +_{\mathbb{Z}} END \rightarrow END \\
END +_{\mathbb{Z}} (x' : y') \rightarrow x' : y' \\
END +_{\mathbb{Z}} MINUS(y') \rightarrow MINUS(y') \\
(x : y) +_{\mathbb{Z}} END \rightarrow x : y \\
(x : y) +_{\mathbb{Z}} (x' : y') \rightarrow normadd_{\mathbb{Z}}(x +_{\mathbb{Z}} x', y +_{\mathbb{D}} y') \\
(x : y) +_{\mathbb{Z}} MINUS(y') \rightarrow (x : y) -_{\mathbb{Z}} y' \\
MINUS(x) +_{\mathbb{Z}} END \rightarrow MINUS(x) \\
MINUS(x) +_{\mathbb{Z}} (x' : y') \rightarrow (x' : y') -_{\mathbb{Z}} x \\
MINUS(x) +_{\mathbb{Z}} MINUS(y') \rightarrow MINUS(x +_{\mathbb{Z}} y')
\end{gathered}
$$

$$\|t +_{\mathbb{Z}} t'\| = \|t\| + \|t'\|$$

6.2.2.3 Pontos-flutuantes

Alguns problemas surgem quando se formaliza aritmética sobre pontos-flutuantes Padrão IEEE 754. O primeiro deles consiste em representar sintaticamente o dígito implícito, não capturado pelos construtores, mas necessário para as operações. Para isto, são fornecidas as regras *put* e *cut*.

$$
\begin{gathered}
/^* \text{ Inclui o bit implícito } ^*/ \\
put(END) \rightarrow END : 1 \\
put(x : y) \rightarrow put(x) : y
\end{gathered}
$$

$$\|put(t)\| = \|t\| + \|1\| \times 2^{\|length(t)\|}$$

$$/^* \text{ Exclui o bit implícito } */$$
$$cut(END : y) \rightarrow END$$
$$cut((w : x) : y) \rightarrow cut(w : x) : y$$

$$\|cut(t)\| = \|t\| - \|1\| \times 2^{\|length(t)\|-1}$$

Quando se realiza adição e subtração de pontos-flutuantes, deve-se igualar os expoentes. A fim de minimizar erros de arredondamento, o menor expoente é incrementado ao maior, e nunca o contrário. Sintaticamente, os expoentes são incrementados pela função $rshift$.

$$/^* \text{ Incrementa expoente:}$$
$$\text{a fração é deslocada, truncando-se os bits à direita}$$
$$\text{(arredondamento para 0!) } */$$
$$rshift(x, e\_min) \rightarrow x$$
$$rshift(x : y, z) \rightarrow rshift(x, z -_{\mathbb{Z}} (END : 1))$$

$$\|rshift(t, t')\| = \|t\| \times 2^{1-\|t'\|}$$

A função $symm$ é usada para se obter o termo do ponto-flutuante simétrico.

$$/^* \text{ ...alterna sinal } */$$
$$symm(0; x; y) \rightarrow 1; x; y$$
$$symm(1; x; y) \rightarrow 0; x; y$$

$$\|symm(t)\| = -\|t\|$$

A função seguinte gera uma lista, informando o tamanho da lista de entrada.

$$/^* \text{ ...calcula tamanho } */$$
$$length(END) \rightarrow END : 0$$
$$length(x : y) \rightarrow (END : 1) +_{\mathbb{Z}} length(x)$$
$$length(MINUS(x)) \rightarrow length(x)$$

Durante a computação, as máquinas precisam mapear expressões de pontos-flutuantes em objetos[1] que não são, necessariamente, pontos-flutuantes Padrão IEEE 754. Em seguida, tais objetos devem ser formatados conforme a precisão selecionada. No SRTF, os resultados intermediários das operações aritméticas de adição, subtração e multiplicação são formatados através das funções $normadd_{\mathbb{F}}$, $normsub_{\mathbb{F}}$ e $normmult_{\mathbb{F}}$, respectivamente.

A função $normadd_{\mathbb{F}}$ é definida em quatro casos, dependendo do valor do expoente e do tamanho da fração do resultado:

[1] Resultados de operações unárias ou binárias, referidos no Padrão IEEE 754 como "destinos".

$$/^* \text{ ...em caso de expoente normalizado } */$$
$$normadd_{\mathbb{F}}(s; x; y) \rightarrow$$
$$s; x +_{\mathbb{Z}} (length(y) -_{\mathbb{Z}} f\_size); rshift(y, length(y) -_{\mathbb{Z}} f\_size),$$
$$\text{se } (x \neq e\_min) \text{ e } ((x +_{\mathbb{Z}} (length(y) -_{\mathbb{Z}} f\_size)) \leq e\_max)$$

Nota: A diferença $length(y) -_{\mathbb{Z}} f\_size$ indica o quanto a fração y estourou a precisão. Sendo x um expoente normalizado ($x \neq e\_min$), aplica-se $rshift$ em y, como se o ponto implícito do significante fosse deslocado para a esquerda. O expoente é proporcionalmente aumentado.

$$/^* \text{ ...em caso de expoente desnormalizado } */$$
$$normadd_{\mathbb{F}}(s; e\_min; y) \rightarrow s; e\_min; y,$$
$$\text{se } (length(y) = f\_size)$$

Nota: Se a fração y não estourou a precisão, não há nada a fazer, e o ponto-flutuante permanece desnormalizado (expoente e_min).

$$/^* \text{ ...em caso de precisar normalizar o expoente } */$$
$$normadd_{\mathbb{F}}(s; e\_min; y) \rightarrow s; e\_esp; cut(y),$$
$$\text{se } (length(y) > f\_size)$$

Nota: Se a fração y estourou a precisão, significa que o resultado da soma foi um número normalizado. Ao invés de aplicar $rshift$ em y, neste caso, deve-se apenas extrair o bit implícito com cut. O expoente, que era e_min, passa então a ser e_esp.

$$/^* \text{ ...em caso de } overflow\text{: gera QNaN } */$$
$$normadd_{\mathbb{F}}(s; x; y) \rightarrow s; e\_max; f\_max,$$
$$\text{se } (x +_{\mathbb{Z}} (length(y) -_{\mathbb{Z}} f\_size)) > e\_max$$

Nota: $s; e\_max; f\_max$ é um NaN reservado do SRTF.

As regras a seguir descrevem a adição de pontos-flutuantes, começando pelos casos triviais:

$$/^* \text{ Zero Pos + Qualquer Pos = Qualquer Pos } */$$
$$0; e\_min; f\_min +_{\mathbb{F}} 0; e'\_any; f'\_any \rightarrow 0; e'\_any; f'\_any$$

$$/^* \text{ NaN Pos + Qualquer Pos = NaN Pos } */$$
$$0; e\_max; f\_any +_{\mathbb{F}} 0; e'\_any; f'\_any \rightarrow 0; e\_max; f\_any,$$
$$\text{se } (f\_any \neq f\_min)$$

/* Infinito Pos + Núm Pos = Infinito Pos */
$0; e\_max; f\_min +_{\mathbb{F}} 0; e'\_any; f'\_any \rightarrow 0; e\_max; f\_min,$
se $(e'\_any \neq e\_max)$

/* Infinito Pos + Infinito Pos = Infinito Pos */
$0; e\_max; f\_min +_{\mathbb{F}} 0; e\_max; f\_min \rightarrow 0; e\_max; f\_min$

/* Infinito Pos + NaN Pos = NaN Pos */
$0; e\_max; f\_min +_{\mathbb{F}} 0; e\_max; f'\_any \rightarrow 0; e\_max; f'\_any,$
se $(f'\_any \neq f\_min)$

/* Núm Pos + Zero Pos = Núm Pos */
$0; e\_any; f\_any +_{\mathbb{F}} 0; e\_min; f\_min \rightarrow 0; e\_any; f\_any,$
se $(e\_any \neq e\_max)$

/* Núm Pos + Infinito Pos = Infinito Pos */
$0; e\_any; f\_any +_{\mathbb{F}} 0; e\_max; f\_min \rightarrow 0; e\_max; f\_min,$
se $(e\_any \neq e\_max)$

/* Núm Pos + NaN Pos = NaN Pos */
$0; e\_any; f\_any +_{\mathbb{F}} 0; e\_max; f'\_any \rightarrow 0; e\_max; f'\_any,$
se $(e\_any \neq e\_max)$ e $(f'\_any \neq f\_min)$

Casos não-triviais da adição de pontos-flutuantes:

/* Núm Pos Norm + Núm Pos Norm: $e \geq e'$ */
$0; e\_any; f\_any +_{\mathbb{F}} 0; e'\_any; f'\_any \rightarrow$
$normadd_{\mathbb{F}}(0; e\_any; cut(put(f\_any) +_{\mathbb{Z}} rshift(put(f'\_any), e\_any -_{\mathbb{Z}} e'\_any)),$
se $(e\_any \neq e\_min)$ e $(e\_any \neq e\_max)$ e $(e'\_any \neq e\_min)$ e
$(e'\_any \neq e\_max)$ e $(e\_any \geq e'\_any)$

Nota: Cada bit implícito é concatenado à sua respectiva fração, através de *put*. O maior expoente (no caso, e_any) é tomado como o expoente da soma e aplica-se $rshift$ em f'_any para igualar os expoentes. Por fim, a função *cut* é usada para extrair o bit implícito do resultado. A próxima regra é análoga.

/* Núm Pos Norm + Núm Pos Norm: $e < e'$ */
$0; e\_any; f\_any +_{\mathbb{F}} 0; e'\_any; f'\_any \rightarrow$
$normadd_{\mathbb{F}}(0; e'\_any; cut(rshift(put(f'\_any),$
$e'\_any -_{\mathbb{Z}} e\_any) +_{\mathbb{Z}} put(f\_any))),$
se $(e\_any \neq e\_min)$ e $(e\_any \neq e\_max)$ e $(e'\_any \neq e\_min)$ e
$(e'\_any \neq e\_max)$ e $(e\_any < e'\_any)$

$$\begin{gathered}
/^* \text{ Núm Pos Norm + Núm Pos Desnorm: } e = e_esp \ ^*/ \\
0; e_any; f_any +_{\mathbb{F}} 0; e_min; f'_any \rightarrow \\
normadd_{\mathbb{F}}(0; e_any; cut(put(f_any) +_{\mathbb{Z}} f'_any))), \\
\text{se } (e_any \neq e_min) \text{ e } (e_any \neq e_max) \text{ e } (f'_any \neq f_min) \text{ e} \\
(e_any = e_esp)
\end{gathered}$$

Nota: Se e_any é o expoente especial e_esp e o outro expoente é e_min, o sistema não deve tentar igualar os expoentes, pois eles são semanticamente os mesmos. A diferença está no bit implícito: 1, no caso de e_esp; e 0, no caso de e_min. Por isso não se aplica put em f'_any.

$$\begin{gathered}
/^* \text{ Núm Pos Norm + Núm Pos Desnorm: } e > e_esp \ ^*/ \\
0; e_any; f_any +_{\mathbb{F}} 0; e_min; f'_any \rightarrow \\
normadd_{\mathbb{F}}(0; e_any; cut(put(f_any) +_{\mathbb{Z}} rshift(f'_any, e_any -_{\mathbb{Z}} e_esp)), \\
\text{se } (e_any \neq e_min) \text{ e } (e_any \neq e_max) \text{ e } (f'_any \neq f_min) \text{ e} \\
(e_any > e_esp)
\end{gathered}$$

Nota: Agora, o sistema deve igualar os expoentes, mas a diferença entre os expoentes não é dada por $e\_any -_{\mathbb{Z}} e\_min$, mas sim por $e\_any -_{\mathbb{Z}} e\_esp$, considerando que o bit implícito de f'_any é 0, e não 1.

$$\begin{gathered}
/^* \text{ Núm Pos Desnorm + Núm Pos Norm: } ^*/ \\
0; e_min; f_any +_{\mathbb{F}} 0; e'_any; f'_any \rightarrow \\
0; e'_any; f'_any +_{\mathbb{F}} 0; e_min; f_any, \\
\text{se } (f_any \neq f_min) \text{ e } (e'_any \neq e_min) \text{ e } (e'_any \neq e_max)
\end{gathered}$$

Nota: Esta regra fornece contra-partida para as duas regras anteriores.

$$\begin{gathered}
/^* \text{ Núm Pos Desnorm + Núm Pos Desnorm: } ^*/ \\
0; e_min; f_any +_{\mathbb{F}} 0; e_min; f'_any \rightarrow \\
normadd_{\mathbb{F}}(0; e_min; f_any +_{\mathbb{Z}} f'_any), \\
\text{se } (f_any \neq f_min) \text{ e } (f'_any \neq f_min)
\end{gathered}$$

Nota: Quando ambos os pontos-flutuantes são desnormalizados, basta somar as frações.

A fim de evitar regras redundantes e passos de reescrita cíclicos, a adição envolvendo operandos com sinais diferentes é mapeada na subtração correspondente:

$$\text{/* Positivo + Negativo */}$$
$$0; e\_any; f\_any +_{\mathbb{F}} 1; e'\_any; f'\_any \rightarrow 0; e\_any; f\_any -_{\mathbb{F}} 0; e'\_any; f'\_any$$

$$\text{/* Negativo + Positivo */}$$
$$1; e\_any; f\_any +_{\mathbb{F}} 0; e'\_any; f'\_any \rightarrow 0; e'\_any; f'\_any -_{\mathbb{F}} 0; e\_any; f\_any$$

A soma de dois operandos negativos quaisquer é mapeada na soma dos respectivos simétricos (positivos) e, em seguida, troca-se o sinal do resultado:

$$\text{/* Negativo + Negativo */}$$
$$1; e\_any; f\_any +_{\mathbb{F}} 1; e'\_any; f'\_any \rightarrow$$
$$symm(0; e\_any; f\_any +_{\mathbb{F}} 0; e'\_any; f'\_any)$$

6.2.3 Subtração

6.2.3.1 Dígitos

$$\text{/* Subtração de dígitos */}$$
$$0 -_{\mathbb{D}} 0 \rightarrow 0$$
$$0 -_{\mathbb{D}} 1 \rightarrow MINUS(1)$$
$$1 -_{\mathbb{D}} 0 \rightarrow 1$$
$$1 -_{\mathbb{D}} 1 \rightarrow 0$$

$$\|t -_{\mathbb{D}} t'\| = \|t\| - \|t'\|$$

6.2.3.2 Inteiros

$$\text{/* Decremento de inteiros positivos */}$$
$$dec(x:1) \rightarrow x:0$$
$$dec(x:0) \rightarrow dec(x):1$$

$$\|dec(t)\| = \|t\| - 1$$

$$\text{/* Normalização de inteiros: remove dígitos negativos de listas */}$$
$$normsub_{\mathbb{Z}}(END, 0) \rightarrow END$$
$$normsub_{\mathbb{Z}}(END, 1) \rightarrow END:1$$
$$normsub_{\mathbb{Z}}(END, MINUS(1)) \rightarrow MINUS(END:1)$$
$$normsub_{\mathbb{Z}}(x:y, 0) \rightarrow (x:y):0$$
$$normsub_{\mathbb{Z}}(x:y, 1) \rightarrow (x:y):1$$
$$normsub_{\mathbb{Z}}(x:y, MINUS(1)) \rightarrow dec(x:y):1$$
$$normsub_{\mathbb{Z}}(MINUS(x), 0) \rightarrow MINUS(x:0)$$
$$normsub_{\mathbb{Z}}(MINUS(x), 1) \rightarrow MINUS(dec(x):1)$$
$$normsub_{\mathbb{Z}}(MINUS(x), MINUS(1)) \rightarrow MINUS(x:1)$$

$$\|normsub_{\mathbb{Z}}(t,t')\| = \|t'\| + 2 \times \|t\|$$

/* Subtração de inteiros */
$$END -_{\mathbb{Z}} END \rightarrow END$$
$$END -_{\mathbb{Z}} (x:y) \rightarrow MINUS(x:y)$$
$$END -_{\mathbb{Z}} MINUS(y') \rightarrow y'$$
$$(x:y) -_{\mathbb{Z}} END \rightarrow (x:y)$$
$$(x:y) -_{\mathbb{Z}} (x':y') \rightarrow normsub_{\mathbb{Z}}(x -_{\mathbb{Z}} x', y -_{\mathbb{D}} y')$$
$$(x:y) -_{\mathbb{Z}} MINUS(y') \rightarrow (x:y) +_{\mathbb{Z}} y'$$
$$MINUS(x) -_{\mathbb{Z}} END \rightarrow MINUS(x)$$
$$MINUS(x) -_{\mathbb{Z}} (x':y') \rightarrow MINUS(x +_{\mathbb{Z}} (x':y'))$$
$$MINUS(x) -_{\mathbb{Z}} MINUS(y') \rightarrow y' -_{\mathbb{Z}} x$$

$$\|t -_{\mathbb{Z}} t'\| = \|t\| - \|t'\|$$

6.2.3.3 Pontos-flutuantes

A subtração é um pouco mais complexa do que a adição. Ao invés de truncar frações largas demais e incrementar seus respectivos expoentes, o sistema deve tratar as frações com zeros extras à esquerda. A presença de zeros extras à esquerda em operações é conseqüência das alterações feitas no sistema de Kennaway. Isto não afeta a terminação, porque o SRTF sempre lida com listas (inteiros) de tamanho fixo.

Visto que a subtração de duas listas dadas resulta em uma lista de mesmo tamanho, possivelmente com zeros extras à esquerda, tudo que se tem a fazer é "mover" da esquerda para a direita os zeros extras nas frações, e decrementar os respectivos expoentes.

As regras abaixo descrevem funções para decrementar expoentes:

/* ...calcula soma de dígitos */
$$digsum(END) \rightarrow 0$$
$$digsum(x:0) \rightarrow digsum(x)$$
$$digsum(x:1) \rightarrow 1 + digsum(x)$$

$$\|digsum((...((d_0:d_1):d_2):...):d_n)\| = \|d_0\| + \|d_1\| + \|d_2\| + ... + \|d_n\|$$

/* ...$lshift$ trunca todos os 0's à esquerda; para isto, aplica-se $digsum$ em z para informar a quantidade esperada de 1's. */
$$lshift(x:y,0) \rightarrow END$$
$$lshift(x:0,z) \rightarrow lshift(x,z):0$$
$$lshift(x:1,z) \rightarrow lshift(x, z -_{\mathbb{Z}} (END:1)):1$$

$$/^* \ldots fit \text{ trunca exatamente a quantidade informada de 0's. } */$$
$$fit(x : y, f\_size) \rightarrow END$$
$$fit(x : y, z) \rightarrow fit(x, z +_{\mathbb{Z}} (END : 1)) : y,$$
$$\text{se } (z < f\_size)$$

A regras a seguir descrevem funções auxiliares para normalização de fração.

$$/^* \ldots leading \text{ conta os 0's à esquerda removidos por } lshift. */$$
$$leading(x : y) \rightarrow length(x : y) -_{\mathbb{Z}} length(lshift(x : y, digsum(x : y)))$$

$$/^* \ldots \text{e } rpad \text{ completa o lado direito com aqueles 0's. } */$$
$$rpad(x : y, 0) \rightarrow x : y$$
$$rpad(x : y, z) \rightarrow rpad((x : y) : 0, z -_{\mathbb{Z}} (END : 1))$$

A função $normsub_{\mathbb{F}}$ é definida em cinco casos:

$$/^* \ldots \text{em caso de expoente normalizado } */$$
$$normsub_{\mathbb{F}}(s; w : x; y : z) \rightarrow s; (w : x) -_{\mathbb{Z}} leading(y : z);$$
$$rpad(cut(lshift(y : z,\ digsum(y : z))), leading(y : z)),$$
$$\text{se } ((w : x) \neq e\_min) \text{ e } (((w : x) -_{\mathbb{Z}} leading(y : z)) \geq e\_esp)$$

Nota: Os 0's à esquerda da fração $y : z$ são removidos por $lshift$, que vai reconstruindo a lista até encontrar o último dígito 1, o qual é indicado por $digsum$. A remoção dos 0's à esquerda permite deslocar o ponto implícito do significante para a direita, até chegar no bit implícito, o qual é extraído, em seguida, por cut. A quantidade de 0's removidos é guardada em $leading$, cujo valor de retorno é aplicado em $rpad$ para completar a fração com 0's à direita. O valor de $leading$ também é usado para diminuir o expoente $w : x$.

$$/^* \ldots \text{em caso de precisar desnormalizar o expoente } */$$
$$normsub_{\mathbb{F}}(s; w : x; y : z) \rightarrow s; e\_min;$$
$$rpad(fit(cut(y : z),\ (w : x) -_{\mathbb{Z}} e\_esp), (w : x) -_{\mathbb{Z}} e\_esp),$$
$$\text{se } ((w : x) \neq e\_min) \text{ e } (((w : x) -_{\mathbb{Z}} leading(y : z)) < e\_esp)$$

Nota: Ao contrário da adição, em que a soma de dois pontos-flutuantes desnormalizados poderia resultar em um ponto-flutuante normalizado, aqui a diferença entre dois pontos-flutuantes normalizados pode resultar em um ponto-flutuante desnormalizado. Isto ocorre quando a diferença $(w : x) -_{\mathbb{Z}} leading(y : z)$ é menor do que e_esp. A aplicação de fit, em lugar de $lshift$, assegura que o ponto implícito do significante vai ser deslocado para a direita somente até que o expoente alcance e_esp (lembrando que e_min é apenas uma convenção sintática, pois semanticamente o número desnormalizado possui expoente e_esp, com bit implícito 0). O uso de $lshift$, aqui, resultaria em uma lista do tipo MINUS(x) para o expoente, ferindo o construtor ";" e o Padrão IEEE 754.

$$/* \text{ ...em caso de dar zero } */$$
$$normsub_{\mathbb{F}}(s; w : x; f\_zero) \rightarrow s; e\_min; cut(f\_zero),$$
$$\text{se } (w : x) \neq e\_min$$

Nota: O uso de f_zero, no lugar de f_min se faz necessário porque a operação $-_{\mathbb{F}}$, no caso de pontos-flutuantes com expoentes normalizados, passa para $normsub_{\mathbb{F}}$ a fração com o bit implícito concatenado. O bit implícito é removido, aqui, por *cut*.

$$/* \text{ ...em caso de expoente desnormalizado } */$$
$$normsub_{\mathbb{F}}(s; e\_min; y : z) \rightarrow s; e\_min; y : z$$

Nota: Neste caso, o bit implícito não é introduzido por $-_{\mathbb{F}}$.

$$/* \text{ ...em caso de fração negativa } */$$
$$normsub_{\mathbb{F}}(s; w : x; MINUS(z)) \rightarrow symm(normsub_{\mathbb{F}}(s; w : x; z))$$

Nota: Trivialmente, o sinal da fração passa a ser o sinal do ponto-flutuante.

Por fim, seguem as regras para subtração de pontos-flutuantes, começando pelos casos triviais:

$$/* \text{ Zero Pos - Qualquer Pos = Qualquer Neg } */$$
$$0; e\_min; f\_min -_{\mathbb{F}} 0; e'\_any; f'\_any \rightarrow 1; e'\_any; f'\_any$$

$$/* \text{ NaN Pos - Qualquer Pos = NaN Pos } */$$
$$0; e\_max; f\_any -_{\mathbb{F}} 0; e'\_any; f'\_any \rightarrow 0; e\_max; f\_any,$$
$$\text{se } (f\_any \neq f\_min)$$

$$/* \text{ Infinito Pos - Núm Pos = Infinito Pos } */$$
$$0; e\_max; f\_min -_{\mathbb{F}} 0; e'\_any; f'\_any \rightarrow 0; e\_max; f\_min,$$
$$\text{se } (e'\_any \neq e\_max)$$

$$/* \text{ Infinito Pos - Infinito Pos = QNaN Pos } */$$
$$0; e\_max; f\_min -_{\mathbb{F}} 0; e\_max; f\_min) \rightarrow 0; e\_max; f\_max$$

$$/* \text{ Infinito Pos - NaN Pos = NaN Pos } */$$
$$0; e\_max; f\_min -_{\mathbb{F}} 0; e\_max; f'\_any \rightarrow 0; e\_max; f'\_any,$$
$$\text{se } (f'\_any \neq f\_min)$$

$$/^* \text{ Núm Pos - Zero Pos = Núm Pos } ^*/$$
$$0; e\_any; f\_any -_{\mathbb{F}} 0; e\_min; f\_min \rightarrow 0; e\_any; f\_any,$$
$$\text{se } (e\_any \neq e\_max)$$

$$/^* \text{ Núm Pos - Infinito Pos = Infinito Neg } ^*/$$
$$0; e\_any; f\_any -_{\mathbb{F}} 0; e\_max; f\_min \rightarrow 1; e\_max; f\_min,$$
$$\text{se } (e\_any \neq e\_max)$$

$$/^* \text{ Núm Pos - NaN Pos = NaN Neg } ^*/$$
$$0; e\_any; f\_any -_{\mathbb{F}} 0; e\_max; f'\_any \rightarrow 1; e\_max; f'\_any,$$
$$\text{se } (e\_any \neq e\_max) \text{ e } (f'\_any \neq f\_min)$$

Casos não-triviais da subtração de pontos-flutuantes:

$$/^* \text{ Núm Pos Norm - Núm Pos Norm: } e \geq e' \ ^*/$$
$$0; e\_any; f\_any -_{\mathbb{F}} 0; e'\_any; f'\_any \rightarrow$$
$$normsub_{\mathbb{F}}(0; e\_any;\ put(f\_any) -_{\mathbb{Z}} rshift(put(f'\_any), e\_any -_{\mathbb{Z}} e'\_any)),$$
$$\text{se } (e\_any \neq e\_min) \text{ e } (e\_any \neq e\_max) \text{ e } (e'\_any \neq e\_min) \text{ e}$$
$$(e'\_any \neq e\_max) \text{ e } (e\_any \geq e'\_any)$$

Nota: Assim como na adição, cada bit implícito é concatenado à sua respectiva fração, através de *put*. O maior expoente (no caso, e_any) é tomado como o expoente da diferença e aplica-se $rshift$ em f'_any para igualar os expoentes. A próxima regra é análoga.

$$/^* \text{ Núm Pos Norm - Núm Pos Norm: } e < e' \ ^*/$$
$$0; e\_any; f\_any -_{\mathbb{F}} 0; e'\_any; f'\_any \rightarrow$$
$$normsub_{\mathbb{F}}(0; e'\_any; rshift(put(f\_any), e'\_any -_{\mathbb{Z}} e\_any) -_{\mathbb{Z}} put(f'\_any)),$$
$$\text{se } (e\_any \neq e\_min) \text{ e } (e\_any \neq e\_max) \text{ e } (e'\_any \neq e\_min) \text{ e}$$
$$(e'\_any \neq e\_max) \text{ e } (e\_any < e'\_any)$$

$$/^* \text{ Núm Pos Norm - Núm Pos Desnorm } e = e\_esp \ ^*/$$
$$0; e\_any; f\_any -_{\mathbb{F}} 0; e\_min; f'\_any \rightarrow$$
$$normsub_{\mathbb{F}}(0; e\_any; put(f\_any) -_{\mathbb{Z}} f'\_any),$$
$$\text{se } (e\_any \neq e\_min) \text{ e } (e\_any \neq e\_max) \text{ e } (f'\_any \neq f\_min) \text{ e}$$
$$(e\_any = e\_esp)$$

Nota: Também como na adição, se e_any é o expoente especial e_esp e o outro expoente é e_min, o sistema não deve tentar igualar os expoentes, pois eles são semanticamente os mesmos. A diferença está no bit implícito: 1, no caso de e_esp; e 0, no caso de e_min.

$$\text{/* Núm Pos Norm - Núm Pos Desnorm } e > e\_esp \text{ */}$$
$$0; e\_any; f\_any -_{\mathbb{F}} 0; e\_min; f'\_any \rightarrow$$
$$normsub_{\mathbb{F}}(0; e\_any; put(f\_any) -_{\mathbb{Z}} rshift(f'\_any, e\_any -_{\mathbb{Z}} e\_esp)),$$
$$\text{se } (e\_any \neq e\_min) \text{ e } (e\_any \neq e\_max) \text{ e } (f'\_any \neq f\_min) \text{ e}$$
$$(e\_any > e\_esp)$$

Nota: Aqui, o sistema deve igualar os expoentes, mas a diferença entre os expoentes não é dada por $e\_any -_{\mathbb{Z}} e\_min$, mas sim por $e\_any -_{\mathbb{Z}} e\_esp$, considerando que o bit implícito de f'_any é 0, e não 1.

$$\text{/* Núm Pos Desnorm - Núm Pos Norm */}$$
$$0; e\_min; f\_any -_{\mathbb{F}} 0; e'\_any; f'\_any \rightarrow$$
$$symm(0; e'\_any; f'\_any -_{\mathbb{F}} 0; e\_min; f\_any),$$
$$\text{se } (f\_any \neq f\_min) \text{ e } (e'\_any \neq e\_min) \text{ e } (e'\_any \neq e\_max)$$

Nota: Esta regra fornece contra-partida para as duas anteriores.

$$\text{/* Núm Pos Desnorm - Núm Pos Desnorm */}$$
$$0; e\_min; f\_any -_{\mathbb{F}} 0; e\_min; f'\_any \rightarrow$$
$$normsub_{\mathbb{F}}(0; e\_min; f\_any -_{\mathbb{Z}} f'\_any),$$
$$\text{se } (f\_any \neq f\_min) \text{ e } (f'\_any \neq f\_min)$$

Nota: Quando ambos os pontos-flutuantes são desnormalizados, basta subtrair as frações.

Novamente, a fim de evitar regras redundantes, a subtração envolvendo operandos com sinais diferentes é mapeada na adição correspondente:

$$\text{/* Positivo - Negativo */}$$
$$0; e\_any; f\_any -_{\mathbb{F}} 1; e'\_any; f'\_any \rightarrow 0; e\_any; f\_any +_{\mathbb{F}} 0; e'\_any; f'\_any$$

$$\text{/* Negativo - Positivo */}$$
$$1; e\_any; f\_any -_{\mathbb{F}} 0; e'\_any; f'\_any \rightarrow$$
$$symm(0; e'\_any; f'\_any +_{\mathbb{F}} 0; e\_any; f\_any)$$

A diferença de dois operandos negativos quaisquer, por sua vez, é mapeada na diferença dos respectivos simétricos (positivos):

$$\text{/* Negativo - Negativo */}$$
$$1; e\_any; f\_any -_{\mathbb{F}} 1; e'\_any; f'\_any \rightarrow 0; e'\_any; f'\_any -_{\mathbb{F}} 0; e\_any; f\_any$$

6.2.4 Multiplicação

6.2.4.1 Dígitos

$$
\begin{gathered}
\text{/* Multiplicação de dígitos */} \\
0 \times_{\mathbb{D}} 0 \rightarrow END \\
0 \times_{\mathbb{D}} 1 \rightarrow END \\
1 \times_{\mathbb{D}} 0 \rightarrow END \\
1 \times_{\mathbb{D}} 1 \rightarrow END : 1
\end{gathered}
$$

$$\|t \times_{\mathbb{D}} t'\| = \|t\| \times \|t'\|$$

6.2.4.2 Inteiros

$$
\begin{gathered}
\text{/* Multiplicação por 2 */} \\
timestwo(END) \rightarrow END \\
timestwo(x : y) \rightarrow (x : y) : 0 \\
timestwo(MINUS(x)) \rightarrow MINUS(timestwo(x))
\end{gathered}
$$

$$\|timestwo(t)\| = 2 \times \|t\|$$

$$
\begin{gathered}
\text{/* Multiplicação de inteiros */} \\
END \times_{\mathbb{Z}} END \rightarrow END \\
END \times_{\mathbb{Z}} (x' : y') \rightarrow END \\
END \times_{\mathbb{Z}} MINUS(y') \rightarrow END \\
(x : y) \times_{\mathbb{Z}} END \rightarrow END
\end{gathered}
$$

$$
\begin{gathered}
(x : y) \times_{\mathbb{Z}} (x' : y') \rightarrow (y \times_{\mathbb{D}} y') +_{\mathbb{Z}} \\
timestwo(((END : y) \times_{\mathbb{Z}} x') + (x \times_{\mathbb{Z}} (END : y'))) +_{\mathbb{Z}} \\
timestwo(timestwo(x \times_{\mathbb{Z}} x'))
\end{gathered}
$$

$$
\begin{gathered}
(x : y) \times_{\mathbb{Z}} MINUS(y') \rightarrow MINUS((x : y) \times_{\mathbb{Z}} y') \\
MINUS(x) \times_{\mathbb{Z}} END \rightarrow END \\
MINUS(x) \times_{\mathbb{Z}} (x' : y') \rightarrow MINUS(x \times_{\mathbb{Z}} (x' : y')) \\
MINUS(x) \times_{\mathbb{Z}} MINUS(y') \rightarrow x \times_{\mathbb{Z}} y'
\end{gathered}
$$

$$\|t \times_{\mathbb{Z}} t'\| = \|t\| \times \|t'\|$$

6.2.4.3 Pontos-flutuantes

A multiplicação é a mais simples das operações. Basicamente, a máquina deve multiplicar os significantes, somar os expoentes e normalizar o resultado.

A regra a seguir descreve uma função auxiliar para normalização de fração:

$$/^* \; lpad \text{ completa com 0's o lado esquerdo de uma lista,}$$
$$\text{para que a mesma fique do tamanho de } f\_size \; ^*/$$
$$lpad(x) \rightarrow x +_{\mathbb{Z}} f\_min$$

Nota: O artifício de somar x e f_min é possível, porque a adição de listas preserva os 0's à esquerda.

A função $normmult_{\mathbb{F}}$ é definida em apenas três casos:

$$/^* \text{ ...em caso de expoente normalizado } ^*/$$
$$normmult_{\mathbb{F}}(s; x; y) \rightarrow s; x +_{\mathbb{Z}} length(y);$$
$$rshift(y, length(y) -_{\mathbb{Z}} f\_size),$$
$$\text{se } ((x +_{\mathbb{Z}} length(y)) \geq e\_esp) \text{ e } ((x +_{\mathbb{Z}} length(y)) \leq e\_max)$$

Nota: A operação de multiplicação ($\times_{\mathbb{F}}$) concatena o bit implícito e a fração de cada fator, e efetua o produto das listas (inteiros), como se o ponto implícito fosse deslocado totalmente para a direita da parte fracionária do significante. Aqui, o ponto implícito é deslocado de volta para o lado esquerdo da fração. Para isso, o expoente deve ser aumentado de $length(y)$. A diferença $length(y) -_{\mathbb{Z}} f\_size$ indica o estouro de precisão, e a função $rshift$ é aplicada para formatar a fração.

$$/^* \text{ ...em caso de precisar desnormalizar o expoente } ^*/$$
$$normmult_{\mathbb{F}}(s; x; y) \rightarrow s; e\_min;$$
$$lpad(rshift(y, (length(y) -_{\mathbb{Z}} f\_size) +_{\mathbb{Z}} (e\_esp -_{\mathbb{Z}} x))),$$
$$\text{se } ((x +_{\mathbb{Z}} length(y)) < e\_esp) \text{ e } ((x +_{\mathbb{Z}} length(y)) \leq e\_max)$$

Nota: Se a soma $x +_{\mathbb{Z}} length(y)$ é menor do que e_esp, significa que o resultado não pode ser expresso como um ponto-flutuante normalizado. Desta forma, $rshift$ será aplicada na fração não apenas para remover os dígitos além da precisão ($length(y) -_{\mathbb{Z}} f\_size$), mas também para desnormalizar o expoente ($e\_esp -_{\mathbb{Z}} x$). A função *lpad* assegura que, no final, a fração não terá dígitos aquém da precisão.

$$/^* \text{ ...em caso de } overflow\text{: gera QNaN } ^*/$$
$$normmult_{\mathbb{F}}(s; x; y) \rightarrow s; e\_max; f\_max,$$
$$\text{se } ((x +_{\mathbb{Z}} length(y)) > e\_max)$$

Nota: $s; e\_max; f\_max$ é um NaN reservado do SRTF.

Seguem agora as regras para multiplicação de pontos-flutuantes, começando pelas triviais:

$$/^* \text{ Zero Pos} \times \text{Núm Pos} = \text{Zero Pos } ^*/$$
$$0; e\_min; f\_min \times_{\mathbb{F}} 0; e'\_any; f'\_any \rightarrow 0; e\_min; f\_min,$$
$$\text{se } (e'\_any \neq e\_min)$$

$$/^* \text{ Zero Pos} \times \text{Infinito Pos} = \text{Zero Pos } ^*/$$
$$0; e\_min; f\_min \times_{\mathbb{F}} 0; e\_max; f\_min) \rightarrow 0; e\_min; f\_min$$

$$/^* \text{ Zero Pos} \times \text{NaN Pos} = \text{NaN Pos } ^*/$$
$$0; e\_min; f\_min \times_{\mathbb{F}} 0; e\_max; f'\_any \rightarrow 0; e\_max; f'\_any,$$
$$\text{se } (f'\_any \neq f\_min)$$

$$/^* \text{ NaN Pos} \times \text{Qualquer Pos} = \text{NaN Pos } ^*/$$
$$0; e\_max; f\_any \times_{\mathbb{F}} 0; e'\_any; f'\_any \rightarrow 0; e\_max; f\_any,$$
$$\text{se } (f\_any \neq f\_min)$$

$$/^* \text{ Infinito Pos} \times \text{Zero Pos} = \text{Zero Pos } ^*/$$
$$0; e\_max; f\_min \times_{\mathbb{F}} 0; e\_min; f\_min \rightarrow 0; e\_min; f\_min$$

$$/^* \text{ Infinito Pos} \times \text{Núm Pos} = \text{Infinito Pos } ^*/$$
$$0; e\_max; f\_min \times_{\mathbb{F}} 0; e'\_any; f'\_any \rightarrow 0; e\_max; f\_min,$$
$$\text{se } (e'\_any \neq e\_max) \text{ e } (f'\_any \neq f\_min)$$

$$/^* \text{ Infinito Pos} \times \text{Infinito Pos} = \text{Infinito Pos } ^*/$$
$$0; e\_max; f\_min \times_{\mathbb{F}} 0; e\_max; f\_min) \rightarrow 0; e\_max; f\_min$$

$$/^* \text{ Infinito Pos} \times \text{NaN Pos} = \text{NaN Pos}^*/$$
$$0; e\_max; f\_min \times_{\mathbb{F}} 0; e\_max; f'\_any \rightarrow 0; e\_max; f'\_any,$$
$$\text{se } (f'\_any \neq f\_min)$$

$$/^* \text{ Núm Pos} \times \text{Zero Pos} = \text{Zero Pos } ^*/$$
$$0; e\_any; f\_any \times_{\mathbb{F}} 0; e\_min; f\_min \rightarrow 0; e\_min; f\_min,$$
$$\text{se } (e\_any \neq e\_max)$$

$$/^* \text{ Núm Pos} \times \text{Infinito Pos} = \text{Infinito Pos } ^*/$$
$$0; e\_any; f\_any \times_{\mathbb{F}} 0; e\_max; f\_min \rightarrow 0; e\_max; f\_min,$$
$$\text{se } (e\_any \neq e\_max)$$

$$/^* \text{ Núm Pos} \times \text{NaN Pos} = \text{NaN Pos } ^*/$$
$$0; e\_any; f\_any \times_{\mathbb{F}} 0; e\_max; f'\_any \rightarrow 0; e\_max; f'\_any,$$
$$\text{se } (e\_any \neq e\_max) \text{ e } (f'\_any \neq f\_min)$$

Casos não-triviais da multiplicação de pontos-flutuantes:

$$/^* \text{ Núm Pos Norm} \times \text{Núm Pos Norm: } ^*/$$
$$0; e\_any; f\_any \times_{\mathbb{F}} 0; e'\_any; f'\_any \rightarrow$$
$$normmult_{\mathbb{F}}(0; (e\_any -_{\mathbb{Z}} f\_size) +_{\mathbb{Z}} (e'\_any -_{\mathbb{Z}} f\_size);$$
$$cut(put(f\_any) \times_{\mathbb{Z}} put(f'\_any))),$$
$$\text{se } (e\_any \neq e\_min) \text{ e } (e\_any \neq e\_max) \text{ e } (e'\_any \neq e\_min) \text{ e}$$
$$(e'\_any \neq e\_max)$$

Nota: Conforme foi mencionado anteriormente, o bit implícito e a fração de cada fator são concatenados, e efetua-se o produto das listas (inteiros), como se o ponto implícito fosse deslocado totalmente para a direita da parte fracionária do significante (daí diminuir-se f_size de cada expoente). Em seguida, o bit implícito é removido do produto.

$$\begin{gathered}
/^* \text{ Núm Pos Norm} \times \text{Núm Pos Desnorm: } */ \\
0; e_any; f_any \times_{\mathbb{F}} 0; e_min; f'_any \rightarrow \\
normmult_{\mathbb{F}}(0; (e_any -_{\mathbb{Z}} f_size) +_{\mathbb{Z}} (e_esp -_{\mathbb{Z}} f_size); put(f_any) \times_{\mathbb{Z}} f'_any), \\
\text{se } (e_any \neq e_min) \text{ e } (e_any \neq e_max) \text{ e } (f'_any \neq f_min)
\end{gathered}$$

Nota: Quando um dos fatores é um ponto-flutuante desnormalizado, o bit implícito (no caso, 0) não precisa ser concatenado à fração, e assume-se que o expoente é e_esp, já que e_min, como já foi comentado, é uma mera convenção sintática. Isto ocorre, de forma análoga, nas duas regras a seguir.

$$\begin{gathered}
/^* \text{ Núm Pos Desnorm} \times \text{Núm Pos Norm: } */ \\
0; e_min; f_any \times_{\mathbb{F}} 0; e'_any; f'_any \rightarrow \\
normmult_{\mathbb{F}}(0; (e_esp -_{\mathbb{Z}} f_size) +_{\mathbb{Z}} (e'_any -_{\mathbb{Z}} f_size); f_any \times_{\mathbb{Z}} put(f'_any)), \\
\text{se } (f_any \neq f_min) \text{ e } (e'_any \neq e_min) \text{ e } (e'_any \neq e_max)
\end{gathered}$$

$$\begin{gathered}
/^* \text{ Núm Pos Desnorm} \times \text{Núm Pos Desnorm: } */ \\
0; e_min; f_any \times_{\mathbb{F}} 0; e_min; f'_any \rightarrow \\
normmult_{\mathbb{F}}(0; (e_esp -_{\mathbb{Z}} f_size) +_{\mathbb{Z}} (e_esp -_{\mathbb{Z}} f_size); f_any \times_{\mathbb{Z}} f'_any), \\
\text{se } (f_any \neq f_min) \text{ e } (f'_any \neq f_min)
\end{gathered}$$

A multiplicação envolvendo um ou mais operandos negativos é mapeada na multiplicação dos respectivos módulos dos operandos e a função *symm* é usada para ajustar o sinal do produto, se necessário.

$$\begin{gathered}
/^* \text{ Positivo} \times \text{Negativo } */ \\
0; e_any; f_any \times_{\mathbb{F}} 1; e'_any; f'_any \rightarrow \\
symm(0; e_any; f_any \times_{\mathbb{F}} 0; e'_any; f'_any)
\end{gathered}$$

$$\begin{gathered}
/^* \text{ Negativo} \times \text{Positivo } */ \\
1; e_any; f_any \times_{\mathbb{F}} 0; e'_any; f'_any \rightarrow \\
symm(0; e_any; f_any \times_{\mathbb{F}} 0; e'_any; f'_any)
\end{gathered}$$

$$\begin{gathered}
/^* \text{ Negativo} \times \text{Negativo } */ \\
1; e_any; f_any \times_{\mathbb{F}} 1; e'_any; f'_any \rightarrow \\
0; e_any; f_any \times_{\mathbb{F}} 0; e'_any; f'_any)
\end{gathered}$$

6.2.5 Função de comparação

É possível definir-se uma certa ordem "$\leq$" sobre o conjunto $\mathbb{F}$. O próprio Padrão IEEE 754 prevê a implementação de uma função de comparação entre os pontos-flutuantes (veja [IEE85] pg 8). Esta função induz, informalmente, a seguinte ordenação entre os pontos-flutuantes:

$$-\infty \leq \text{reais negativos} \leq -0 = +0 \leq \text{reais positivos} \leq +\infty,$$

onde "reais negativos" e "reais positivos" devem ser entendidos como os pontos-flutuantes cujos valores numéricos são reais negativos e reais positivos, respectivamente, em ordem crescente.

Nota: O Padrão IEEE 754 estabelece que os NaN's não são comparáveis, ou seja, apenas os elementos de $\mathbb{F}$ são relacionados segundo "$\leq$".

A função *cmp*, fornecida nesta seção, expressa a ordem entre pontos-flutuantes informalmente descrita acima, e será usada posteriormente para identificar o mínimo de dois pontos-flutuantes dados, na construção de intervalos Padrão IEEE 754. Uma restrição se faz necessária, em relação ao padrão: O par $(-0, +0)$ é removido da relação $\leq$, em conformidade com a Definição 5.3.2, resultando na relação

$$-\infty \leq \text{reais negativos} \leq +0 \leq -0 \leq \text{reais positivos} \leq +\infty.$$

Seguem as regras da função *cmp*, a qual recebe dois pontos-flutuantes f e f' como parâmetros e retorna:

- 0, se $f \leq f'$;
- 1, se $f \geq f'$;
- 2, se $f \bowtie f'$, i.e. f e f' são incomparáveis.

$$\text{/* Zero Pos} \leq \text{Zero */}$$
$$cmp(0; e\_min; f\_min, s'; e\_min; f\_min) \rightarrow 0$$

$$\text{/* Zero Neg} \geq \text{Zero */}$$
$$cmp(1; e\_min; f\_min, s'; e\_min; f\_min) \rightarrow 1$$

$$\text{/* Zero} \leq \text{Infinito Pos */}$$
$$cmp(s; e\_min; f\_min, 0; e\_max; f\_min) \rightarrow 0$$

/* Zero $\geq$ Infinito Neg */
$cmp(s; e\_min; f\_min, 1; e\_max; f\_min) \rightarrow 1$

/* Zero $\leq$ Não-Zero Pos */
$cmp(s; e\_min; f\_min, 0; e'\_any; f'\_any) \rightarrow 0,$
se $(e'\_any \neq e\_max)$ e $((e'\_any \neq e\_min)$ ou $(f'\_any \neq f\_min))$

/* Zero $\geq$ Não-Zero Neg */
$cmp(s; e\_min; f\_min, 1; e'\_any; f'\_any) \rightarrow 1,$
se $(e'\_any \neq e\_max)$ e $((e'\_any \neq e\_min)$ ou $(f'\_any \neq f\_min))$

/* Zero $\bowtie$ NaN */
$cmp(s; e\_min; f\_min, s'; e\_max; f'\_any) \rightarrow 2,$
se $(f'\_any \neq f\_min)$

/* NaN $\bowtie$ Qualquer */
$cmp(s; e\_max; f\_any, s'; e'\_any; f'\_any) \rightarrow 2,$
se $(f\_any \neq f\_min)$

/* Infinito Pos $\geq$ Não-NaN */
$cmp(0; e\_max; f\_min, s'; e'\_any; f'\_any) \rightarrow 1,$
se $(e'\_any \neq e\_max)$ ou $(f'\_any = f\_min)$

/* Infinito Pos $\bowtie$ NaN */
$cmp(0; e\_max; f\_min, s'; e\_max; f'\_any) \rightarrow 2,$
se $(f'\_any \neq f\_min)$

/* Infinito Neg $\leq$ Não-NaN */
$cmp(1; e\_max; f\_min, s'; e'\_any; f'\_any) \rightarrow 0,$
se $(e'\_any \neq e\_max)$ ou $(f'\_any = f\_min)$

/* Infinito Neg $\bowtie$ NaN */
$cmp(1; e\_max; f\_min, s'; e\_max; f'\_any) \rightarrow 2,$
se $(f'\_any \neq f\_min)$

/* Não-Zero Neg $\leq$ Zero */
$cmp(1; e\_any; f\_any, s'; e\_min; f\_min) \rightarrow 0,$
se $(e\_any \neq e\_max)$ e $((e\_any \neq e\_min)$ ou $(f\_any \neq f\_min))$

/* Não-Zero Pos $\geq$ Zero */
$cmp(0; e\_any; f\_any, s'; e\_min; f\_min) \rightarrow 1,$
se $(e\_any \neq e\_max)$ e $((e\_any \neq e\_min)$ ou $(f\_any \neq f\_min))$

/* Não-Zero $\leq$ Infinito Pos */
$cmp(s; e\_any; f\_any, 0; e\_max; f\_min) \rightarrow 0,$
se $(e\_any \neq e\_max)$ e $((e\_any \neq e\_min)$ ou $(f\_any \neq f\_min))$

/* Não-Zero $\geq$ Infinito Neg */
$cmp(s; e\_any; f\_any, 1; e\_max; f\_min) \rightarrow 1,$
se $(e\_any \neq e\_max)$ e $((e\_any \neq e\_min)$ ou $(f\_any \neq f\_min))$

/* Não-Zero $\bowtie$ NaN */
$cmp(s; e\_any; f\_any, s'; e\_max; f'\_any) \rightarrow 2,$
se $(e\_any \neq e\_max)$ e $((e\_any \neq e\_min)$ ou $(f\_any \neq f\_min))$ e
$(f'\_any \neq f\_min)$

/* Não-Zero Neg $\leq$ Não-Zero Pos */
$cmp(1; e\_any; f\_any, 0; e'\_any; f'\_any) \rightarrow 0,$
se $(e\_any \neq e\_max)$ e $((e\_any \neq e\_min)$ ou $(f\_any \neq f\_min))$ e
$(e'\_any \neq e\_max)$ e $((e'\_any \neq e\_min)$ ou $(f'\_any \neq f\_min))$

/* Não-Zero Pos $\geq$ Não-Zero Neg */
$cmp(0; e\_any; f\_any, 1; e'\_any; f'\_any) \rightarrow 1,$
se $(e\_any \neq e\_max)$ e $((e\_any \neq e\_min)$ ou $(f\_any \neq f\_min))$ e
$(e'\_any \neq e\_max)$ e $((e'\_any \neq e\_min)$ ou $(f'\_any \neq f\_min))$

/* Não-Zero Pos com Não-Zero Pos: Compara $f - f'$ com $f' - f$ */
$cmp(0; e\_any; f\_any, 0; e'\_any; f'\_any) \rightarrow$
$cmp(0; e\_any; f\_any -_{\mathbb{F}} 0; e'\_any; f'\_any,$
$0; e'\_any; f'\_any -_{\mathbb{F}} 0; e\_any; f\_any),$
se $(e\_any \neq e\_max)$ e $((e\_any \neq e\_min)$ ou $(f\_any \neq f\_min))$ e
$(e'\_any \neq e\_max)$ e $((e'\_any \neq e\_min)$ ou $(f'\_any \neq f\_min))$

/* Não-Zero Neg com Não-Zero Neg: Compara os simétricos */
$cmp(1; e\_any; f\_any, 1; e'\_any; f'\_any) \rightarrow$
$cmp(0; e'\_any; f'\_any),\ 0; e\_any; f\_any)$
se $(e\_any \neq e\_max)$ e $((e\_any \neq e\_min)$ ou $(f\_any \neq f\_min))$ e
$(e'\_any \neq e\_max)$ e $((e'\_any \neq e\_min)$ ou $(f'\_any \neq f\_min))$

6.3 SRTI

Nesta seção, o SRTF será estendido com regras para a aritmética intervalar de Moore, apresentada no Capítulo 5. As equações do sistema resultante, chamado Sistema de Reescrita de Termos para Intervalos (SRTI), expressam as fórmulas do Teorema 5.3.2, descrito na Seção 5.3, com uma restrição: As operações entre

extremos de intervalos sempre aplicam arredondamento para 0, sejam extremos à esquerda ou à direita, em desacordo com o que foi enunciado no referido teorema, devido a uma limitação do sistema subjacente, comentada anteriormente.

O construtor intervalar apresentado a seguir reflete a Definição 5.3.7, para intervalo Padrão IEEE 754 estendido, onde os extremos são representados por pontos-flutuantes Padrão IEEE 754, neste caso, os termos definidos na seção 6.2.1.3. Vale recordar a convenção de que o -0 não pode aparecer como um extremo esquerdo, e o $+0$ não pode aparecer como um extremo direito. A ordem entre os extremos é checada através da função auxiliar "*cmp*", não permitindo a construção de intervalos com extremo esquerdo maior que o direito.

Para maior legibilidade, serão usadas variáveis simples a, b, c, e d para pontos-flutuantes nos extremos de intervalos; além disso, os seus respectivos bits de sinais serão referenciados simplesmente como a_s, b_s, c_s e d_s.

6.3.1 Construtor

Através da função *cmp*, é possível fazer a distinção entre intervalos e INaN's.

$$\begin{gathered}/^* \text{ Intervalo Padrão IEEE 754 } ^*/ \\ \|\langle a, b\rangle\| = \{x \in [-\infty, +\infty] \mid \|a\| \leq x \leq \|b\|\}, \\ \text{se } (cmp(a, b) = 0)\end{gathered}$$

$$\begin{gathered}/^* \text{ NaN Intervalar (INaN) } ^*/ \\ \|\langle a, b\rangle\| = \emptyset, \\ \text{se } (cmp(a, b) = 2)\end{gathered}$$

No que segue, as operações aritméticas de adição, subtração e multiplicação serão apresentadas, e respectivamente denotadas por $+_{\mathbb{IF}}$, $-_{\mathbb{IF}}$ e $\times_{\mathbb{IF}}$. Note-se, entretanto, que apesar desta notação simplificada, tais operações estão definidas sobre o conjunto $\mathbb{IF} \cup \mathbb{INAN}$, definido na Seção 5.3.

6.3.2 Adição

$$\langle a, b\rangle +_{\mathbb{IF}} \langle c, d\rangle \rightarrow \langle a +_{\mathbb{F}} c, b +_{\mathbb{F}} d\rangle$$

6.3.3 Subtração

$$\langle a, b\rangle -_{\mathbb{IF}} \langle c, d\rangle \rightarrow \langle a -_{\mathbb{F}} d, b -_{\mathbb{F}} c\rangle$$

6.3.4 Multiplicação

$$
\begin{gathered}
/^* \text{ Positivo} \times \text{Positivo } ^*/ \\
\langle a, b\rangle \times_{\mathbb{IF}} \langle c, d\rangle \rightarrow \langle a \times_{\mathbb{F}} c, b \times_{\mathbb{F}} d\rangle, \\
\text{se } (a_s = 0) \text{ e } (b_s = 0) \text{ e } (c_s = 0) \text{ e } (d_s = 0)
\end{gathered}
$$

$$
\begin{gathered}
/^* \text{ Misto} \times \text{Positivo } ^*/ \\
\langle a, b\rangle \times_{\mathbb{IF}} \langle c, d\rangle \rightarrow \langle a \times_{\mathbb{F}} d, b \times_{\mathbb{F}} d\rangle, \\
\text{se } (a_s = 1) \text{ e } (b_s = 0) \text{ e } (c_s = 0) \text{ e } (d_s = 0)
\end{gathered}
$$

$$
\begin{gathered}
/^* \text{ Negativo} \times \text{Positivo } ^*/ \\
\langle a, b\rangle \times_{\mathbb{IF}} \langle c, d\rangle \rightarrow \langle a \times_{\mathbb{F}} d, b \times_{\mathbb{F}} c\rangle, \\
\text{se } (a_s = 1) \text{ e } (b_s = 1) \text{ e } (c_s = 0) \text{ e } (d_s = 0)
\end{gathered}
$$

$$
\begin{gathered}
/^* \text{ Positivo} \times \text{Misto } ^*/ \\
\langle a, b\rangle \times_{\mathbb{IF}} \langle c, d\rangle \rightarrow \langle b \times_{\mathbb{F}} c, b \times_{\mathbb{F}} d\rangle, \\
\text{se } (a_s = 0) \text{ e } (b_s = 0) \text{ e } (c_s = 1) \text{ e } (d_s = 0)
\end{gathered}
$$

Na multiplicação de dois intervalos mistos $\langle a, b\rangle$ e $\langle c, d\rangle$, faz-se necessário encontrar $min(a \times d, b \times c)$ e $max(b \times d, a \times c)$, conforme a Figura 5.3. Como as funções *min* e *max* não estão definidas no SRTF, a mesma operação será feita, por casos, através da função *cmp*:

$$
\begin{gathered}
/^* \text{ Misto} \times \text{Misto: } a \times d \leq b \times c,\ b \times d \geq a \times c \ ^*/ \\
\langle a, b\rangle \times_{\mathbb{IF}} \langle c, d\rangle \rightarrow \langle a \times_{\mathbb{F}} d, b \times_{\mathbb{F}} d\rangle, \\
\text{se } (a_s = 1) \text{ e } (b_s = 0) \text{ e } (c_s = 1) \text{ e } (d_s = 0) \text{ e} \\
(cmp(a \times_{\mathbb{F}} d, b \times_{\mathbb{F}} c) = 0) \text{ e } (cmp(b \times_{\mathbb{F}} d, a \times_{\mathbb{F}} c) = 1)
\end{gathered}
$$

$$
\begin{gathered}
/^* \text{ Misto} \times \text{Misto: } a \times d \geq b \times c,\ b \times d \geq a \times c \ ^*/ \\
\langle a, b\rangle \times_{\mathbb{IF}} \langle c, d\rangle \rightarrow \langle b \times_{\mathbb{F}} c, b \times_{\mathbb{F}} d\rangle, \\
\text{se } (a_s = 1) \text{ e } (b_s = 0) \text{ e } (c_s = 1) \text{ e } (d_s = 0) \text{ e} \\
(cmp(a \times_{\mathbb{F}} d, b \times_{\mathbb{F}} c) = 1) \text{ e } (cmp(b \times_{\mathbb{F}} d, a \times_{\mathbb{F}} c) = 1)
\end{gathered}
$$

$$
\begin{gathered}
/^* \text{ Misto} \times \text{Misto: } a \times d \leq b \times c,\ b \times d \leq a \times c \ ^*/ \\
\langle a, b\rangle \times_{\mathbb{IF}} \langle c, d\rangle \rightarrow \langle a \times_{\mathbb{F}} d, a \times_{\mathbb{F}} c\rangle, \\
\text{se } (a_s = 1) \text{ e } (b_s = 0) \text{ e } (c_s = 1) \text{ e } (d_s = 0) \text{ e} \\
(cmp(a \times_{\mathbb{F}} d, b \times_{\mathbb{F}} c) = 0) \text{ e } (cmp(b \times_{\mathbb{F}} d, a \times_{\mathbb{F}} c) = 0)
\end{gathered}
$$

$$
\begin{gathered}
/^* \text{ Misto} \times \text{Misto: } a \times d \geq b \times c,\ b \times d \leq a \times c \ ^*/ \\
\langle a, b\rangle \times_{\mathbb{IF}} \langle c, d\rangle \rightarrow \langle b \times_{\mathbb{F}} c, a \times_{\mathbb{F}} c\rangle, \\
\text{se } (a_s = 1) \text{ e } (b_s = 0) \text{ e } (c_s = 1) \text{ e } (d_s = 0) \text{ e} \\
(cmp(a \times_{\mathbb{F}} d, b \times_{\mathbb{F}} c) = 1) \text{ e } (cmp(b \times_{\mathbb{F}} d, a \times_{\mathbb{F}} c) = 0)
\end{gathered}
$$

Nas regras do SRTI apresentadas até aqui, a possível ocorrência de um INaN ficou implícita, pois, em geral, quando operações entre extremos ocasionam um NaN, este mesmo NaN será propagado em todas as operações subseqüentes. A próxima regra, entretanto, requer a sinalização explícita de um INaN (reservado do SRTI), para o caso em que a função *cmp* tenha recebido NaN como um de seus parâmetros:

$$\begin{gathered}
/^* \text{ Misto } \times \text{ Misto: INaN } ^*/ \\
\langle a, b\rangle \times_{\mathbb{IF}} \langle c, d\rangle \rightarrow \langle 0; e_max; f_max, 0; e_max; f_max\rangle, \\
\text{se } (a_s = 1) \text{ e } (b_s = 0) \text{ e } (c_s = 1) \text{ e } (d_s = 0) \text{ e} \\
((cmp(a \times_{\mathbb{F}} d, b \times_{\mathbb{F}} c) = 2) \text{ ou } (cmp(b \times_{\mathbb{F}} d, a \times_{\mathbb{F}} c) = 2))
\end{gathered}$$

$$\begin{gathered}
/^* \text{ Negativo } \times \text{ Misto } ^*/ \\
\langle a, b\rangle \times_{\mathbb{IF}} \langle c, d\rangle \rightarrow \langle a \times_{\mathbb{F}} d, a \times_{\mathbb{F}} c\rangle, \\
\text{se } (a_s = 1) \text{ e } (b_s = 1) \text{ e } (c_s = 1) \text{ e } (d_s = 0)
\end{gathered}$$

$$\begin{gathered}
/^* \text{ Positivo } \times \text{ Negativo } ^*/ \\
\langle a, b\rangle \times_{\mathbb{IF}} \langle c, d\rangle \rightarrow \langle b \times_{\mathbb{F}} c, a \times_{\mathbb{F}} d\rangle, \\
\text{se } (a_s = 0) \text{ e } (b_s = 0) \text{ e } (c_s = 1) \text{ e } (d_s = 1)
\end{gathered}$$

$$\begin{gathered}
/^* \text{ Misto } \times \text{ Negativo } ^*/ \\
\langle a, b\rangle \times_{\mathbb{IF}} \langle c, d\rangle \rightarrow \langle b \times_{\mathbb{F}} c, a \times_{\mathbb{F}} c\rangle, \\
\text{se } (a_s = 1) \text{ e } (b_s = 0) \text{ e } (c_s = 1) \text{ e } (d_s = 1)
\end{gathered}$$

$$\begin{gathered}
/^* \text{ Negativo } \times \text{ Negativo } ^*/ \\
\langle a, b\rangle \times_{\mathbb{IF}} \langle c, d\rangle \rightarrow \langle b \times_{\mathbb{F}} d, a \times_{\mathbb{F}} c\rangle, \\
\text{se } (a_s = 1) \text{ e } (b_s = 1) \text{ e } (c_s = 1) \text{ e } (d_s = 1)
\end{gathered}$$

$$\begin{gathered}
/^* \text{ Zero } \times \text{ Qualquer } ^*/ \\
\langle a, b\rangle \times_{\mathbb{IF}} \langle c, d\rangle \rightarrow \langle 0; e_min; f_min, 1; e_min; f_min\rangle, \\
\text{se } (a_s = 0) \text{ e } (b_s = 1)
\end{gathered}$$

$$\begin{gathered}
/^* \text{ Qualquer } \times \text{ Zero } ^*/ \\
\langle a, b\rangle \times_{\mathbb{IF}} \langle c, d\rangle \rightarrow \langle 0; e_min; f_min, 1; e_min; f_min\rangle, \\
\text{se } (c_s = 0) \text{ e } (d_s = 1)
\end{gathered}$$

Neste capítulo, foram apresentados dois sistemas de reescrita de termos condicionais para aritmética (adição, subtração e multiplicação): O primeiro deles, chamado SRTF, é baseado no Padrão IEEE 754, e foi proposto como modelo formal para a computação de pontos-flutuantes. Apenas os requisitos essenciais do padrão em questão foram implementados. O outro sistema apresentado foi o SRTI, que, por sua vez, visa delinear a base para a formulação de um modelo formal para a computação intervalar, sobre intervalos Padrão IEEE 754. O escopo das contribuições desta dissertação, bem como as propriedades dos sistemas propostos e trabalhos correlatos serão discutidas no capítulo a seguir.

Capítulo 7

Conclusão

7.1 Considerações finais

O crescente investimento em pesquisas na área de métodos formais, tanto por parte da indústria como por parte das instituições de ensino, conforme foi colocado no Capítulo 1, tem levado a esforços contínuos na direção da criação e do aperfeiçoamento de metodologias e ferramentas de forma que os benefícios do seu uso possam ser, de fato, disseminados. Entretanto, concordando com Bowen [BH95], a falta de uma ligação mais forte entre a indústria e a academia ainda constitui um obstáculo a ser enfrentado. Uma iniciativa neste sentido, por exemplo, vem do grupo CoFI [CoF04], que propôs a linguagem CASL para especificações algébricas, aplicando padronização e simplificação notacionais.

O enfoque da pesquisa aqui apresentada está na construção de um sistema de reescrita de termos que possa ser usado para verificação formal (e automática) de especificações de aplicações que envolvam números reais, contribuindo, desta forma, para a solução do problema colocado acima, já que é praticamente impossível pensar na aplicação real de métodos formais sem abranger sistemas desta natureza.

É digno de nota que, apesar da crucial necessidade (intrínseca à maioria das aplicações do mundo real) de representação e manipulação algébrica de números reais, exemplos canônicos de linguagens de especificação formal tenham tão pouco (ou nada) a dizer sobre isto, ao menos em suas implementações originais. É o caso, por exemplo, das linguagens Z [PS96, Bow96], B [Rob00, Wor96], orientadas a modelo, e as linguagens algébricas OBJ3 [Gog04, GM96] e CASL [CoF04, Mos01].

7.2 Trabalhos correlatos

Muitas soluções já foram e continuam sendo propostas, e algumas, de fato, implementadas com relativo sucesso. Uma abordagem bastante popular tem sido a criação de teorias para computação de números reais exatos, implementando aritmética de ponto-flutuante clássica [Esc97, Har96, MR99, RSM99]; esta abordagem tem se mostrado bastante aplicável no contexto de linguagens funcionais, como é o caso, por exemplo, de PCF [Esc97], mas parece estar bem distante ainda de prover um modelo para especificação e verificação formal de sistemas. Outro aspecto negativo desta abordagem é que ela herda os problemas intrínsecos da aritmética de ponto-flutuante comentados no capítulo introdutório.

Outra abordagem que merece destaque é a abordagem intervalar [San99, Lop04], à qual constitui uma interessante alternativa para especificação e verificação formal de sistemas que envolvam números reais. Um problema crucial, aqui, reside no tratamento da igualdade (e, portanto, de equações), devido à ausência de inversos aditivo e multiplicativo, como mencionado na Seção 5.2.2. A teoria das equações intervalares locais proposta por Santiago [San99] foi uma importante contribuição neste sentido, mas possui algumas dificuldades com relação à lógica equacional resultante, pois ela generaliza a noção de igualdade que atualmente é a base para a verificação equacional, não tendo sido descoberto, ainda, um princípio de substituição de iguais por iguais adequado para esta teoria; veja Lopes [Lop04], que implementou um tipo de dados Intervalos para a linguagem CASL.

Espera-se, com este trabalho, contribuir para o desenvolvimento de um modelo que possa prover os benefícios da representação de números reais através de intervalos, sem que, contudo, seja necessário aplicar a nova noção de igualdade referida acima, ou seja, sem perder a simplicidade da aritmética intervalar clássica. Neste sentido, o Sistema de Reescrita de Termos para Intervalos (SRTI) está sendo proposto como a base para a construção de uma semântica operacional para a computação intervalar.

Por fim, importantes trabalhos correlatos se destacam na área de sistemas de reescrita de termos para aritmética: Cohen e Watson [CW91] apresentaram um sistema para adição e multiplicação de inteiros (em base 4), cuja terminação permanece como problema aberto; Walters e Zantema [WZ95] fornecem um sistema para adição, subtração e multiplicação de inteiros, confluente apenas para termos fechados, assim como o sistema apresentado por de Vries e Yamada [dVY94], para números reais (na verdade, números racionais com expansões decimais finitas); e, em especial, destaca-se o sistema de Kennaway [Ken95], para adição, subtração e multiplicação de números inteiros.

O sistema de Kennaway [Ken95] foi usado como base para as regras de aritmética binária subjacente ao SRTF, proposto na Seção 6.2. Dentre os sistemas mencionados, este é o único ortogonal. Sua terminação foi provada através de um mapeamento de termos a um conjunto bem ordenado, baseado no padrão de ocorrências de símbolos de função nos lados direitos das regras.

7.3 Trabalhos futuros

Alguns pontos sejam levantados, em relação às propriedades do sistema proposto:

1) O SRTF é correto no sentido de que todas as regras de reescrita expressam verdades sobre pontos-flutuantes Padrão IEEE 754, e todas as expressões se reduzem para pontos-flutuantes, embora o SRTF não seja, evidentemente, completo, já que vários requisitos do Padrão IEEE 754 foram omitidos nesta implementação. Dentre os requisitos não contemplados, merecem destaque a operação aritmética de divisão e a seleção dos diferentes modos de arredondamento definidos.

2) A corretude do sistema SRTI, na versão corrente, não foi alcançada, justamente pelo fato de que as operações de ponto-flutuante subjacentes sempre aplicam arredondamento para 0, mas este requisito pode ser alcançado caso a função $rshift$ seja aprimorada, de forma que o arredondamento seja aplicado para baixo nas operações entre extremos à esquerda de intervalos, e para cima entre extremos à direita.

Das considerações acima, decorre a necessidade, em primeiro lugar, que os requisitos do Padrão IEEE 754 não atendidos sejam implementados. Além disso, estão sendo deixadas como problema aberto as provas da terminação e da confluência, já que toda a teoria de sistemas de reescrita de termos apresentada no Capítulo 4 acabou se mostrando insuficiente para construir as provas destas propriedades. Mais teoremas e conceitos precisam ser explorados, para o caso de sistemas de reescrita condicionais (veja, por exemplo [Gra94]).

A partir daí, as respectivas provas devem ser fornecidas para o SRTI, a fim de se alcançar, efetivamente, um modelo formal para a computação intervalar que seja correto e ótimo. Uma significante contribuição, ainda, seria generalizar a definição de intervalo Padrão IEEE 754 dada na Seção 5.3 (que leva em conta apenas intervalos fechados) para a classe mais geral dos subconjuntos conexos de $\mathbb{R}$ (veja [HJE01] pg. 1060); esta generalização implica na totalidade de certas funções que, para intervalos fechados, são parciais. Por fim, a implementação em alguma ferramenta automática de reescrita será fundamental para validação e uso do sistema proposto.

Referências Bibliográficas

[ABL02] Benedito M. Acióly, Benjamín R. C. Bedregal, and Aarão Lyra. *Introdução a Teoria das Linguagens Formais, dos Autômatos e da Computabilidade*. Edições UNP, 2002.

[BA01] Benjamín R. C. Bedregal and Benedito M. Acióly. Lógica para a ciência da computação. UFRN/CCET/DIMAp, 2001.

[Bar98] Jorge Muniz Barreto. *Fundamentos de Matemática Aplicada à Informática*. Universidade Federal de Santa Catarina, 1998.

[Bed96] Benjamín R. C. Bedregal. *Sistemas de Informação Contínuos: Uma abordagem Lógica e Computacional para a Matemática Intervalar*. PhD thesis, Universidade Federal de Pernambuco, Brasil, 1996.

[Bee85] Michael J. Beeson. *Foundations of Construtive Mathematics*. Springer-Verlag, 1985.

[BH95] Jonathan P. Bowen and Michael G. Hinchey. Seven more myths of formal methods. *IEEE Software*, 12(3):34–41, 1995.

[Bir35] G. Birkhoff. On the structure of abstract algebras. In *Proceedings Cambridge Phil. Soc.*, volume 31, pages 433–454, 1935.

[BL74] W. S. Brainerd and L. H. Landweber. *Theory of Computation*. Wiley, 1974.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[Bow96] Jonathan Bowen. *Formal Specification and Documentation Using Z*. International Thomson Computer Press, 1996.

[Bra03] Christiano Braga. Lógica de reescrita e maude: Especificações em semântica operacional e executáveis. Simpósio Brasileiro de Linguagens de Programação, 2003.

[BS81] S. Burris and H. P. Sankappanavar. *A Course in Universal Algebra*. Springer-Verlag, 1981.

[CoF04] CoFI. The common framework initiative for algebraic specification and development. http://www.brics.dk/Projects/CoFI/, 2004.

[CS05] Adriano Xavier Carvalho and Regivan Hugo Nunes Santiago. Interval term rewriting system: A formal model for interval computation. Congresso Nacional de Matemática Aplicada e Computacional, São Paulo, Brasil, 2005.

[Cut80] Nigel Cutland. *Computability: An Introduction to Recursive Function Theory*. Cambridge University Press, 1980.

[CW91] D. Cohen and P. Watson. An efficiente representation of arithmetic for term rewriting. In *Lecture Notes in Computer Science*, volume 488, pages 240–251. 4th International Conference on Rewriting Techniques and Applications, 1991.

[Dav89] Ruth E. Davis. *Truth, Deduction, and Computation*. Computer Science Press, 1989.

[Der93] Nachum Dershowitz. A taste of rewrite systems. In P. E. Lauer, editor, *Functional Programming, Concurrency, Simulation and Automated Reasoning*, pages 199–228. Springer-Verlag, Berlin, Heidelberg, 1993.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*, pages 243–320. Elsevier, 1990.

[dM96] Thalles Cerqueira de Mello. Uma semântica algébrica para o λ-cálculo intervalar. Master's thesis, Universidade Federal de Pernambuco, Brasil, 1996.

[DV04] Nachum Dershowitz and Laurent Vigneron. Rewriting home page. http://www.brics.dk/Projects/CoFI/, 2004.

[dVY94] de Vries and Yamada. On termination and confluence of rewriting with real numbers. Technical report, NTT Communication Science Laboratories, Kyoto, 1994.

[Esc97] Martín Hotzel Escardó. *PCF extended with real numbers: a domain-theoretic approach to higher-order exact real number computation*. PhD thesis, University of London, England, 1997.

[GM96] Joseph A. Goguen and Grant Malcolm. *Algebraic Semantics of Imperative Programs*. MIT Press, 1996.

[Gog77] Joseph A. Goguen. Initial algebra semantics and continuous algebras. *Journal of the ACM*, 24(1):68–95, 1977.

[Gog04] Joseph Goguen. Joseph goguen's home page. http://www.cs.ucsd.edu/users/goguen/, 2004.

[Gol91] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, pages 5–48, 1991.

[Gra94] Bernhard Gramlich. On termination and confluence of conditional rewrite systems. In *Conditional Term Rewriting Systems*, pages 166–185, 1994.

[Har96] John Robert Harrison. *Theorem Proving with the Real Numbers*. PhD thesis, University of Cambridge, England, 1996.

[Hen88] Matthew Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.

[HJE01] T. Hickey, Q. Ju, and M. H. Van Emden. Interval arithmetic: From principles to implementation. *Journal of the ACM*, 48(5):1038–1068, 2001.

[Hol05] S. Hollasch. IEEE standard 754 floating point numbers. http://research.microsoft.com/ hollasch/cgindex/coding/ieeefloat.html, 2005.

[Hue80] G. Huet. Confluent reduction: Abstract properties and aplications to term rewriting systems. In *Journal of the ACM*, volume 27(4), pages 797–821, 1980.

[IEE85] IEEE. IEEE standard for binary floating-point arithmetic. IEEE Computer Society Press, 1985.

[Kea96] B. Kearfott. Interval computations: Introduction, uses and resources. *Euromath Boulletin*, pages 95–112, 1996.

[Ken89] J. R. Kennaway. Sequential evaluation strategies for parallel or and related reduction systems. In *Annals of Pure and Applied Logic*, volume 43, pages 31–56, 1989.

[Ken95] R. Kennaway. Complete term rewrite systems for decimal arithmetic and other total recursive functions. Second International Workshop on Termination, La Bresse, France, 1995.

[KK01] C. Kirchner and H. Kirchner. *Rewriting Solving Proving*. LORIA, INRIA & CNRS, 2001.

[KN85] D. Kapur and P. Narendran. A finite thue system with decidable word problem and without equivalent finite canonical system. In *Theoretical Computer Science*, volume 35, pages 337–344, 1985.

[Lop04] Katiane R. Lopes. A linguagem de especificação algébrica CASL e o tipo de dados intervalos. Master's thesis, Universidade Federal do Rio Grande do Norte, Brasil, 2004.

[Lyr03] Aarão Lyra. *Uma Fundamentação Matemática para o Processamento de Imagens Digitais Intervalares*. PhD thesis, Universidade Federal do Rio Grande do Norte, PPgEE, Brasil, 2003.

[MG85] J. Meseguer and Joseph A. Goguen. *Initiality, induction and computability*. Cambridge University Press, 1985.

[Moo62] R. E. Moore. *Interval Arithmetic and Automatic Error Analysis in Digital Computing*. PhD thesis, Department of Mathematics, Stanford University, Stanford, 1962.

[Moo79] R. E. Moore. Methods and applications of interval analysis. Society for Industrial and Applied Mathematics, 1979.

[Mos01] Peter D. Mosses. CoFI: The common framework initiative for algebraic specification and development. In *Current Trends in Theoretical Computer Science*, pages 153–163, 2001.

[MR99] T. Mossakowski and M. Roggenbach. The datatypes real and complex in CASL, 1999. Note M-7, in [24].

[O'D77] M. J. O'Donnell. *Computing in Systems Described by Equations*. Springer-Verlag, 1977.

[PS96] Ben Potter and Jane Sinclair. *An Introduction to Formal Specification and Z*. Prentice Hall Int., 1996.

[Rin98] Maurício Ayala Rincón. Combinação de mecanismos dedutivos aritméticos e equacionais. Seminário de pós-graduação, Depto. Ciência da Computação - Universidade de Brasília, 1998.

[Rin04] Maurício Ayala Rincón. *Fundamentos da Programação Lógica e Funcional*. Depto. Ciência da Computação - Universidade de Brasília, 2004.

[Rob00] Ken Robinson. An introduction to b method. University of New South Wales - Scool of Computer Science and Engineering, 2000.

[RS63] H. Rasiowa and R. Sikorski. The mathematics of metamathematics. In *Monografie Matematyczne*, 1963.

[RSM99] Markus Roggenbach, Lutz Schroder, and Till Mossakowski. Specifying real numbers in CASL. In *Workshop on Algebraic Development Techniques*, volume 1827 of *LNCS*, pages 146–161, 1999.

[Rum88] S. M. Rump. Algorithms for verified inclusions: Theory and practice. In *Reliability in Computing: The Role of Interval Methods in Scientific Computing*, pages 109–126, 1988.

[RW03] Kenneth A. Ross and Charles R. B. Wright. *Discrete Mathematics*. Prentice Hall Int., 5 edition, 2003.

[San99] Regivan H. N. Santiago. *Teoria das Equações Intervalares Locais*. PhD thesis, Universidade Federal de Perbambuco, Brasil, 1999.

[SB04] Regivan H. N. Santiago and Benjamín R. C. Bedregal. Computabilidade: Os limites da computação. notas em matemática aplicada. In *SBMAC*, volume 11. Sociedade Brasileira de Matemática Aplicada e Computacional, 2004.

[SBA05] Regivan H. N. Santiago, Benjamín R. C. Bedregal, and Benedito M. Acióly. Formal aspects of correctness and optimality of interval computations. *Formal Aspects Of Computing*, Submitted, 2005.

[Smi94] Carl H. Smith. *A Recursive Introduction to the Theory of Computation*. Springer-Verlag, 1994.

[Wal90] G. William Walster. Interval arithmetic: The new floating-point arithmetic paradigm. Fortran Compiler Technology, 1990.

[Wor96] J. B. Wordsworth. *Software Engineering with B*. Addison-Wesley, 1996.

[WZ95] H. R. Walters and H. Zantema. Rewrite systems for integer arithmetic. Rta, Centrum voor Wiskunde en Informatica (CWI), 1995.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo