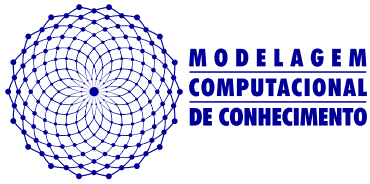


Dissertação de Mestrado

Especificação Semântica de *LaND*: uma linguagem para o Método das Diferenças Finitas

Fernando Antônio Dantas Gomes Pinto
fernando.dantas@maceio.al.gov.br

Maceió, Dezembro de 2013

Fernando Antônio Dantas Gomes Pinto

Especificação Semântica de *LaND*: uma linguagem para o Método das Diferenças Finitas

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-Graduação em Modelagem Computacional de Conhecimento do Instituto de Computação da Universidade Federal de Alagoas.

Orientadora: Dra. Eliana Silva de Almeida
Coorientador: Dr. Leonardo Viana Pereira

Maceió, Dezembro de 2013

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecária: Helena Cristina Pimentel do Vale

P659e Pinto, Fernando Antônio Dantas Gomes.

Especificação Semântica de LaND: uma linguagem para o Método das Diferenças Finitas / Fernando Antônio Dantas Gomes Pinto. – 2013
71 f. : il.

Orientadora: Eliana Silva de Almeida.
Coorientador: Leonardo Viana Pereira.
Dissertação (Mestrado em Modelagem Computacional de Conhecimento) – Universidade Federal de Alagoas. Instituto de Computação. Maceió, 2013.

Bibliografia: f. 63–66.
Apêndices: f. 67–71..

1. Especificação semântica. 2. Método das diferenças finitas. 3. Communicating Sequential Processes (CSP). I. Título.

CDU: 004.738.5

UNIVERSIDADE FEDERAL DE ALAGOAS/UFAL
Programa de Pós-Graduação em Modelagem Computacional de Conhecimento
Avenida Lourival Melo Mota, Km 14, Bloco 09, Cidade Universitária
CEP 57.072-900 – Maceió – AL – Brasil
Telefone: (082) 3214-1364

Membros da Comissão Julgadora da Dissertação de Mestrado de Fernando Antônio Dantas Gomes Pinto, intitulada: "Especificação Semântica de LAND: uma Linguagem para o Método das Diferenças Finitas", apresentada ao Programa de Pós-Graduação em Modelagem Computacional de Conhecimento da Universidade Federal de Alagoas em 20 de dezembro de 2013, às 14h30min, na sala de aula do Mestrado em Modelagem Computacional de Conhecimento da UFAL.

COMISSÃO JULGADORA

Prof. Dr. Leonardo Viana Pereira
UFAL – Instituto de Computação
Orientador

Profa. Dra. Eliana Silva de Almeida
UFAL – Instituto de Computação
Orientadora

Prof. Dr. João Carlos Cordeiro Barbirato
UFAL – Centro de Tecnologia
Examinador

Prof. Dr. Carlos Bazilio Martins
Pólo Universitário de Rio das Ostras/UFF – Instituto de Ciência e Tecnologia
Examinador

Maceió, dezembro de 2013.

AGRADECIMENTOS

Nestes anos de mestrado, de muito estudo e empenho, tenho a obrigação de agradecer a algumas pessoas que foram fundamentais para a realização desta conquista. Assim, expresso através de poucas palavras a sua importância nesta fase da minha vida e, com certeza, ainda contribuirão com significativa importância nos meus próximos desafios.

Primeiramente gostaria de agradecer ao meu pai Alderico, por ter ensinado os verdadeiros valores da vida como a honestidade e a lealdade. Pilares fundamentais que definiram o meu caráter. A minha mãe, Mary, por ter superado grandes dificuldades para que eu tivesse uma boa educação. Às minhas irmãs Juliana e Aurelina, que sempre estiveram ao meu lado, compartilhando comigo meus sofrimentos e dificuldades. Agradeço a todos da minha família, pelo incomensurável apoio nesta conquista. Pela compreensão das minhas ausências. Obrigado a todos.

Ao Prof. Alexandre Paes pelo grande apoio inicial dado no mestrado.

Ao Prof. Adilson Santos, ao qual nos momentos mais difíceis tinha sempre um valioso conselho a ser dado. Muito obrigado as observações na escrita deste trabalho.

A Professora Eliana Silva de Almeida, minha orientadora, pelo apoio, motivação, desafio e, principalmente, pelo equilíbrio e paciência. Características estas que sempre renovavam as minhas energias nos momentos mais difíceis. Minha eterna gratidão. Muito obrigado “Profa”.

Também quero agradecer imensamente ao Professor Leonardo Viana, meu orientador, pala paciência e pelas orientações fundamentais deste trabalho. Muito obrigado “Léo”.

Aos amigos do mestrado, Felipe Prata, Fabiano Brião e Leonardo Torres, pelas incontáveis horas que passamos juntos e os bons momentos no mestrado.

Aos amigos da UFRN, Rodrigo Gaete, Eduardo Marcondes, Iria Cosme e Natal Henrique pelos momentos incríveis em que passamos juntos quando estive em Natal.

À minha amada esposa Simone, que nunca deixou de me apoiar, me dar forças nos momentos mais estressantes e, principalmente, de compreender pelo tempo que precisei dedicar a este trabalho e não pude compartilhar de momentos importantes de sua vida.

Finalmente, ao meu amado filho Fernando ao qual dedico este trabalho e que sirva, de alguma forma, como uma fonte de inspiração na sua formação.

RESUMO

Ciência e Engenharia Computacional (CSE) é uma disciplina relativamente nova que lida com o desenvolvimento e aplicação de modelos computacionais e simulações, muitas vezes associada à computação de alto desempenho. A utilização efetiva de métodos de CSE apresenta uma barreira para engenheiros e cientistas, sua falta de formação específica em algoritmos, estruturas de dados, programação paralela e computação de alto desempenho. Muitas linguagens artificiais, de propósito geral ou específico, têm sido desenvolvidas; Matlab, Scilab e as bibliotecas de programação numéricas Basic Linear Algebra Subprograms(BLAS), LINPACK, EISPACK, LAPACK e ScaLAPACK são alguns exemplos. Na maioria dos casos o próprio engenheiro ou cientista desenvolve, em linguagem de programação específica, o software que atende as suas necessidades. O problema deste modelo está no esforço que é transferido ao usuário no desenvolvimento destes algoritmos. Além de ter que conhecer o domínio do problema e o método numérico a ser utilizado, ele deverá tratar do desenvolvimento do programa computacional, implementando o algoritmo para solução do problema. O objetivo deste trabalho é apresentar a especificação semântica formal de *LaND - La*nguage of *N*umerical *D*iscretization, uma linguagem artificial capaz de minimizar a complexidade no desenvolvimento do software científico para os problemas que envolvem simulações a partir das Equações Diferenciais Parciais com o Método das Diferenças Finitas. O pressuposto neste trabalho é que o estudante, engenheiro ou pesquisador deve apenas se preocupar com os aspectos inerentes à solução dentro do domínio de um problema, deixando a cargo da ferramenta a geração automática do programa equivalente. Esta é uma proposta inicial do modelo com foco nos problemas hiperbólicos de segunda ordem com a malha computacional geometricamente uniforme. A abordagem está fundamentada por técnicas formais da computação como a Semântica Denotacional, responsável pelo mapeamento dos objetos matemáticos presentes no modelo de aproximação numérica, dando significado a estas construções, e *Communicating Sequential Processes* (CSP), um formalismo utilizado para descrever os padrões de comunicação entre os nós da malha computacional.

Palavras-chave: Especificação semântica. Método das Diferenças Finitas. CSP.

ABSTRACT

Computer Science and Engineering (CSE) is a relatively new subject. It deals with applying and developing computer models and simulations, and it is often associated to high-performance computing. Using CSE methods effectively is currently an obstacle to engineers, due to their lack of specific training in algorithms, data structures, parallel programming and high-performance computing. Many artificial languages, either general or specific, have been recently developed: Matlab, Scilab and the numeric programming libraries Basic Linear Algebra Subprograms (BLAS), LINPACK, EISPACK, LAPACK and ScaLAPACK for instance. Generally, the engineer or scientist develops on their own a specific programming language, a software to meet their needs. The problem found within this process is the struggle transferred to the user as a consequence of these algorithms development. Besides the need to have total control over the problem and numeric method in question, they must deal with the development of the computer program implementing the algorithm for the problem resolution. The objective here is to present the formal semantic specifications for *LaND - La*nguage of *N*umerical *D*iscretization, an artificial language able to reduce the complexity of scientific software development for problems involving Partial Differential Equations simulations with Finite Difference Methods. It is assumed the student, engineer or researcher only concern should be the inherent aspects of the solution within a certain problem, leaving the automatic generation of an equivalent program to the tool. This is a initial model proposal with focus on second-order hyperbolic problems with geometrically uniform computational mesh. Our approach is based on formal techniques of computing, such as denotational semantics, responsible for the mapping the mathematical objects in the numerical approximation model, giving meaning to these structures, and *Communicating Sequential Processes* (CSP), a formalism used to describe the communication patterns within the computational mesh.

Keywords: Semantic Specification. Finite Difference Method. CSP.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

Em um período de constantes mudanças tecnológicas e mudanças de paradigmas, os computadores tornaram-se poderosas ferramentas usadas nas mais diversas áreas do conhecimento, desde o seu uso como ferramenta de apoio pessoal, atendendo as necessidades mais simples, como também de natureza científica, tratando de simulações de fenômenos encontrados na natureza. Por causa desta variedade de aplicações muitas linguagens artificiais, seja de propósito geral e/ou específicos, têm sido desenvolvidas nas últimas décadas. Como exemplo as linguagens Matlab, Scilab e das bibliotecas de programação numéricas *Basic Linear Algebra Subprograms* (BLAS), LINPACK, EISPACK, LAPACK e ScaLAPACK onde o produto final pode ser classificado como um software científico.

Tipicamente, o software científico possui como principal característica o seu uso amplamente adotado pela comunidade científica em suas respectivas áreas de atuação. Na maioria dos casos o próprio cientista desenvolve, em linguagem de programação específica, o software que atenda as suas necessidades.

Segundo Sebesta (2003), o que faz uma aplicação ser classificada como científica é a natureza numérica dos dados processados. Geralmente estas se utilizam de estruturas de dados simples, mas que exigem um grande esforço de computação para uma precisão quanto ao elevado número de dados. Muitos destes esforços computacionais estão ligados à resolução de problemas de larga escala de cálculos, como apresentado em Oliveira & Stewart (2006), produzindo uma grande quantidade de dados (a exemplo das soluções de uma Equação Diferencial Parcial), para solucionar os fenômenos que ocorrem na natureza nas áreas da eletricidade, mecânica, fluídos, etc.

Como dito, as resoluções destes problemas podem ser modeladas a partir de métodos numéricos computacionais que minimizam o esforço na geração destes dados de análise. O problema deste modelo está no esforço que é transferido ao cientista para o desenvolvimento destes algoritmos. Além de ter que conhecer o domínio do problema e o método numérico a ser tratado (condições de contorno, condições iniciais e funções de aproximação), ele é levado a assumir uma atividade que vai além de suas habilidades inerentes à pesquisa: deverá se preocupar com o modelo computacional (algoritmo) de solução do problema.

Com base no apresentado, faz-se a seguinte pergunta: seria possível a especificação de uma linguagem de programação capaz de minimizar a complexidade no desenvolvimento do software científico para os problemas que envolvem simulações com equações diferenciais parciais? O pressuposto neste trabalho é que o cientista ou engenheiro deve apenas se preocupar com os aspectos inerentes ao domínio do problema, tais como condições de contorno, valores iniciais, tamanho da malha numérica computacional, etc, deixando a cargo da ferramenta a geração automática do programa equivalente.

1.2 Contribuições

Uma das preocupações no desenvolvimento de linguagens artificiais é quanto ao seu grau de expressividade [1], legibilidade [2] e significado dados as suas construções sintáticas. Neste trabalho será considerado que a modelagem semântica, da linguagem proposta, deverá ser projetada com o objetivo de fornecer este significado às resoluções das Equações Diferenciais Parciais (EDPs) por Método das Diferenças Finitas (MDFs), dentro de um domínio específico, com uma sintaxe simples de ser compreendida.

Os programas gerados, a partir desta nova linguagem, devem ser naturalmente entendidos, tornando-os comumente fáceis de serem lidos, consequentemente reduzindo o esforço de manutenção.

Um outro fator determinante, quanto ao seu grau de expressividade e um das características da modelagem proposta neste trabalho, foi a criação de um baixo número de componentes básicos da linguagem. Segundo Sebesta (2003), uma linguagem artificial com um grande número de componentes básicos é mais difícil de ser aprendida e/ou compreendida do que uma com poucos componentes.

Diante do apresentado, esta especificação semântica será um arcabouço para uma nova linguagem capaz de abstrair a complexidade envolvida no desenvolvimento de programas que implementam os métodos numéricos envolvidos nas resoluções das Equações Diferenciais Parciais.

No processo de validação deste trabalho, os seguintes artigos foram produzidos e públicados:

- Pinto, F. D., Almeida, E. S. e Viana, L. P. (2011), Detecção de paralelismo em equações discretizadas por aplicação do método das diferenças finitas em equações diferenciais parciais, in ‘I ERAD - NE – Escola Regional de Alto Desempenho, Região Nordeste’. URL http://www.infojr.com.br/ERAD/view/ERAD_NE_2011_ForumPG.pdf.

[1] A expressividade de uma linguagem é uma característica que permite que uma grande quantidade de computação seja realizada com um programa muito pequeno (Sebesta, 2003).

[2] Programas de difícil leitura complicam sua interpretação, consequentemente sua manutenção.

- Pinto, F. D., Almeida, E. S. e Viana, L. P. (2013), Especificação Semântica de uma Linguagem para o Método das Diferenças Finitas, in 'XXXIV Iberian Latin American Congress on Computational Methods in Engineering (CILAMCE)'.

1.3 Objetivos

O objetivo geral deste trabalho é propor uma linguagem DSL[3] que trata a resolução de problemas fenomenológicos que envolvem a discretização das Equações Diferenciais Parciais pelo Método das Diferenças Finitas. Nesta proposta, o uso do formalismo da Semântica Denotacional dará significados às funções semânticas mapeadas a partir destas EDPs.

São objetivos específicos:

- Apresentar a importância da especificação semântica no projeto de linguagens;
- Estudar as Equações Diferenciais bem como os métodos de resolução numérica, com ênfase no desenvolvimento das equações hiperbólicas;
- Especificação semântica denotacional, utilizando CSP.

1.4 Organização da Dissertação

Além deste capítulo introdutório, esta dissertação está dividida em cinco capítulos. O conteúdo de cada um deles é descrito a seguir:

Capítulo 2: São apresentadas a Equações Diferenciais Ordinárias e Parciais e suas fenomenologias, com ênfase nas EDPs e suas resoluções a partir de métodos numéricos computacionais, especificamente o Método das Diferenças Finitas.

Capítulo 3: São introduzidos os Métodos de Definições Semânticas e alguns formalismos: Semântica Operacional, Semântica Denotacional e Semântica Axiomática, exemplificando suas características e aplicabilidades.

Capítulo 4: É apresentada a *Communication Sequential Processes* (CSP) , uma linguagem de especificação formal para processos concorrentes, e discutidas suas sintaxe e semântica.

Capítulo 5: Nesse capítulo será especificado, com uso dos formalismos da Semântica Denotacional e CSP, a linguagem de programação para resolução dos Métodos das Diferenças Finitas.

[3] *Domain-Specific Language* (DSL) é um tipo de linguagem de programação dedicada à um domínio de problema específico (Cook et al., 2007))

Capítulo 6: Nesse capítulo serão apresentadas as conclusões e sugestões de trabalhos futuros.

Os apêndices A e B, apresentam, respectivamente, o programa que, por meio de técnicas de aproximação numérica, resolve um problema clássicos da engenharia (equação diferencial de segunda ordem hiperbólica), e a modelagem de outros domínios semânticos de aproximação (acurácia).

Este capítulo apresentou a abordagem do problema foco nesta dissertação e as contribuições científicas conseguidas até o momento. No próximo capítulo serão apresentadas, de forma breve, as Equações Diferenciais Ordinárias e Parciais bem como será demonstrado, com exemplo, o processo de resolução de uma EDP pelo Método das Diferenças Finitas.

2 EQUAÇÕES DIFERENCIAIS E MÉTODOS NUMÉRICOS

EQUAÇÕES diferenciais são equações matemáticas que contém derivadas, de várias ordens, de uma função com uma ou mais variáveis independentes (Brannan & Boyce, 2008).

Equações diferenciais (EDs) desempenham um papel de destaque na engenharia, física, economia e outras disciplinas. Alguns exemplos importantes de EDs são: a Segunda Lei de Newton na dinâmica, as equações de Maxwell no eletromagnetismo, a equação do calor na termodinâmica, a equação de campo de Einstein na relatividade geral ou as equações de Navier-Stokes na dinâmica dos fluidos (Chaquet & Carmona, 2012). Todos estes fenômenos, comumente observados na natureza, podem ser modelados por Equações Diferenciais e são motivo de estudos por cientistas e engenheiros quanto a sua aplicabilidade, modelagem e simulação.

Neste capítulo será apresentada uma introdução às Equações Diferenciais, Ordinárias (EDOs) e Parciais (EDPs), bem como o Método das Diferenças Finitas (MDFs) aplicado a um problema diferencial evolutivo, particularizando as equações hiperbólicas com destaque à equação da onda.

2.1 Equações Diferenciais

Uma equação diferencial é uma equação que contém uma ou mais derivadas de uma função. Esta característica, inerente ao modelo do domínio a ser trabalhado, faz com que existam variedades de tipos desta equação. Quanto às derivadas, as Equações Diferenciais Ordinárias são equações que contém derivadas de uma função com uma única variável independente (derivadas simples), posta na forma:

$$F\big(x, y(x), y'(x), y''(x), \dots y^{(n)}(x)\big) = 0. \tag{2.1}$$

Já as Equações Diferenciais Parciais são equações que contém derivadas de uma função com duas ou mais variáveis independentes (derivadas parciais) (Knabner & Angermann, 2003). Comumente estas equações são apresentadas na forma:

$$F(x, y, u, u_x, u_y, u_{xx}, u_{xy}, u_{yy}) = 0, \tag{2.2}$$

onde u é função das variáveis independentes x e y.

2.1.1 Ordem e Grau das Equações Diferenciais

Para um bom entendimento das equações diferenciais ordinárias ou parciais, estas são classificadas quanto à sua ordem e seu grau.

A ordem de uma equação diferencial é a ordem da maior derivada presente na equação (Brannan & Boyce 2008). Cita-se como exemplo a Equação (2.1), definida como: uma equação diferencial ordinária de ordem n que expressa as relações entre a variável independente x e os valores da função y em suas n primeiras derivadas.

Para classificar uma equação quanto ao seu grau, observa-se o maior expoente da derivada de maior ordem.

Para ilustrar o que foi apresentado, a tabela 2.1 apresenta algumas equações e suas classificações quanto à natureza de ordem e grau.

Tabela 2.1: Exemplos de classificação das Equações Diferenciais.

Equação	Tipo	Ordem	Grau
$\frac{\partial y}{\partial x} = 3x - 1$	*EDO*	*Primeira ordem*	*Primeiro grau*
$\frac{\partial z}{\partial x} - x\frac{\partial z}{\partial y} = 3xyz$	*EDP*	*Primeira ordem*	*Primeiro grau*
$\left(\frac{\partial^2 y}{\partial x^2}\right)^3 - 5\left(\frac{\partial y}{\partial x}\right)^4 = \cos x$	*EDO*	*Segunda ordem*	*Terceiro grau*
$\frac{\partial^2 y}{\partial x^2} - 7\frac{\partial y}{\partial x} + 12y = 6e^{5x}$	*EDO*	*Segunda ordem*	*Primeiro grau*
$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$	*EDP*	*Segunda ordem*	*Primeiro grau*

Fonte: Adaptada de Brannan & Boyce (2008).

2.1.2 Equação Diferencial Parcial

Considerada a forma mais comum em problemas de ciência e engenharia (Claudio & Marins, 2000), as EDPs lineares de segunda ordem, com duas variáveis, podem ser apresentadas na forma,

$$Au_{xx} + Bu_{xy} + Cu_{yy} + Du_x + Eu_y + Fu + G = 0, \tag{2.3}$$

onde u é uma função das variáveis independentes x e y e todos os coeficientes A, B, C, D, E e F são dependentes destas variáveis independentes.

Outra forma de apresentar esta mesma equação pode ser,

$$a\frac{\partial^2 \varnothing}{\partial x^2} + b\frac{\partial^2 \varnothing}{\partial x \partial y} + c\frac{\partial^2 \varnothing}{\partial y^2} + d\frac{\partial \varnothing}{\partial x} + e\frac{\partial \varnothing}{\partial y} + f\varnothing + g = 0, \tag{2.4}$$

onde a, b, c, d, e, f e g podem ser funções das variáveis independentes x e y e da variável dependente $\varnothing$.

Como exemplo apresentam-se as Equações Diferenciais Parciais Lineares mais populares e frequentemente utilizadas nas diversas áreas da engenharia:

- Equação do calor (tridimensional),

$$\frac{\partial u}{\partial t} = \eta\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right), \tag{2.5}$$

  ou com a uma função que contém uma fonte externa de calor,

$$\frac{\partial u}{\partial t} = \eta\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right) + f(x). \tag{2.6}$$

- Equação de Laplace bidimensional:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0. \tag{2.7}$$

- Equação de Laplace tridimensional:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = 0. \tag{2.8}$$

- Equação de Poisson bidimensional que modela problemas de transferência de calor sobre um corpo físico a partir de uma fonte de calor,

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y). \tag{2.9}$$

- Equação da Onda unidimensional: Um exemplo de equação diferencial parcial linear do tipo hiperbólica. Neste caso, admite solução única se as funções f e g têm derivadas segundas contínuas no intervalo $(0,a)$ e se $f(a) = f(0) = 0$ (Zill & Cullen, 2001).

$$c^2\frac{\partial^2 u}{\partial x^2} = \frac{\partial^2 u}{\partial t^2}. \tag{2.10}$$

Na Equação (2.10) a variável $t > 0$ representa o tempo, $x \in \mathbb{R}$ é a variável espacial e $c > 0$ é uma constante que representa a velocidade de propagação da onda.

O que foi discutido nesta seção está relacionado à modelagem matemática de problema fenomenológicos comumente explorados. Esta modelagem, por ser analítica, gera um alto grau de esforço na obtenção de dados para análise. Diante deste fato serão discutidas, na próxima seção, algumas técnicas que facilitam a obtenção de uma solução por aproximação, através de métodos numéricos, que minimizam este esforço.

2.2 O Método das Diferenças Finitas

Muitos modelos matemáticos foram estabelecidos a partir de modelos fenomenológicos que recaem em sistemas de Equações Diferenciais Ordinárias ou Parciais que se apresentam com um elevado número de incógnitas. De modo geral, soluções analíticas para estas equações não são viáveis devido as incógnitas presentes no modelo, que onera o especialista no seu desenvolvimento.

Nestes casos, a modelagem destes fenômenos recai sobre a necessidade de aproximações destas soluções através de métodos numéricos computacionais. O **Método das Diferenças Finitas** é uma das técnicas numéricas de aproximação mais presentes na solução destes problemas. Esquemas de diferenças finitas foram utilizados pela primeira vez por Leonhard Euler, no século XVIII, para encontrar soluções aproximadas das equações diferenciais, técnica conhecida como método de Euler (Brannan & Boyce, 2008). Após 1945 as pesquisas sobre o tema ganharam força com o advento dos supercomputadores que trouxeram a capacidade de execução paralela, reduzindo o tempo de processamento da solução.

Atualmente, o Método das Diferenças Finitas fornece uma abordagem poderosa para resolver equações diferenciais e são amplamente utilizados em qualquer campo das ciências naturais aplicadas. Equações com coeficientes variáveis e mesmo problemas não-lineares podem ser tratados por esta técnica. Erros de aproximação e/ou arredondamento, que são considerados inevitáveis no processo computacional, podem ser controlados por análises da estabilidade numérica destes esquemas.

2.2.1 Aproximações por Diferenças Finitas

O método das diferenças finitas consiste em substituir as derivadas parciais presentes na equação diferencial por aproximações por diferenças finitas. O primeiro passo é a discretização do domínio da variável independente. Neste caso, discretizar é dividir o domínio de cálculo em certo número de subdomínios delimitados por pontos em uma malha. O segundo passo é gerar as aproximações (explícito e implícito) para as derivadas das variáveis dependentes, nestes pontos discretizados.

Na aproximação, adotando o método explícito, as equações são independentes, permitindo solução por computação direta. Este método tem solução mais ágil e tem como fator negativo a instabilidade. No método implícito, apesar das condições de estabilidade mais favoráveis, as equações resultantes são acopladas, o que exige a resolução de um sistema de equações a cada passo de integração no tempo, podendo tornar o método mais lento e/ou de difícil paralelização (Robaina et al., 2005).

Para entender a técnica das diferenças finitas é necessário considerar as concepções fundamentais em torno da teoria de aproximação da série de Taylor. O domínio dado por uma EDP é primeiramente subdividido por um conjunto finito de pontos (nós), e suas relações (informações) são obtidas utilizando-se expansões da série de Taylor (Lapidus & Pinder, 1999), aplicadas a uma malha computacional, como apresentado nas Figuras 2.1 e 2.2. A derivada em cada ponto é então trocada, discretizada, por uma aproximação de finitas diferenças. A série de Taylor é a base para todas as aproximações numéricas usadas pelo Método das Diferenças Finitas (Schneider, 2007).

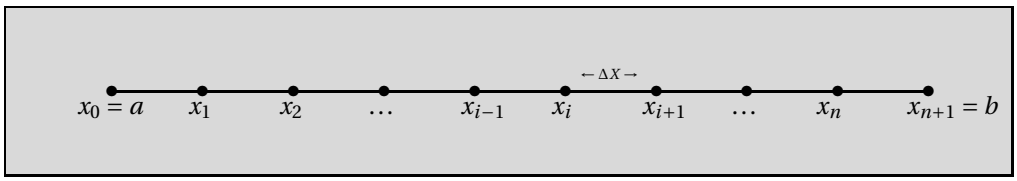
2.2.2 Expansão da Série de Taylor para problemas unidimensionais/EDO

Sejam os sistemas que apresentam uma única variável independente $u(x)$, definida no intervalo dado $a \leq x \leq b$. Suponha que este intervalo $[a,b]$ contenha o conjunto $x_0, x_1, x_2, \ldots, x_i, x_{i+1}, \ldots, x_n, x_{n+1}$, onde $x_0 = a$ e $x_{n+1} = b$. O valor de $u(x_i)$ é representado por u_i e sua discretização fica assim definida:

$$[u_i] = [u(a), u(x_1), u(x_2), \ldots, u(x_i), u(x_{i+1}), \ldots, u(x_n), u(b)].$$

O valor de $\Delta x = x_{i+1} - x_i$, representa o espaço da malha. Quando as malhas apresentam uniformidade (característica presente neste trabalho), este espaço é representado por $x_i = a + i\Delta x$ com $i = 0, \ldots, n+1$ e $\Delta x = (b-a)/(n+1)$.

Figura 2.1: Malha computacional unidimensional.



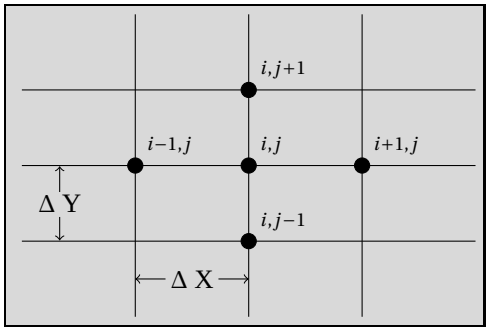
Fonte: Autor, 2013.

A Figura 2.1 representa esta malha computacional para os problemas unidimensionais ($u(x)$). O X_i representa o ponto central e ΔX representa a distância entre os pontos discretos (tamanho da malha).

2.2.3 Expansão da Série de Taylor para problemas bidimensionais/EDP.

De forma análoga ao apresentado nas EDOs, as equações EDPs representam casos em que o sistema apresenta duas variáveis independentes. Na Figura 2.2 é dada uma malha computacional bidimencional com suas variáveis independentes x e y, onde Δx e Δy representam (com muita frequência) as diferenças com relação ao espaço e ao tempo, respectivamente.

Figura 2.2: Malha computacional bidimensional.



Fonte: Autor, 2013.

Para o desenvolvimento da Série de Taylor, por exemplo, deve-se aproximar $\partial\varnothing/\partial x$ no ponto (x_i, y_i) por um diferença discreta para um valor finito (origem da denominação "diferença finita") de Δx, ficando esta série na forma:

$$\varnothing(x_i + \Delta x, y_i) = \sum_{m=0}^{\infty} \frac{(\Delta x)^m}{m!} \frac{\partial^m \varnothing(x_i, y_i)}{\partial x^m} \tag{2.11}$$

Dado o ponto (x_i, y_i) a sua equação à direita seria:

$$\varnothing(x_i + \Delta x) = \varnothing(x_i, y_i) + \frac{\partial \varnothing}{\partial x}|_i \Delta x + \frac{\partial^2 \varnothing}{\partial x^2}|_i \frac{\Delta x^2}{2!} + \frac{\partial^3 \varnothing}{\partial x^3}|_i \frac{\Delta x^3}{3!} + \ldots \tag{2.12}$$

E na sua esquerda, teríamos:

$$\varnothing(x_i - \Delta x, y_i) = \varnothing(x_i, y_i) - \frac{\partial_\varnothing}{\partial_x}|_i \Delta x + \frac{\partial^2 \varnothing}{\partial x^2}|_i \frac{\Delta x^2}{2!} + \frac{\partial^3 \varnothing}{\partial x^3}|_i \frac{\Delta x^3}{3!} + \ldots \tag{2.13}$$

Já para cima, tem-se:

$$\varnothing(x_i, y_i + \Delta y) = \varnothing(x_i, y_i) + \frac{\partial_\varnothing}{\partial_y}|_i \Delta y + \frac{\partial^2 \varnothing}{\partial y^2}|_i \frac{\Delta y^2}{2!} + \frac{\partial^3 \varnothing}{\partial y^3}|_i \frac{\Delta y^3}{3!} + \ldots \tag{2.14}$$

E finalmente para baixo, obtemos:

$$\varnothing(x_i, y_i - \Delta y) = \varnothing(x_i, y_i) - \frac{\partial_{\varnothing}}{\partial_y}|_i \Delta y + \frac{\partial^2 \varnothing}{\partial y^2}|_i \frac{\Delta y^2}{2!} + \frac{\partial^3 \varnothing}{\partial y^3}|_i \frac{\Delta y^3}{3!} + \ldots \tag{2.15}$$

Na próxima seção, no intuito de esclarecer o método de aproximação das derivadas por diferenças finitas que será o nosso modelo de referência da especificação semântica que este trabalho se propõe, tomamos como base o desenvolvimento de uma simples equação linear de segunda ordem. O MDF aplicado à Equação (2.10) ilustra como uma equação diferencial é transformada em um sistema de equações algébricas para, por exemplo, calcular os deslocamentos de uma corda, que está fixada nas suas extremidades nos pontos de espaço 0 e L (limites laterais), sendo estes dados posteriormente discretizados em um conjunto finito de pontos de uma malha computacional uniforme.

2.3 Aplicação do Método das Diferenças Finitas em equações hiperbólicas

Em problemas da física, uma onda é vista como uma perturbação oscilante de alguma grandeza física no espaço e periódica no decorrer do tempo. Neste exemplo, o MDF irá ilustrar um sistema de equações algébricas do cálculo dos deslocamentos realizados por uma corda que está fixa nas extremidades. Sendo estes dados posteriormente discretizados em um conjunto finito de pontos de uma malha computacional uniforme.

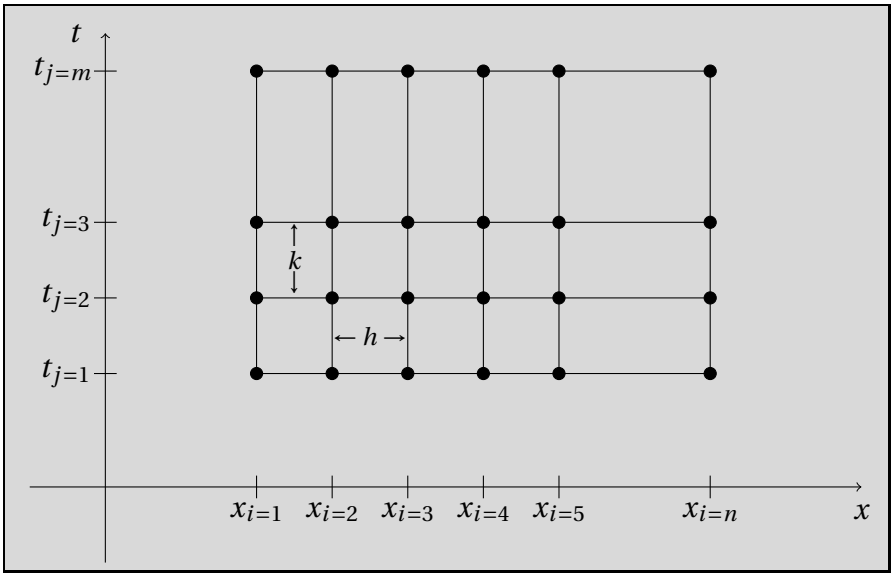
O primeiro passo neste método é substituir a equação de derivadas parciais (2.10), por sua respectiva equação de aproximação em suas diferenças centrais (2.16), para fins de ilustração com acurácia em segunda ordem no espaço (diferença central) e primeira ordem no tempo (diferença para frente),

$$\frac{c^2}{h^2}\Big[\, u\,(x+h,t) - 2u\,(x,t) + u\,(x-h,t)\,\Big] = \frac{1}{k^2}\Big[\, u\,(x,t+k) - 2u\,(x,t) + u\,(x,t-k)\,\Big] \tag{2.16}$$

O próximo passo é discretizar a região onde se procura a solução. É nesta etapa que será definida uma **malha computacional** (Figura 2.3) como um conjunto finitos de pontos distribuídos em um plano cartesiano. Neste processo de discretização, os valores da malha são definidos como o resultado de uma função das variáveis independentes no plano x (espaço) e plano t (tempo), onde estes planos são subdivididos em intervalos de espaços h e k ($\delta x = h$ e $\delta t = k$). Um ponto, em específico, desta malha é a localização indexada pela coordenada (i, j) no plano cartesiano, então $x = ih$ e $t = jk$. Por prática, os índices i e j são definidos como valores inteiros e as diferenças h e k são, respectivamente, os espaçamentos da malha no espaço (x)

e no tempo (t). Assim, para simplificação, pode-se definir: $(x,t) = (xi, tj) = (ih, jk)$. Neste trabalho, a especificação formal da linguagem tratará problemas onde o conjunto discreto de pontos apresenta-se uniformemente distribuído em malha estruturada bidimensional (posição *versus* tempo), ou seja, $h = k = 1$, intervalos unitários entre os pontos discretizados de x e t, conforme Equação (2.16).

Figura 2.3: Definição da malha nos eixos do espaço e do tempo.



Fonte: Adaptada de Burden & Faires (2011).

Portando, sejam x e t os planos do espaço e do tempo. Comumente adota-se a notação $u(x,t) = u_{i,j}$, para $i = 1,2,3,\ldots,n-1$ e $j = 1,2,3,\ldots,m-1$ (uniformes), e os intervalos h e k são definidos como $h=x/n$ e $k=t/m$, onde n é o número total dos intervalos de espaço e m é o número total dos intervalos de tempo.

Para a Equação (2.10) e sua respectiva discretização Equação (2.16), reescrevem-se estas para calcular os pontos interiores da malha (x_i, t_j) com base nas coordenadas (espaço e tempo), onde o método das diferenças é obtido utilizando o quociente das diferenças centrais para as segundas derivadas parciais,

$$\frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{k^2} - c^2 \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} = 0. \tag{2.17}$$

Definindo uma função $\lambda = ck/h$, onde este seu termo é o produto da constante da velocidade de propagação da onda (c) pela razão dos intervalos do espaço e do tempo, pode-se escrever a Equação (2.17) como,

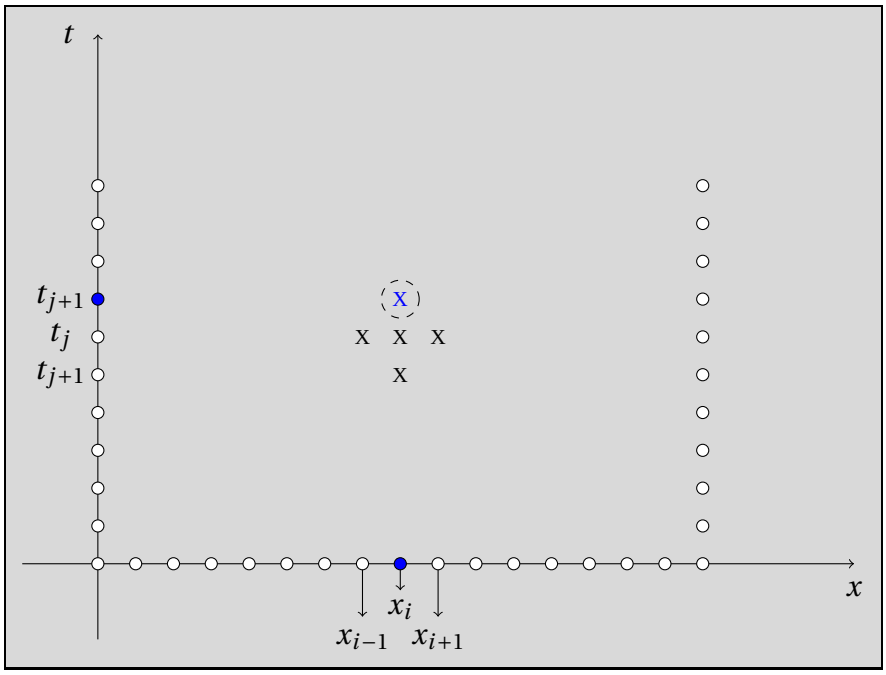
$$u_{i,j+1} - 2u_{i,j} + u_{i,j-1} - \lambda^2 u_{i+1,j} + 2\lambda^2 u_{i,j} - \lambda^2 u_{i-1,j} = 0. \tag{2.18}$$

Para resolver $u_{i,j+1}$, a aproximação com avanço no passo do tempo tem-se,

$$u_{i,j+1} = \lambda^2 u_{i+1,j} + 2(1-\lambda^2)u_{i,j} + \lambda^2 u_{i-1,j} - u_{i,j-1}. \tag{2.19}$$

A Figura 2.4, uma malha de pontos no plano cartesiano, demonstra o nó a ser calculado no ponto $u_{i,j+1}$ a partir dos nós dependentes $u_{i-1,j}, u_{i+1,j}, u_{i,j}$ e $u_{i,j-1}$

Figura 2.4: Ponto discreto ($u_{i,j+1}$) a ser calculado.



Fonte: Adaptada de Burden & Faires (2011).

Uma pré-condição, para execução do algoritmo do MDF, é definir as **condições de contorno** da região a ser estudada e as **condições iniciais** do problema.

Condições de contorno - Nesta etapa o engenheiro especifica um limite, seja espacial ou temporal, da região a ser estudada criando um subdomínio para a solução do problema. Neste exemplo de aplicação do MDF, estas condições expressam o fato que a corda estará fixa por ambos os lados, em alguma superfície na condição 0 <= x <= 1 e 0 <= t <= 1.

Condições iniciais - Uma segunda etapa é a resolução do problema de obtenção dos valores iniciais. Como o princípio básico é aplicar a Equação (2.19) para encontrar os valores da solução no tempo $j+1$. Para o caso onde $j = 1$, precisamos conhecer os valores de $u_{i,1}$, ou seja, as estimativas de u na primeira linha do tempo, no intuito de encontrar $u_{i,2}$. Ocorre que o valor de $u_{i,1}$, na primeira linha, depende dos valores de $u_{i,0}$ na linha zero do tempo e dos valores de

Tabela 2.2: Condição de contorno do problema.

Função nas coord.		Valores de Contorno
$u(0,t)$	=	0 Onde $t>0$
$u(1,t)$	=	0 Onde $t>0$

Fonte: Autor, 2013.

u_{i-1}. Para calcular estes últimos valores, utiliza-se a condição de velocidade inicial, obtendo-se o caso especial,

$$u_{i,1} = \frac{\lambda^2}{2}\left(u_{i+1,0} + u_{i-1,0}\right) + \left(1-\lambda^2\right)u_{i,0} + kg\left(x_i\right). \tag{2.20}$$

Para o desenvolvimento deste exemplo, foram definidos os valores iniciais apresentados na tabela 2.3.

Tabela 2.3: valores iniciais do problema para os tempos “0” e “1”.

Função de coord.		Valores iniciais
$u(x,0)$	=	$sen(\pi x)$
$u(x,1)$	=	A Equação (2.20)

Fonte: Autor, 2013.

Tabela 2.4: Aproximações numéricas dos deslocamentos da corda ao longo de tempo.

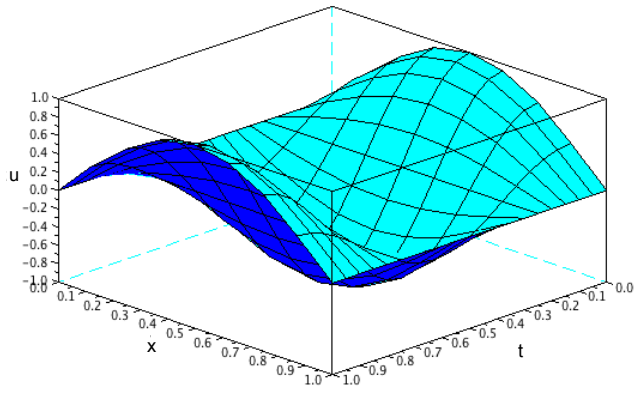
Tempo	Espaço								
	x = 0,00	x = 0,12	x = 0,25	x = 0,37	x = 0,50	x = 0,62	x = 0,75	x = 0,87	x = 1,00
0.00	0.0000	0.3827	0.7071	0.9239	1.0000	0.9239	0.7071	0.3827	0.0000
0.05	0.0000	0.3640	0.6727	0.8789	0.9513	0.8789	0.6727	0.3640	0.0000
0.10	0.0000	0.3099	0.5727	0.7482	0.8099	0.7482	0.5727	0.3099	0.0000
0.15	0.0000	0.2256	0.4169	0.5447	0.5896	0.5447	0.4169	0.2256	0.0000
0.20	0.0000	0.1193	0.2205	0.2881	0.3118	0.2881	0.2205	0.1193	0.0000
0.25	0.0000	0.0014	0.0026	0.0034	0.0037	0.0034	0.0026	0.0014	0.0000
0.30	0.0000	-0.1167	-0.2155	-0.2816	-0.3048	-0.2816	-0.2155	-0.1167	0.0000
0.35	0.0000	-0.2233	-0.4127	-0.5392	-0.5836	-0.5392	-0.4127	-0.2233	0.0000
0.40	0.0000	-0.3083	-0.5696	-0.7442	-0.8056	-0.7442	-0.5696	-0.3083	0.0000
0.45	0.0000	-0.3632	-0.6710	-0.8768	-0.9490	-0.8768	-0.6710	-0.3632	0.0000
0.50	0.0000	-0.3827	-0.7071	-0.9239	-1.0000	-0.9239	-0.7071	-0.3827	0.0000
0.55	0.0000	-0.3649	-0.6742	-0.8809	-0.9535	-0.8809	-0.6742	-0.3649	0.0000
0.60	0.0000	-0.3116	-0.5757	-0.7522	-0.8142	-0.7522	-0.5757	-0.3116	0.0000
0.65	0.0000	-0.2279	-0.4211	-0.5501	-0.5955	-0.5501	-0.4211	-0.2279	0.0000
0.70	0.0000	-0.1220	-0.2254	-0.2945	-0.3188	-0.2945	-0.2254	-0.1220	0.0000
0.75	0.0000	-0.0042	-0.0078	-0.0102	-0.0110	-0.0102	-0.0078	-0.0042	0.0000
0.80	0.0000	0.1140	0.2106	0.2752	0.2978	0.2752	0.2106	0.1140	0.0000
0.85	0.0000	0.2211	0.4085	0.5337	0.5777	0.5337	0.4085	0.2211	0.0000
0.90	0.0000	0.3066	0.5665	0.7402	0.8012	0.7402	0.5665	0.3066	0.0000
0.95	0.0000	0.3623	0.6694	0.8746	0.9467	0.8746	0.6694	0.3623	0.0000
1.00	0.0000	0.3826	0.7070	0.9238	0.9999	0.9238	0.7070	0.3826	0.0000

Fonte: Autor, 2013.

Com base nos dados gerados pelo algoritmo de aproximação (Tabela 2.4), a Figura 2.5 mostra o deslocamento espacial da corda, ao passar do tempo e com suas extremidades fixas no eixo do espaço.

Detalhes do algoritmo que simula a resolução numérica deste problema pode ser visto no apêndice (A). Um fator a ser considerado deste algorítmo é quanto à sua baixa legibilidade, ou seja, mediante a forma como foi escrito torna-se cansativo de ser lido e consequentemente entendido.

Figura 2.5: Vibração da corda modelado pela EDP hiperbólica



.

Fonte: Autor, 2013.

Neste capítulo foram apresentadas, de forma breve, as equações diferencias ordinárias e parciais; como estas se aplicam no seu contexto fenomenológico. Também foi apresentado o método de resolução numérica destas equações, especificamente o Método das Diferenças Finitas. Para contextualizar o tema, apresentou-se a resolução computacional para um problema que envolve EDPs: o problema da vibração de uma corda, típico problema de equações diferenciais do tipo hiperbólica.

No próximo capítulo serão tratadas algumas questões relacionadas a especificação de linguagens, com ênfase nos aspectos semânticos.

3 SEMÂNTICA DENOTACIONAL

Este capítulo introduz os conceitos que norteiam a Semântica Denotacional, um formalismo semântico utilizado para modelar os significados de programas na forma de objetos matemáticos (funções semânticas), que representam o efeito da execução das estruturas sintáticas da linguagem.

Este trabalho está concentrado em uma especificação Semântica Denotacional que dará apoio ao projeto de *LaND*. Além da Semântica Denotacional, também foi utilizado o formalismo da CSP ao qual será apresentado na seção 4.

Inicialmente serão analisados os aspectos que envolvem os projetos de análise sintática e, posteriormente, semântica de linguagens. Além da semântica denotacional, também serão apresentados outros paradigmas semânticos, como a Semântica Axiomática e Operacional.

3.1 Introdução à Definição Formal de Linguagens

No projeto de uma nova linguagem de programação é conveniente a adoção de formalismos que definam, com precisão, as idiossincrasias desta linguagem. Uma definição formal, de linguagens de programação, consiste no mínimo de duas partes, a *sintaxe* e a *semântica*. A parte sintática está preocupada com a forma das expressões que serão permitidas na linguagem, ou seja, as estruturas válidas para suas sentenças. Por outro lado, o formalismo da parte *semântica* está preocupado com o correto significado destas sentenças, presentes na sua parte sintática, e dos valores associados aos seus tipos de dados (Noonan & Tucker, 2008). Resumidamente, ambas as abordagens possuem uma estreita relação: a sintaxe decide a forma e a estrutura dos programas que são pré-requisitos básicos para descrever a semântica de programas.

Historicamente, o primeiro caso de formalização semântica para linguagens de programação se deu na década de 30, antes da primeira linguagem de programação dita de alto nível, com o λ-cálculo (ou cálculo lambda) de Alonzo Church e Stephen Cole Kleene, que em 1936 definiram regras de um sistema formal baseado no conceito de funções computáveis para execução de algoritmos (Sá & Silva, 2006).

Na época do surgimento das linguagens de alto nível, como o caso da Fortran (década de 50), as definições formais para a parte sintática e semântica ainda não existiam. O suporte

disponível aos projetistas eram apenas definições escritas em *Assembly*. No fim dos anos 50, as técnicas utilizadas nos projetos de linguagens de programação eram concentradas apenas no modelo sintático da linguagem, pois o estudo da semântica formal ainda era bastante incipiente.

Muitas destas técnicas de verificação sintática foram amplamente estudadas, desenvolvidas e aplicadas na indústria. Um exemplo deste cenário é quanto ao uso de gramáticas livre de contexto expressas em BNF [1], ou em diagramas de sintaxe [2], também conhecidos como grafos sintáticos. Estas técnicas foram e ainda são de grande benefício para cientistas da computação desde que John Backus e Peter Naur especificaram formalmente a sintaxe de Algol-58 e posteriormente Algol-60. Uma pequena amostra desta última pode ser vista na tabela 3.1, cujos elementos incluem símbolos terminais (em negrito) e não-terminais (delimitados por "<" e ">"). Backus apresentou pela primeira vez a notação BNF para descrever a sintaxe de linguagens de programação no comitê de Zurique, na *International Conference on Information Processing* de 1959 (Sebesta, 2003).

Tabela 3.1: Parte da BNF da linguagem Algol 60.

< *program* >	::=	< *block* > | < *compound statement* >
< *block* >	::=	< *unlabelled block* > | < *label* > : < *block* >
< *unlabelled block* >	::=	< *block head* > ; < *compound tail* >
< *block head* >	::=	**begin** < *declaration* > | < *block head* > ; < *declaration* >
< *compound statement* >	::=	< *unlabelled compound* > | < *label* >: < *compound statement* >
< *unlabelled compound* >	::=	**begin** < *compoundtail* >
< *compound tail* >	::=	< *statement* > **end** | < *statement* > ; < *compound tail* >
⋮		
< *type* >	::=	**real** | **integer** | **boolean**
< *specifier* >	::=	< *type* > **procedure**

Fonte: Adaptada de MacLennan (2012).

Atualmente, as linguagens de programação têm suas sintaxes definidas com a utilização dessas ferramentas. O resultado é uma sintaxe mais clara, melhores métodos de *parsing*, geradores de *parsers* e manuais de linguagens com maior expressividade.

De forma contrária, a popularidade e adoção de um formalismo semântico é inversamente proporcional à popularidade dos projetos com uso de uma análise sintática formal. Contraste a

[1] *Backus Naur Form*, um formalismo comumente utilizado para representar as gramáticas das linguagens de programação (Menezes, 2008)

[2] Nesta notação, os símbolos da gramática e não-terminais tem sua representação na forma de gráficos (Ricarte, 2008)

isto que, em mais de 50 anos de desenvolvimento do formalismo sintático, nenhum mecanismo semântico atingiu a popularidade universal da BNF na construção de novas linguagens (Allison, 1986). Neste período, definições semânticas eram normalmente feitas em linguagem natural, sendo susceptível de ambiguidade, gerando mais de uma interpretação possível para uma mesma construção sintática.

Hoje, sabe-se que a análise sobre os aspectos semânticos da linguagem é tão importante quanto o tratamento formal dado a sua sintaxe. Uma especificação semântica formal concisa assegura que um programa tenha sempre o mesmo significado, independente da sua forma ou ambiente de execução (Noonan & Tucker, 2008). Além desse fato, uma especificação semântica dá suporte ao projeto da especificação sintática da linguagem ao passo que, como dito, ambas se completam.

Podem-se destacar alguns motivos que levam a definição formal, tanto sintática quanto semântica, de uma linguagem de programação:

- Suporte aos programadores a partir de uma documentação precisa dos significados das construções da linguagem a partir de sua especificação completa.
- Suporte aos projetos de tradutores, fornecendo uma definição concisa do significado das construções, evitando ambiguidades na implementação da linguagem a partir de sua análise em um ponto de vista formal.
- Base para padronizações em linguagens artificiais.

Diante destes problemas apresentados, fica claro que apenas a aplicação do tratamento sintático formal não garante a eficiência nos projetos de linguagens. Um outro mecanismo formal, agora preocupado com os aspectos semânticos da linguagem, deverá ser utilizado em complemento à sua especificação.

3.2 Semântica de Linguagens

Como apresentado, uma especificação semântica assegura que um programa tenha sempre o mesmo “significado” em suas construções sintáticas válidas na linguagem. Com o objetivo de demonstrar os efeitos da execução de determinado programa, em geral, estas definições são frequentemente utilizadas como base de referência na construção de compiladores, justificando a adoção de um formalismo complementar ao projeto sintático da linguagem. Quando este formalismo é suprimido no projeto, ou mesmo quando demonstrada em linguagem natural, há grandes chances que a mesma possua ambiguidades em suas construções sintáticas. De fato, o uso de definições precisas auxiliam os projetistas e desenvolvedores de programas a entender o significado das construções da linguagem e os efeitos da execução dos seus programas.

Dois paradigmas semânticos se destacam e são comumente utilizados: a **semântica estática** e a **semântica dinâmica**. A **semântica estática** está relacionada apenas às formas legais

dos programas. Neste caso, tem-se o exemplo das regras impostas às variáveis que devem ser declaradas antes do seu efetivo uso e dos valores associados a seus tipos declarados (processo realizado na fase de compilação). A **semântica dinâmica** já é um paradigma que está preocupado em dar significado às expressões, instruções e às unidades básicas da linguagem. Ainda existe uma terceira abordagem, a **semântica declarativa**, utilizada para as linguagens do paradigma lógico onde suas declarações são construídas na forma de proposições lógicas (Sebesta, 2003). Segundo Krishnamurthi (2007), na semântica dinâmica, três abordagens tem sido especialmente utilizadas: a **Semântica Denotacional**, a **Operacional** e a **Axiomática**.

Uma das primeiras atividades a serem executadas, no processo de especificação semântica, é quanto à análise do conjunto base das entidades matemáticas e suas propriedades para seus tipos de dados *(semântica estática)*: o conjunto dos valores inteiros, reais, caracteres e booleanos de uma linguagem. Estas entidades e suas propriedades são limitadas pelas restrições impostas pelos computadores reais. Por exemplo, o tipo de dado primitivo *int*, em linguagem como C[3], não inclui a faixa completa dos valores inteiros. Um esforço da semântica é justamente definir um conjunto de propriedades e operações sobre estas entidades da linguagem, restringindo seu domínio.

Este conceito, conhecido como *Domínio da Semântica*, explica e dá noções de computabilidade às entidades de um domínio na forma de funções matemáticas que a denotam. A Semântica Denotacional, que será apresentada na seção 3.5, faz uso desta *Teoria dos Domínios* para restringir as entidades matemáticas de uma linguagem (Scott, 1982).

A próxima fase é definir a *semântica dinâmica* e uma das atividades iniciais é criar um modelo de estados com suas variáveis e valores associados, como segue:

$$estado = < var_1, val_1 >, < var_2, val_2 >, < var_n, \ldots, val_n >$$

Como apresentado em Sebesta (2003), o conceito de *estado* é representado por um conjunto de pares ordenados de variáveis (var_1, var_2, var_n) e seus valores associados (val_1, val_2, val_n) neste estado. Em uma perspectiva de mais alto nível, nos projetos de linguagem, a execução de um programa pode ser modelada como uma série de *funções* de transformação destes *estados* [4].

Estas funções podem definir, além do significado de *Programa*, os *Comandos* (salto, blocos, estruturas condicionais, laços de repetição e atribuições) e *Expressões* que são especificadas em uma sintaxe abstrata da linguagem. Segundo Noonan & Tucker (2008), na sintaxe abstrata de uma linguagem de programação, sua estrutura hierárquica é modelada em forma de árvore cuja raiz é o elemento abstrato *Programa*. Desta forma, a definição semântica formal de uma linguagem deve ser iniciada pelo significado de *Programa*, seguida por uma série de funções presentes nesta árvore.

[3] Linguagem de programação amplamente utilizada em problemas de engenharia e computação.

[4] O estado de um programa é o conjunto de todos os objetos (comandos como salto, bloco, condicional, laço e atribuição) ativos e seus valores correntes (Sebesta, 2003).

Diante destas afirmações, apresenta-se a tabela 3.2, um exemplo da semântica dinâmica onde o significado de um programa pode ser definido como um conjunto de funções *S* (significado) de transformação de estado (Sebesta, 2003).

Tabela 3.2: Regras que definem o significado de um programa abstrato.

S: Programa	→	*Estado*
S: Comando x Estado	→	*Estado*
S: Expressão x Estado	→	*Valor*

Fonte: Adaptada de Sebesta (2003).

Na tabela 3.2, os significados dados a programa, comandos e expressões podem ser definidos em três transformações de estados (função de significado *S*): o significado de um *Programa* abstrato é uma função que produz um *Estado* de um programa. Também é visto que o significado de um *Comando* é uma função que, dado um *Estado* atual, produz um novo *Estado*. Já para as expressões, é uma função que, dado um *Estado* atual, produz um certo valor (Sebesta, 2003).

Como pode-se perceber, esta forma de especificar as construções sintáticas, como funções de transformações de estados, é um modelo de alto nível de representação para significados semânticos. Esta abordagem formal não traz, para o projeto da linguagem, informações suficientemente necessárias para sua construção. Para isso, cientistas da computação se utilizam de outros formalismos para representar o significado a programas: as semânticas **Operacional** (seção 3.3), **Axiomática** (seção 3.4) e **Denotacional** (seção 3.5).

Para este trabalho, utilizar-se-á, nas descrições da linguagem *LaND*, a Semântica Denotacional pela sua expressividade, o aspecto da composicionalidade semântica, independência de linguagem artificial, consequentemente de uma arquitetura, e pelo seu formalismo matemático.

3.3 Semântica Operacional

Em contraste com uma semântica que descreve apenas o que um programa faz, o propósito da semântica operacional é descrever *como* uma computação é realizada (Slonneger & Kurtz, 1995).

Nesta abordagem **operacional**, a semântica de um programa é especificada como uma sequência, ou histórico de execução, de transições de estados, geralmente como operações em alguma máquina hipotética abstrata como SECD [5] e VDL [6].

[5] Sigla de *Stack, Environment, Code, Dump*. Uma máquina abstrata, proposta por Peter Landin em 1964, para avaliação mecânica de expressões lambda (Hannan & Miller, 1990)

[6] Sigla de *Viena Definition Language*. Foi a tentativa mais ambiciosa de definir uma máquina abstrata para semântica operacional, desenvolvida no laboratório da IBM em Viena em 1969 (Slonneger & Kurtz, 1995).

Esta abordagem está preocupada em *como o efeito da computação é produzido*. Ou seja, como é executada cada construção sintática da linguagem alvo, não se preocupando com o seu resultado.

A semântica operacional está dividida em duas formas refinadas: a *Semântica Operacional Estruturada*, definida em 1981 com o trabalho de Plotkin (1981) e a *Semântica Natural*, proposta por Kahn (1987).

A semântica natural está preocupada em determinar a ligação entre o estado inicial do programa e seu estado final, não se preocupando com os detalhes intermediários da execução, simplificando sua própria execução. Ao contrário da forma natural, a estruturada está preocupada com a especificação mais detalhada das execuções na linguagem, dando importância a todos os passos envolvidos no processo de execução da linguagem, detalhando melhor os passos exigidos na computação de um programa (Nielson & Nielson, 1992).

Para um entendimento simplificado e preciso sobre a ideia da execução da Semântica Operacional, optou-se por apresentar nas seções 3.3.1 e 3.3.2 uma pequena parte da especificação semântica dos comandos (expressões aritméticas e lógicas) de uma linguagem com uso da Semântica Natural.

3.3.1 A avaliação dos comandos (expressões aritméticas):

A regra para especificar os comandos, como *expressões*, é de avaliar os seus valores associados em um determinado estado. Como apresentado em Winskel (1993), a regra para a execução dos comandos é vista como uma execução de *troca de estados*. A definição semântica de linguagens faz uso de *funções semânticas* que se aplicam sobre as estruturas da linguagem dando-lhes significado. Neste tipo de definição a ideia de *estados* definem os valores às variáveis em um determinado momento. Para o exemplo,

$$< C, s > \rightarrow s'$$

baseada na Semântica Natural, 'C' representa um comando que: iniciado no estado 's', a conclusão de sua execução o leva ao novo estado s'.

Uma notação prática para os *comandos de substituição*, é dada por:

$$< X := 7, s > \rightarrow s[7/X]$$

que significa que a expressão $X := 7$ será avaliada no estado s como sendo a transição que conduz ao estado em que a variável X foi substituída pelo valor 7.

Da mesma forma, pode-se definir um *comando de soma* como sendo,

$$\frac{< a_0, \sigma > \rightarrow n_0 \qquad < a_1, \sigma > \rightarrow n_1}{< a_0 + a_1, \sigma > \rightarrow n} \tag{3.1}$$

onde n é a soma de $n_0 + n_1$.

Um exemplo de aplicação da Equação (3.1) é mostrado na Equação (3.2),

$$\frac{< 2,\sigma_0 > \rightarrow 3 \qquad < 3,\sigma_0 > \rightarrow 4}{< 2+3,\sigma_0 > \rightarrow 7} \tag{3.2}$$

onde o resultado da operação de soma é a derivação da expressão do comando 2 pelo comando 3, resultando em 7.

3.3.2 A avaliação dos comandos (expressões lógicas):

Dando continuidade à especificação das estruturas sintáticas da linguagem, apresenta-se a especificação para as expressões lógicas, com seus valores (**true**, **false**), e dadas nas Equações (3.3) e (3.4). Estas representam as expressões atômicas **true** e **false** (Winskel, 1993):

$$< \text{true}, s > \rightarrow \textbf{true} \tag{3.3}$$

$$< \text{false}, s > \rightarrow \textbf{false} \tag{3.4}$$

Como exemplos de aplicabilidade, a Equação (3.5) avalia a expressão com valor **true** quando n e m são iguais. A Equação (3.6) avalia a expressão e define seu valor como **true** quando n é menor ou igual a m e, finalmente, a Equação (3.7) avalia a expressão com valor **true** quando n não é menor a m.

$$\frac{< a_0,\sigma > \rightarrow n \qquad < a_1,\sigma > \rightarrow m}{< a_0 = a_1,\sigma > \rightarrow true} \tag{3.5}$$

$$\frac{< a_0,\sigma > \rightarrow n \qquad < a_1,\sigma > \rightarrow m}{< a_0 \leq a_1,\sigma > \rightarrow true} \tag{3.6}$$

$$\frac{< a_0,\sigma > \rightarrow n \qquad < a_1,\sigma > \rightarrow m}{< a_0 \geq a_1,\sigma > \rightarrow true} \tag{3.7}$$

Um dos fatores determinantes para a não utilização desta descrição, neste trabalho, é que segundo Sebesta (2003), esta abordagem operacional é fortemente dependente de algoritmos, não de um modelo matemático (como será apresentada na Semântica Denotacional, seção 3.5), onde a sintaxe de uma linguagem é descrita em termos das instruções de um nível mais baixo da própria linguagem, levando à dependência na própria arquitetura (Noonan & Tucker, 2008) além de possíveis circularidades nos seus conceitos.

3.4 Semântica Axiomática

A semântica Axiomática foi desenvolvida por Hoare no final da década de 60 (Hoare, 1969). Seu objetivo inicial era de fornecer uma base formal para a verificação de algoritmos

abstratos e teve, mais tarde, sua aplicação prática na definição de linguagens de programação. Em Hoare & Wirth (1973) é apresentado um exemplo clássico de utilização da Semântica Axiomática foi na especificação da linguagem artificial Pascal[7].

Envolvendo regras para deduzir afirmações sobre a exatidão ou a equivalência dos programas, e suas partes sintáticas (a partir de provas), a semântica axiomática está preocupada em dizer *quais proposições lógicas são válidas* para um programa.

Cada instrução de um programa é *precedida* como *seguida* de uma expressão lógica que especifica restrições a variáveis (Sebesta, 2003). Esta expressão lógica utilizada é o cálculo de predicados.

3.4.1 Avaliação dos comandos (Asserções)

Na abordagem da semântica axiomática as *expressões lógicas* (na forma de predicados) são chamadas de *Asserções*. Estas precedem as instruções de programa, criando regras de restrição (pré-condição) neste ponto do programa. Da mesma forma, uma asserção poderá vir após a expressão, originando uma pós-condição. Esta especificação garante que: dada a especificação de um programa, toda a instrução terá uma pré-condição e uma pós-condição (Sebesta, 2003).

Um exemplo de expressão utilizada para especificar a semântica axiomática, de uma simples forma de instrução é dado,

$$\{P\} \quad S \quad \{Q\} \tag{3.8}$$

onde o símbolo proposicional ‘P‘ é a asserção de pré-condição de execução da instrução ‘S‘ e sua proposição ‘Q‘ é dita uma pós-condição a execução de ‘S‘.

3.4.2 A avaliação dos comandos de substituição

Nesta seção será demonstrada a semântica da avaliação das instruções de substituição de valores à variáveis. Para esta avaliação um axioma é necessariamente definido como regra geral para estas substituições:

$$P = Q_{x \to E} \tag{3.9}$$

.

Este axioma diz que: dado $x = E$ como referência geral para os comandos de substituição, P é a variável que representa a pré-condição e Q a pós-condição. Ou seja, significa que P será computado como Q com todas os valores da instância de x substituídas por E.

Como exemplo de aplicabilidade da Semântica Axiomática, a Equação (3.10) mostra que dada a pós-condição $x > 25$, uma computação é realizada caso uma possível pré-condição $y > 14$

[7] A linguagem de programação Pascal foi originalmente projetada por Niklaus Wirth por volta de 1970 (Canneyt, 2011).

for válida.

$$x = 2 * y - 3 \quad \{x > 25\} \tag{3.10}$$

3.5 A Semântica Denotacional

Enquanto que na abordagem da Semântica Operacional havia o interesse em *como um programa é executado*, na abordagem Denotacional o interesse está no *efeito da execução* de suas estruturas, o *significado* de um programa, bem como seus comandos e expressões. Neste formalismo o efeito da computação interessa mais do que como ela é produzida. A Semântica Denotacional é um método formal [8] que define o *significado* para os comandos ou estruturas que ocorrem na especificação sintática da linguagem. Estes significados são modelados por objetos matemáticos [9] como funções semânticas, definidas composicionalmente, de transformações de estados [10]. Neste caso, o significado de um programa pode ser expresso como um conjunto destas funções que atuam sobre o estado do programa, representando o efeito de executar suas estruturas (Nielson & Nielson, 1992).

Originalmente a Semântica Denotacional foi desenvolvida por Christopher Strachey em meados dos anos 60 no grupo de pesquisa em Programação na Universidade de Oxford. Em 1969 Dana Scott deu sua contribuição com a fundamentação matemática à linguagem. Embora originalmente concebido como um mecanismo para a análise de linguagens de programação, a Semântica Denotacional tornou-se uma poderosa ferramenta para *design* de linguagens artificiais (Slonneger & Kurtz, 1995).

Em Scott & Strachey (1971), os autores justificam a importância de se ter uma semântica matemática na definição das linguagens de programação, dando significado as suas construções, independente de sua implementação.

Em Tennent (1976), houve uma grande contribuição para o entendimento da aplicação no campo teórico envolvido na Semântica Denotacional. Neste trabalho é apresentado os conceitos semânticos de *environments*, *stores*, e *continuations*.

No trabalho de Scott (1982) é apresentado a teoria dos domínios, base da semântica denotacional. Seu objetivo foi dar um modelo matemático aos tipos de dados, explicando as noções de computabilidade, como exemplo, aos tipos de dados em linguagens artificiais. Segundo a teoria dos domínios, os tipos são categorizados como um conjunto de elementos finitos de mesma propriedade. Ou seja, neste formalismo assume-se que os programas representam funções computáveis ao qual serão mapedos em elementos correspondentes de um domínio. Esta

[8] Métodos formais são técnicas baseadas em formalismos matemáticos para a especificação, desenvolvimento e verificação dos sistemas de softwares e hardwares (NASA, 2001).

[9] Por exemplo, o significado de um programa pode ser dado por uma função que possui entrada e o resultado da sua computação como saída (Zhang & Xu, 2004)

[10] São as etapas individuais que ocorrem durante a execução de um programa (Noonan & Tucker, 2008)

característica faz com que a especificação semântica de uma linguagem de programação mapeie suas funções (programas com seus elementos sintáticos) em funções que a denotem (Allison, 1986).

Basicamente, a Semântica Denotacional foi escrita em λ-notação que é o cálculo λ de Church com tipos de dados (Allison, 1986). Este método é conciso e poderoso o suficiente para descrever as características das linguagens de programação atuais [11]. Uma das características deste formalismo é quanto à sua legibilidade. Suas definições formais são legíveis com a prática, sendo a notação equivalente a uma linguagem de programação poderosa, mas difícil de ler. Por este fato, é mais adequada à projetistas e implementadores de linguagens que ao programador.

Sintaticamente, o significado dado a um programa é feito em termos de suas partes definidas em sua BNF que, seguindo os princípios da composicionalidade, todas as partes denotadas da linguagem (subfrases) darão o completo significado da linguagem (Slonneger & Kurtz, 1995).

Dando continuidade a sintaxe da Semântica Denotacional, para as definições denotacionais utiliza-se um símbolo especial "⟦ ⟧" que separa a parte sintática da parte semântica da especificação. Um pequeno exemplo desta separação pode ser visto em Slonneger & Kurtz (1995), onde K é uma expressão sintática em uma linguagem de programação, e sua especificação denotacional irá definir um mapeamento que dará seu significado, de modo que o significado de ⟦K⟧ é a denotação deste K. Ou seja, uma entidade abstrata que modela a semântica de K.

Seguindo com o exemplo, as expressões "3*5", "(10+5)", "0015" e "15" são frases sintáticas que denotam o mesmo objeto abstrato, o número inteiro 15. Portanto, uma definição denotacional destas expressões deve ser capaz de mostrar que,

$$significado⟦3*5⟧ = significado⟦10+5⟧ = significado⟦0015⟧ = significado⟦15⟧ = 15$$

Uma especificação denotacional consiste de 5 componentes fundamentais. Duas que especificam a parte sintática da linguagem: **domínio sintático** e **sintaxe abstrata** (ou regras de produção abstrata). Uma com o **domínio semântico**, especificando a parte semântica. E duas que definem as funções que fazem o mapeamento dos objetos sintáticos para os objetos semânticos: **funções semânticas** e **equações semânticas**. Esta última responsável em dar significado as partes que constituem a linguagem.

A tabela 3.3 contém uma especificação denotacional de uma simples linguagem de números inteiros (não negativos). Esta definição requer duas funções auxiliares que foram definidas no mundo semântico, onde a relação $Número$ x $Número$ denota um produto cartesiano como abaixo,

$$plus : Número \text{ x } Número \rightarrow Número$$
$$times : Número \text{ x } Número \rightarrow Número$$

[11] Combinado a outros formalismos, como CSP, este poderá especificar as singularidades inerentes ao paradigma paralelo, como exemplo a concorrência (Roscoe, 2005).

Tabela 3.3: Especificação semântica de uma linguagem de números inteiros.

Domínio Sintático

N : Numeral
D : Digito

Regras de Produção Abstratas

Numeral ::= Digito | Numeral
Dígito ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Domínio Semântico

Número = 0, 1, 2, 3, 4, 5,...

Funções Semânticas

$valor$: Numeral → Número
$dígito$: Dígito → Número

Equações Semânticas

$valor[\![ND]\!]$ = $plus(times(10, valor[\![N]\!]), dígito[\![D]\!])$
$valor[\![D]\!]$ = $dígito[\![D]\!]$
$digito[\![0]\!]$ = 0;
$digito[\![1]\!]$ = 1;
$digito[\![2]\!]$ = 2;
$digito[\![3]\!]$ = 3;
$digito[\![4]\!]$ = 4;
$digito[\![5]\!]$ = 5;
$digito[\![6]\!]$ = 6;
$digito[\![7]\!]$ = 7;
$digito[\![8]\!]$ = 8;
$digito[\![9]\!]$ = 9;

Fonte: Adaptada de Slonneger & Kurtz (1995).

Como apresentado em Slonneger & Kurtz (1995), e com base na tabela 3.3, apresenta-se um exemplo do processo de derivação de um valor para o numeral 65, seguindo as definições denotacionais:

$$\begin{aligned} value[\![65]\!] &= plus(times(10, value[\![6]\!]), digit[\![5]\!]) \\ &= plus(times(10, digit[\![6]\!]), 5) \\ &= plus(times(10, 6)\ 5) \\ &= plus(60, 5) = 65 \end{aligned}$$

Para entender o processo na construção da especificação para uma linguagem de programação utilizando a Semântica Denotacional, será apresentado nas próxima seções seus 5 componentes fundamentais: **domínio sintático** (3.5.1), **sintaxe abstrata** (3.5.2), **domínio semântico** (3.5.3), **funções semânticas** (3.5.4) e **equações semânticas** (3.5.5).

3.5.1 Domínio Sintático

No processo de especificação é necessário que se façam definições para as entidades básicas da linguagem (presentes na sua BNF, por exemplo) e uma função que mapeie instâncias destas entidades em instâncias de objetos matemáticos. Este último será discutido na seção 3.5.4.

Segundo Sebesta (2010), as funções de mapeamento da Semântica Denotacional se comportam de forma similar às funções na matemática. Neste caso, o domínio, chamado de **Domínio Sintático**, é uma coleção de valores para os parâmetros das funções e a imagem, chamada de Domínio Semântico, é a coleção de objetos para os quais os parâmetros são mapeados.

No processo que define o domínio sintático, apresenta-se a tabela 3.4, onde os símbolos do lado esquerdo são metavariáveis sobre um conjunto de elementos básicos de linguagem como: expressões, números, comandos e identificadores de variáveis. Classicamente, estas são denotadas por símbolos com três ou mais letras em negrito.

Tabela 3.4: Especificação dos domínios sintáticos.

I	:	**Ide**	Identificadores
E	:	**Exp**	Expressões
C	:	**Com**	Comandos
Φ	:	**Abs**	Abstrações
II	:	**Par**	Parâmetros
ψ	:	**Prog**	Programa

Fonte: Autor, 2013.

Uma outra forma de apresentar o domínio sintático é na forma:

Tabela 3.5: Outra forma de representação de domínios sintáticos.

I	$\in$	**Ide**	Identificadores
E	$\in$	**Exp**	Expressões
C	$\in$	**Com**	Comandos
Φ	$\in$	**Abs**	Abstrações
II	$\in$	**Par**	Parâmetros
ψ	$\in$	**Prog**	Programa

Fonte: Autor, 2013.

Caso seja necessário utilizar mais de uma metavariável do mesmo tipo, a exemplo de "I", devemos indexá-las como I_1, I_2, I_3, ...

3.5.2 Sintaxe Abstrata ou Regras de Produção Abstrata

Segundo Tennent (1976), para definir a sintaxe da linguagem como o domínio de uma interpretação semântica, faz-se necessário ignorar alguns complicadores que são semanticamente irrelevantes como: o grau de procedência sobre os operadores numéricos "+" e "∗". De fato, a Semântica Denotacional não está interessada na forma das construções sintáticas (estruturas válidas das sentenças) e sim no significado destas sentenças. Neste caso, esta representação será dada por uma sintaxe *abstrata* que abstrai a complexidade na ordem de execução das operações em uma expressão. Uma síntese desta regra de produção abstrata é vista na tabela 3.6.

Tabela 3.6: Regras de produção abstratas sem restrições à precedência operacional.

ψ	::=	E
II	::=	I
		| II_1, II_2, II_3, II_4,..., I_n
		| Λ
⋮		
E	::=	B
	|	I
	|	Φ
	|	E_1 "+" E_2
	|	E_1 "-" E_2
	|	E_1 "*" E_2
	|	E_1 "/" E_2
	|	**if** E_0 **then** E_1 **else** E_2
	|	E_1 **and** E_2
	|	E_1 **or** E_2
	|	**case** E_0 **of** E_1, E_2,..., E_n
	|	$E_1 = E_2$, E_2,..., E_n
⋮		

Fonte: Adaptada de Tennent (1976).

3.5.3 Domínios Semânticos

Domínios semânticos podem ser definidos como coleções, ou conjunto, de objetos matemáticos que expressam determinados significados para as construções sintáticas da linguagem.

A especificação, tabela 3.7, apresenta seis domínios semânticos: o primeiro, $\tau : T$, forma o conjunto dos booleanos, o segundo $\nu : N$ é formado pelo conjunto dos números naturais, o terceiro $\mu : R$ é formado pelos números reais, este será o domínio das expressões aritméticas. O quarto domínio, $\eta : H$, representa o conjunto dos caracteres, o quinto representa o conjunto dos

Tabela 3.7: Exemplos de Domínios Semânticos.

τ: T	=	$\{true, false\}$	Valores verdade (booleanos)
ν : N	=	$\{\ldots,-1,-1,0,1,2,\ldots\}$	Inteiros
μ : R	=	$\mathbb{R}$	Conjunto dos Reais
η: H	=	{“a”, ”b”, “c”,...}	Caracteres
υ: V	=	{T+N+H}	Valores básicos
ζ: CSP	=	{Conjunto de processos CSP}	Processos em ambiente CSP

Fonte: Adaptada de Almeida & Haeusler (1994).

valores básicos da linguagem e, finalmente, o sexto um domínio dos processos em uma máquina abstrata CSP (*Communicating Sequential Processes*) que descreve padrões de interação em processos concorrentes (Hoare, 2004), mais detalhes serão apresentados no capítulo 4.

3.5.4 Funções Semânticas

Como descrito anteriormente, na semântica denotacional, o significado de cada frase é dado por meio de uma entidade matemática a qual *denota* um significado. Então, a semântica de uma linguagem é dada por meio de *funções* (**funções semânticas**) que *mapeiam* essas frases presentes no **domínio sintático** para suas denotações do **domínio semântico**.

Como exemplo de funções semânticas, apresenta-se a tabela 3.8 com três funções que darão significado as expressões, comandos e programa em uma linguagem:

Tabela 3.8: Exemplos de Funções Semânticas.

E : Exp	$\rightarrow$	(S $\rightarrow$ N)
M : Prog	$\rightarrow$	(N $\rightarrow$ N)
C : Cmd	$\rightarrow$	(S $\rightarrow$ S)

Fonte: Adaptada de Tennent (1976).

A primeira função de exemplo, apresenta no seu lado esquerdo uma letra (“**E**”) usada para designar uma função de interpretação semântica. Nesta interpretação o domínio é o elemento sintático (“**Exp**”), presente no **domínio sintático** da linguagem. No lado direito tem-se a sua imagem, representada por **domínios semânticos** da linguagem. Ou seja, o significado de uma expressão (“**Exp**”) é uma função que, quando aplicada ao **estado** (“**S**”), produz valores numéricos (“**N**”) da expressão relativa aos estados.

Segundo Tennent (1976), um momento do estado σ define uma relação entre uma variável e um valor numérico específico, para o seu estado atual. Desta forma, torna-se conveniente

$$\sigma\text{:S} \quad = \quad \text{Var} \rightarrow \text{N} \quad \text{(Estados)}$$

modelar estados por funções com domínio ("**Var**") e co-domínio ("**N**") como:

A segunda função apresentada dá significado a programa. Neste exemplo, um programa que recebe valores numéricos como entrada e retorna também valores numéricos. A função de interpretação ("**M**") tem em seu domínio sintático o identificador ("**Prog**"), e no domínio semântico um par de valores numéricos (entrada e saída), dados por $Prog \rightarrow (N \rightarrow N)$.

A terceira função, dos comandos da linguagem. é uma função que significa a transição de estados $Cmd \rightarrow (S \rightarrow S)$.

3.5.5 Equações Semânticas

Como apresentado, as funções semânticas são responsáveis pelo mapeamento entre o domínio sintático e o domínio semântico e que, em geral, temos uma função semântica para cada categoria sintática da linguagem.

Uma vez definida as funções semânticas, resta agora dar significado matemático das construções sintáticas da linguagem através das equações semânticas.

Um exemplo de equação semântica foi apresentada na tabela 3.3 onde, do lado esquerdo desta equação semântica apresenta-se uma função semântica a ser definida. O argumento desta função será uma expressão contina na sintaxe abstrata sobre as metavariáveis do domínio sintático, envolvidos pelos delimitadores especiais "$[\![$" e "$]\!]$".

Geralmente, algumas categorias sintáticas são especificadas como são os casos das expressões e o conceito de estados de memória em uma certa linguagem.

Equação semântica para expressões:

Uma equação semântica de uma *expressão* pode ser escrita na forma $E[\![a]\!]\sigma = \ldots$, onde "$E$" é a função semântica das expressões e "a" é uma expressão composta de elementos sintáticos da linguagem. Esta equação possui a seguinte leitura "*a denotação da expressão 'a' é a função que, dada um estado de memória 'σ' tem um valor...*".

Equação semântica dos estados de memória:

Para a noção de *estado*, pode-se recuperar o valor de uma variável, por exemplo "x", escrevendo a equação $\sigma[\![x]\!]$ que denota a função que recupera o valor da variável "x" do estado corrente "σ" (Noonan & Tucker, 2008).

Para o conceito de transformação de estados, ou atualização de estado, pode-se utilizar duas notações. A primeira, $\sigma[\![v/x]\!]$, onde a *variável* "x" possui o *valor* "v" estando no *estado* "σ". A segunda notação, $\sigma'[\![x]\!] = v$, diz que a *variável* "x" possui o *valor* "v" estando no novo *estado* "σ".

Apesar das formas semânticas apresentadas, outros formalismos podem ser citados (não discutidos neste trabalho), são os casos da Semântica de Ações e Game Semantics. Maiores detalhes destes formalismos podem ser vistos em Mosses (1996) e Japaridze (2005).

Neste capítulo foram apresentados os conceitos básicos que permeiam o formalismo de especificação sintática e principalmente semântica de linguagem de programação. Foram discutidos os formalismos dos paradigmas Axiomáticos, Operacional e, principalmente, Denotacional.

No próximo capítulo tratar-se-á de CSP (*Communicating sequential processes*), uma linguagem formal que especifica padrões de interação em modelos de sistemas concorrentes.

4 COMMUNICATING SEQUENTIAL PROCESSES - CSP

ESTE capítulo introduz aos conceitos básicos da linguagem CSP *Communicating Sequential Processes*, um formalismo que será utilizado como complemento à especificação denotacional de *LaND*.

Seu formalismo característico, relacionado aos padrões de comunicação entre processos, foi determinante para sua adoção para expressar as formas de aproximação numérica apresentadas nas equações hiperbólicas, elípticas e parabólicas.

4.1 Introdução a CSP

Communicating Sequential Processes (CSP) é um modelo algébrico de linguagem formal para descrever padrões de interação entre processos ou programas concorrentes (Roscoe, 2005).

Basicamente, entender o modelo conceitual da CSP é abstrair um ambiente computacional como uma composição de entidades, com comportamentos independentes, que comunicam entre si e/ou com o ambiente externo (outro processo) através de suas interfaces. Segundo Hoare (2004), esta interface é descrita como um conjunto de eventos que representam um tipo específico de ação a ser executado ou sofrido pelo processo. Esta característica permite que pequenos processos possam ser combinados para formar processos maiores e/ou mais complexos, sem alterar as estruturas internas das partes constituintes (composicionalidade de processos).

O primeiro trabalho em CSP foi publicado em Hoare (1978), e deste então está em constante evolução como CSP_σ, uma extensão à semântica operacional da CSP (Colvin & Hayes, 2011).

Neste trabalho a linguagem CSP será explorada como ferramenta de auxílio às interpretações semânticas da Semântica Denotacional, dando significado aos modelos de comunicação inerentes aos métodos numéricos a serem executados.

Como apresentado, processos CSP exprimem algum comportamento que são construídos através de eventos, operadores e também por outros processos.

Dada esta breve introdução, este capítulo apresentará os conceitos fundamentais do formalismos da linguagem CSP.

4.2 Elementos da linguagem

Para uma fácil compreensão dos elementos sintáticos da CSP, seleciona-se um subconjunto algébrico (operadores) que é suficiente e expressivo para ilustrar este formalismo. São eles: prefixo, composição sequencial, escolha interna, ecolha externa e paralelimos (ver tabela 4.1). Estes conceitos serão apresentados na seção 4.2.1.

Um conceito inicial a ser compreendido é a ideia de *eventos* CSP. *Eventos* são ocorrências de acontecimentos instantâneos ou uma ação atômica sem duração (Hoare, 2004). Na forma mais tradicional da CSP a capacidade de representar "processos com tempo de execução" não é possível. Para estes casos, extensões CSP foram implementadas e podem ser vistas em Reed & Roscoe (1988) e Fischer (n.d.).

4.2.1 Processos Primitivos CSP

Os relacionamentos entre processos diferentes e a forma como estes realizam a comunicação são descritos utilizando operadores algébricos que, de forma composicional, transforma processos complexos em processos mais simples a partir de tipos primitivos. Um processo primitivo pode ser compreendido como uma representação de um comportamento básico. Existem dois tipos primitivos: *STOP* e *SKIP*.

$$Processo_p ::= \textbf{STOP} \mid \textbf{SKIP}$$

O primitivo *STOP* representa o processo que teve uma conclusão anômala, a exemplo de uma situação de *deadlock*. De forma inversa, *SKIP* representa a conclusão satisfatória do processo.

4.2.2 Operadores Algébricos

Além dos operadores primitivos, CSP possui um conjunto de operadores algébricos, como apresentados na tabela 4.1.

Tabela 4.1: Operadores algébricos CSP.

Função		Equação	Significado
$Process_p$	::=	STOP	
	|	SKIP	
	|	$a \rightarrow Process_p$	<Prefixo>
	|	$Process_p \ ; \ Process_p$	<Composição sequencial>
	|	$Process_p \ \Box \ Process_p$	<Escolha externa>
	|	$Process_p \ \sqcap \ Process_p$	<Escolha interna>
	|	$Process_p \ \| \| \ Process_p$	<Paralelismo>

Fonte: Adaptada de Hoare (2004).

Os operadores apresentados na tabela 4.1 representam um subconjunto CSP, suficiente para este trabalho. Seus detalhes serão apresentados a seguir.

Prefixo

O operador de *prefixo* é representado pelo símbolo →. Sua sintaxe é posta na forma <evento> → <processo>, ou seja, um evento no seu lado esquerdo e um processo no seu lado direito. No exemplo do processo,

$$x \rightarrow P,$$

na ocorrência do evento x, este processo comportar-se-á como o processo P.
Com o uso do prefixo e da recursão pode-se representar uma sequência de eventos para processos infinitos. No exemplo apresentado em Hoare (2004):

$$CLOCK = (tick \rightarrow CLOCK),$$

o processo *CLOCK* executa o evento *tick* que volta a se comportar como o processo *CLOCK*.

Outro exemplo que demonstra a recursão, visto em Roscoe (2005):

$$P1 = (up \rightarrow down \rightarrow P1),$$

representa um processo de $P1$ que dado a ocorrência do evento up, em seguida o evento $down$, terá seu comportamento como $P1$.

Composição sequencial

Representado pelo símbolo ";", este operador permite que os processos sejam executados segundo uma ordem. Por exemplo,

$$A \ ; \ B$$

inicialmente o processo comporta-se como "A" e, após a conclusão de sua execução comporta-se como "B".

Escolha externa

O operador *escolha externa*, também conhecido como *escolha determinística*, é representado pelo símbolo $\Box$, conforme exemplo abaixo:

$$(a \rightarrow P) \ \Box \ (a \rightarrow Q),$$

onde uma entidade externa deverá optar por qual processo será executado. Neste exemplo, dada ocorrência do evento "a", este processo terá seu comportamento de P, caso opte pelo primeiro caso, ou Q, o outro caso.

Escolha interna

Na *escolha interna* ou *escolha não-determinística*, representado por $\sqcap$, onde o exemplo

$$(a \rightarrow P) \ \sqcap \ (a \rightarrow Q),$$

diz que, na ocorrência do evento "a" o ambiente (entidade interna) irá escolher de forma aleatória se o processo comportar-se-á como o P ou Q.

Paralelismo

Em CSP processos podem ser executados de forma paralela e sua sintaxe é representada pelo símbolo $\|$. Como exemplo,

$$(P \parallel Q) \ ; \ R$$

os processos P e Q terão execuções em paralelo. Após a conclusão de ambos, quando P e Q se comportam como $SKIP$, é que se dará a execução do processo R.

4.2.3 Canais de Comunicação

Os eventos da comunicação entre processos são realizados através de canais de comunicação. Segundo Colvin & Hayes (2011), um evento de um processo $ch!e$, indica a saída do valor da expressão (uma variável) "e" através do canal "ch". Uma outra forma é "$ch?y$", indicando que um processo recebe um valor do canal "ch" e armazena-o na variável "y". Como exemplo de uso de canal de comunicação,

$$P = IN!a \rightarrow IN?a \rightarrow P$$

diz que o processo "P" tem seu estado atual como o resultado da comunicação do valor da variável "a" através do canal "IN".

Estas características apresentadas por CSP, com seus padrões de comunicação entre processos, foram determinantes para sua adoção neste trabalho. Como uma linguagem formal, sua função será de expressar as formas de aproximação numérica presentes na equação hiperbólica de uma EDP. Mais detalhes desta expressividade para os modelos de aproximação, bem como a aplicabilidade dos conceitos aqui apresentados, serão vistos na seção 5.2.4.

O presente capítulo elencou os conceitos fundamentais do formalismo algébrico *Communicating Sequential Processes* (CSP) que descreve, através de padrões de interação, a comunicação entre processos. O objetivo deste capítulo foi apresentar como a expressividade de CSP o habilita a formalismo responsável em descrever as formas de aproximação numérica presentes na equação hiperbólica de uma EDP.
No próximo capítulo será discutida a especificação semântica de *LaND*.

5 *LaND* - LANGUAGE OF NUMERICAL DISCRETIZATION

NESTE capítulo será apresentada a especificação semântica de ***LaND*** - ***La****nguage of* ***N****umerical* ***D****iscretization*. Esta é uma proposta inicial do modelo ao qual o foco será na resolução de problemas com equações hiperbólicas, com acurácia em segunda ordem no espaço (diferença central) e primeira ordem no tempo (diferença para frente), onde a malha computacional numérica apresenta-se com espaços geometricamente uniformes.

A principal característica desta linguagem é na sua capacidade em abstrair a complexidade no desenvolvimento de algoritmos para resolução de problemas das Equações Diferenciais Parciais com o Método das Diferenças Finitas, nas equações hiperbólicas. No desenvolvimento desta linguagem se fez necessário dar significados às expressões, instruções e unidades básicas da linguagem. O objetivo foi criar um formalismo que possa ser usado por geradores automáticos de programas a partir destas descrições semânticas apresentadas. Neste trabalho, a comunicação entre os nós presentes na malha computacional foram definidos como processos, incluindo o sincronismo, e que foram modelados com base na semântica formal de CSP.

A especificação apresentada define o projeto de uma linguagem com as seguintes características:

- É Uma linguagem funcional, baseado na sintaxe de Haskell (Sá & Silva, 2006);
- Existem funções pré-definidas para as operações dos métodos numéricos: o tipo do problema, regras iniciais (para as condições iniciais e condições de contorno) e forma de comunicação para os pontos da malha computacional;

Seguindo as práticas adotadas no processo de especificação semântica de linguagens, faz-se necessária uma modelagem das partes sintáticas e semânticas da linguagem, para isto odota-se a semântica denotacional, apresentada na seção 3.5. A proposta é que o resultado desta especificação semântica seja um conjunto de funções que mapeiam as categorias sintáticas da linguagem em domínios semânticos, de forma a traduzir e dar significados as suas construções.

5.1 Especificação Sintática

Como apresentado na seção 3.5, uma especificação em semântica denotacional constitui de significados que são dados às construções sintáticas da linguagem. Por este motivo foram definidos, sintaticamente, os constituintes fundamentais da linguagem na forma dos seus domínios sintáticos e regras de produção abstratas, como seguem.

5.1.1 Domínios Sintáticos

Inicialmente, define-se os elementos básicos da linguagem que determinam seus identificadores e áreas de declarações para funções pré-definidas.

A tabela 5.1 apresenta alguns destes elementos, onde sua representação foi mapeada em metalinguagem na forma de símbolos que serão utilizados posteriormente (seções 5.3 e 5.4).

Tabela 5.1: Domínio sintático.

Símbolo	Domínio	Descrição
Υ :	**Prog**	Programas;
Φ :	**Module**	Definição de módulo;
I :	**Ide**	Identificadores (variáveis ou funções);
A :	**Arou**	Área de declarações dos valores das condições de contorno;
B :	**Bline**	Área de declarações dos valores das condições iniciais;
C :	**Decl**	Área de declarações dos valores para os intervalos de tempo e espaço;
Γ :	**Gamma**	Área de declaração do domínio espacial e temporal da simulação (malha computacional);
Δ :	**Delta**	Especificação do método utilizado para resolver numericamente a equação (*stencil*).

Fonte: Autor, 2013.

Assim como em Haskell (Sá & Silva, 2006), tem-se aqui o domínio sintático de um identificador ***Module*** para representar os conceitos de subprograma. Ou seja, os programas gerados podem ser modularizados para garantir a legibilidade e coesão do artefato.

5.1.2 Regras de Produção Abstratas

As regras de produção abstratas, ou sintaxe abstrata como é conhecida, definem as estruturas da linguagem de programação em um nível mais funcional, com pouco rigor sintático. De forma simples, a sintaxe abstrata forma uma base para conectar as fases de análise sintática e semântica da linguagem, processo necessário na especificação com a semântica denotacional. Além desta característica, as regras de produção abstratas expressam as informações básicas necessárias para um processo de derivação sintática da linguagem aqui estudada. A tabela 5.2, apresenta as regras de produção *abstratas* da linguagem.

Tabela 5.2: Regras de produção abstratas.

Símbolo		Regras de Produção
Υ	::=	$(I \rightarrow \Phi)$
Φ	::=	Module I where [Γ] [C] [Δ] [A] [B]
INT	::=	$< num >< INT > \mid < num >$
I	::=	$< chact >< CHR > \mid < chact >$
Γ	::=	$size(< \Gamma_s >; < \Gamma_t >)$
Γ_s	::=	$< INT >$
Γ_t	::=	$< INT >$
C	::=	$slot(< C_s >; < C_t >)$
C_s	::=	$< INT >$
C_t	::=	$< INT >$
Δ	::=	$WE\mid$ [demais métodos (*stencil*)]
A	::=	$WE_{Edge}\mid$ [demais condições de contorno]
B	::=	$WE_{Start}\mid$ [demais condições iniciais]

Fonte: Autor, 2013.

Estas regras foram definidas considerando o domínio sintático, definido anteriormente na tabela 5.1 onde a metalinguagem Φ é composta, na ordem, por uma palavra reservada *Module*, um identificador *I*, uma palavra reservada *Where* e os seguintes símbolos:

Γ - a área de definição da malha computacional para o problema a ser simulado. Por exemplo, para os problemas hiperbólicos, como é o caso da equação da onda, o usuário informa duas constantes numéricas: uma que especifica o tamanho da corda Γ_s e, a outra, o tempo de simulação Γ_t. Como percebemos, esta área é definida com uma função $size()$ contendo estes dois parâmetros. Esta função simula o λ presente nas equações da velocidade inicial, Equação (2.20), e do processamento dos valores internos da malha, Equação (2.19).

C - a área que possui as mesmas funções de Γ, ao passo que, esta é definida por C_s e C_t: número de pontos no espaço e no tempo, respectivamente.

Δ - a área onde o usuário definirá o método de aproximação numérica. Sua derivação resulta em uma das formas de aproximação numérica apresentadas neste trabalho.

A - a área de declaração para as condições de contorno do problema;

B - área de declaração das condições iniciais do problema.

5.2 Especificação Semântica

Para a especificação semântica de *LaND* foi definido o domínio semântico das suas unidades básicas. Também foi definido o domínio das entidades que representarão a definição da malha computacional, a memória alocada e as formas de aproximação numérica.

Como este trabalho está focado em demonstrar a especificação formal em problemas hiperbólicos (equação da onda) e com acurácia em segunda ordem no espaço (diferença central) e primeira ordem no tempo (diferença para frente), foi possível especificar outros modelos de acurácia. Estes modelos estão apresentados, utilizando a notação CSP, no apêndice B.

5.2.1 Domínio Semântico das Unidades Básicas

Como em qualquer linguagem de programação é necessário determinar seu conjunto de elementos básicos que definem os aspectos elementares da linguagem. Este é o caso dos valores associados às constantes do tipo: numérico, caracteres e subconjunto de numéricos[1].

Tabela 5.3: Domínio semântico da unidades básicas de *LaND*.

Metalinguagem		Domínio
INT	=	$\{\ldots,-3,-2,-1,0,1,2,3,\ldots\}$
CHR	=	$\{a,b,c,d,\ldots,z,espaço,0,1,2,\ldots,9\}$
VAL	=	$\mathbb{N}^*$

Fonte: Autor, 2013.

Onde, a metavariável INT representa o domínio dos números naturais. CHR é o domínio dos caracteres do alfabeto e números inteiros, e VAL dos naturais positivos.

5.2.2 Domínio semântico da definição da malha computacional

Especificamos quatro aspectos básicos que definem as características da malha computacional: número de pontos no espaço e no tempo, bem como os valores dos seus intervalos no espaço e no tempo. A tabela 5.4 apresenta estes domínios, onde: $Interval_{space}$ representa um domínio para um número discreto de intervalos em que dividimos o domínio espacial e $Interval_{time}$ um domínio para o número de passos de tempo de simulação; $Size_{space}$ e $Size_{time}$ são domínios que definem os valores dos intervalos espaciais, h, e de tempo, k, conforme a Equação (2.16). Estes aspectos definem a escala real da malha usada na simulação. Em todos estes casos estes domínios são representados por constantes inteiras e positivas.

5.2.3 Domínio Semântico dos valores armazenáveis em memória

Para representar os valores associados às variáveis na memória, define-se este domínio como um resultado dos processos de comunicação entre os nós computáveis da malha computacional. A entidade semântica *TValues*, e suas variantes *Wave Equation* (we) e *Other Equation* (oe), são apresentadas na tabela 5.5 e representam valores na memória de uma matriz computacional. Ou

[1] Por exemplo, o subconjunto $\mathbb{N}^*$ dos números naturais.

Tabela 5.4: Domínio semântico das características da malha computacional.

Metalinguagem		Domínio
$Interval_{Space}$	=	INT ∈ VAL
$Interval_{Time}$	=	INT ∈ VAL
$Size_{Space}$	=	INT ∈ VAL
$Size_{Time}$	=	INT ∈ VAL

Fonte: Autor, 2013.

seja, no processo de definição desta linguagem, os valores atribuídos as posições de memória podem ser vistos como processos individuais e construídos na forma de processos CSP. Neste caso, como apresentado na tabela 5.5, define-se inicialmente o domínio semântico para a localização destes endereços de memória *LOC* com um conjunto de valores reais e *TValues*, como o conjunto de processos CSP associados a estas posições.

Tabela 5.5: Domínio dos valores armazenáveis em memória.

Metalinguagem		Domínio
LOC	=	$\mathbb{R}$
$TValues_{(we)}$	=	$ProcessosCSP$ * LOC $\rightarrow \mathbb{R}$
$TValues_{(oe)}$	=	$ProcessosCSP$ * LOC $\rightarrow \mathbb{R}$

Fonte: Autor, 2013.

Para a especificação semântica dos processos envolvidos na geração dos valores da malha computacional, optou-se por defini-la em termos de CSP. Um trabalho que demonstra com sucesso a especificação do comportamento de programa com uso do formalismo semântico é mostrado em Almeida & Haeusler (1994), onde os autores utilizam a semântica denotacional associada ao formalismo semântico da CSP para descrever o comportamento das trocas de mensagens em um programa no paradigma orientado a objetos.

Na especificação do domínio semântico das equações que tratam a fenomenologia aqui estudada, o modelo de aproximação numérica (acurácia no espaço e tempo) podem ser vistos como *stencil* de comunicação entre os nós computáveis da malha computacional. Para este trabalho, fez-se nescessário representar formalmente estas dependências na forma de processos únicos em CSP.

Esta especificação será demonstrada com auxilio de grafos e CSP e será detalhada conforme as regras de produção e sintaxe da linguagem. Neste trabalho inicial foi utilizado, como modelo, a equação de onda (WE), um caso particular de EDP linear de segunda ordem [2]. Também considera-se apenas o caso da aproximação por diferença central (CD) em x e diferença para

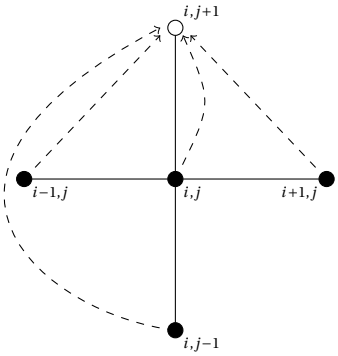
[2]Uma EDP linear de segunda ordem em x e t pode ser escrita como $a_1\frac{\partial^2 u}{\partial x^2} + a_2\frac{\partial^2 u}{\partial t^2} + a_3\frac{\partial^2 u}{\partial x \partial t} + a_4\frac{\partial u}{\partial x} + a_5\frac{\partial u}{\partial t} + a_6 u + g = 0$, onde $g = g(x,t)$ pode ser uma função de x e t. Por simplicidade, no caso da WE do exemplo, $a_1 = -c^2$ e $a_2 = 1$, os demais coeficientes são nulos e $g(x,t) = 0$.

frente (FD) em t, ambas com acurácia de segunda ordem. Conforme mostrado no *stencil* da Figura 5.1. Por *OE* esta-se abstraindo a generalização da EDP linear, através da entrada dos seus coeficientes lineares, do método de diferenças finitas (diferença central, para frente ou para trás, entre outros) e da acurácia, estas duas últimas que definem os *stencils* utilizados para a aproximação das derivadas em qualquer ordem.

5.2.4 Domínio semântico da aproximação para a equação da onda

Dado o domínio sintático, para cada escolha da metalinguagem Δ apresentamos um grafo de dependência associado. Neste exemplo assume-se que o usuário queira resolver a equação de onda por diferença central em x e diferença para frente em t, CDFD, ambas com segunda ordem de acurácia, ou simplesmente o identificador $\Delta = WE$, a sua representação do domínio semântico será dada baseada nas regras de comunicação dos nós presentes na computação, como visto na Figura 5.1. Esta regra tem como base o fluxo da dependência onde o nó rotulado $(i, j+1)$ depende da execução dos nós rotulados $(i-1, j)$, (i, j), $(i+1, j)$ e $(i, j-1)$.

Figura 5.1: Grafo da *Equação da Onda*.



Fonte: Autor, 2013.

Como pré-requisito da especificação do domínio semântico desta dependência presente no modelo de aproximação numérica, mapea-se cada nó, presente na computação, como processos CSP. Esta abordagem auxiliará nas interpretações semânticas associadas às unidades CSP na forma de canal de comunicação. O resultado deste mapeamento pode ser visto na tabela 5.6.

A Fig. 5.2 apresenta as dependências de comunicação entre nós e arestas, em termos de processos CSP, onde seus canais IN_1, IN_2, IN_3 e IN_4 fazem ligações entre os processos individuais P_{CSP_1}, P_{CSP_2}, P_{CSP_3} e P_{CSP_4}.

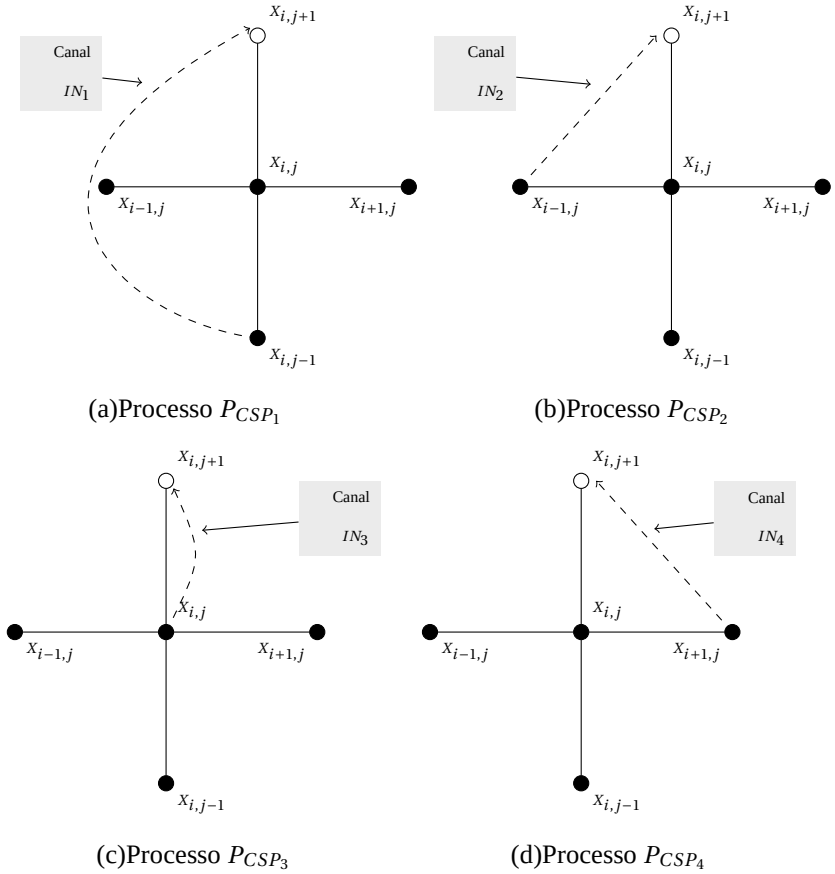
Na Figura 5.2 apresenta-se o refinamento do grafo de dependência em grafos com processos únicos. Como visto nas Figuras 5.3(a), 5.3(b), 5.3(c) e 5.3(d), o fluxo da dependência foi in-

Tabela 5.6: Mapeamento dos processos para a *Equação da Onda* (we).

Nós		Processos
$(i, j+1)$	$\rightarrow$	$P_{(x_{(i,j+1)})}$
$(i-1, j)$	$\rightarrow$	$P_{(x_{(i-1,j)})}$
(i, j)	$\rightarrow$	$P_{(x_{(i,j)})}$
$(i+1, j)$	$\rightarrow$	$P_{(x_{(i+1,j)})}$
$(i, j-1)$	$\rightarrow$	$P_{(x_{(i,j-1)})}$

Fonte: Autor, 2013.

Figura 5.2: Refinamento do grafo da *Equação da Onda (we)*



(a)Processo P_{CSP_1} (b)Processo P_{CSP_2}

(c)Processo P_{CSP_3} (d)Processo P_{CSP_4}

Fonte: Autor, 2013.

terpretado como canais de comunicação onde seu sincronismo representa a troca de dados entre os nós computáveis da malha computacional inerente aos problemas relacionados às equações **Hiperbólicas**.

Atendendo ao modelo de comunicação de CSP, apresenta-se a tabela 5.7, ao qual dar-se-á o significado semântico para cada processo que envolve a comunicação entre os nós. Os processos P_{CSP_1}, P_{CSP_2}, P_{CSP_3} e P_{CSP_4} representam a comunicação, através dos canais IN, entre os nós dependentes na malha computacional. Esta especificação está em conformidade com o modelo

de aproximação numérica (acurácia).

Tabela 5.7: Processos CSP para a *Equação da Onda*.

Processo		Representação CSP
P_{CSP_1}	=	$\big((IN_1!X_{(i,j-1)}) \rightarrow (IN_1?X_{(i,j+1)})\big) \longrightarrow P_{(x_{(i,j+1)})} \longrightarrow SKIP;$
P_{CSP_2}	=	$\big((IN_2!X_{(i-1,j)}) \rightarrow (IN_2?X_{(i,j+1)})\big) \longrightarrow P_{(x_{(i,j+1)})} \longrightarrow SKIP;$
P_{CSP_3}	=	$\big((IN_3!X_{(i,j)}) \rightarrow (IN_3?X_{(i,j+1)})\big) \longrightarrow P_{(x_{(i,j+1)})} \longrightarrow SKIP;$
P_{CSP_4}	=	$\big((IN_4!X_{(i+1,j)}) \rightarrow (IN_4?X_{(i,j+1)})\big) \longrightarrow P_{(x_{(i,j+1)})} \longrightarrow SKIP;$

Fonte: Autor, 2013.

Como resultado final, visto na Equação (5.1), o processo $TValues_{(we)}$ é dado como a composição em paralelo dos processos P_{CSP_1}, P_{CSP_2}, P_{CSP_3} e P_{CSP_4}. Considerando o domínio semântico na tabela 5.5, $TValues$ é uma entidade semântica que tem seu valor associado a um endereço na memória e $TValues_{(we)}$ é uma instância desta entidade para a equação da onda. Já $TValues_{(we)}^{(x_{(i,j+1)})}$ é um ponto específico desta memória com endereço de referência $x_{(i,j+1)}$.

$$\big(P_{CSP_1} \parallel P_{CSP_2} \parallel P_{CSP_3} \parallel P_{CSP_4}\big) \longrightarrow TValues_{(we)}^{(x_{(i,j+1)})} \tag{5.1}$$

5.3 Funções Semânticas

A função semântica mostra a relação entre o domínio sintático e o domínio semântico de *LaND*.

5.3.1 Função semântica dos comandos

A função semântica que define o significado dos comandos será uma função que transforma estados S da memória. Para isto é definido um caso particular da memória σ que mapeia variáveis Ide em inteiros $\mathbb{N}$ ($\sigma : S = Ide \rightarrow \mathbb{N}$).

$$\gamma : Comd \rightarrow S \rightarrow S$$

5.3.2 Função semântica dos pontos (nós) internos da malha

As funções semânticas que definem os pontos internos a serem computados são apresentados na tabela 5.8.

Tabela 5.8: Funções semânticas dos modelos de aproximação.

Função	Domínios
$S_{pwe}:$	$Delta \rightarrow TValues_{(we)}$
$S_{poe}:$	$Delta \rightarrow TValues_{(oe)}$

Fonte: Autor, 2013.

5.4 Equações semânticas

5.4.1 Representação da função λ

No desenvolvimento dos algoritmos para o MDF, uma das primeiras atividades é definir o valor da constante λ (Equações. (2.18), (2.19) , (2.20)). A abstração desta implementação é dada por um comando $size$ e $slot$, presentes na sintaxe abstrata de *LaND*. Lembrando que, no método numérico, esta função é o resultado da expressão $(k * c/h)^2$. Sendo $k = T/m$ (T é o tempo total e m o número de passos no tempo), a constante de propagação da onda c e $h = a/n$ (a é o espaço total e n número de passos no espaço). Em $LaND$, esta função (λ_{we}), uma instância específica para a equação da onda, teve seu comportamento abstraído e apresentado como,

$$\gamma[\![size(<\Gamma_s>;<\Gamma_t>); slot(<C_s>;<C_t>)]\!]_\sigma = \gamma[\![k]\!] \circ \gamma[\![h]\!]$$

onde as equações auxiliares,

$$\gamma[\![h]\!]_\sigma = \sigma[\![\Gamma_s]\!]/\sigma[\![C_s]\!]$$

$$\gamma[\![k]\!]_\sigma = \sigma[\![\Gamma_t]\!]/\sigma[\![C_t]\!]$$

confere significados sobre as computações sobre h e k, respectivamente.

Já o valor da função λ_{we}, é dado por uma expressão de valores armazenados em memória:

$$\gamma[\![\lambda_{we}]\!] = \sigma[\![k]\!] * C/\sigma[\![h]\!]$$

5.4.2 Representação das condições de contorno

Neste caso, a corda fixa em ambas as extremidades é uma condição de contorno do problema. Em uma matriz computacional (significado para $[\![TValues]\!]$), estrutura ideal para representar tal computação, os limites de espaço inicial e final para todos os tempos de simulação terão valores 0 (zero). Seguindo com as interpretações do primeiro comando: $j' = \{0, 1, \ldots, C_t\}$, representa uma computação para todos os passos de tempo (com seu limite C_t informado previamente pelo

usuário e apresentado na tabela 5.2), com o passo de espaço (inicial) $C_s = 0$, e no segundo comando o passo de espaço (final) $C'_s = C_s$ (também informado pelo usuário).

$$\gamma[\![WE_{edge}]\!] = [\![TValues]\!]_{j',0} \circ [\![TValues]\!]_{j',[\![C'_s]\!]} = 0.$$

5.4.3 Representação das condições iniciais

Para definir os valores das condições iniciais, o usuário fornece uma matriz computacional $[\![TValues]\!]$. Caso o usuário opte por WE_{Start}, na área sintática B (tabela 5.2), os valores de $[\![TValues]\!]$ serão definidos por uma função matemática externa de *LaND*. Esta função dá os valores no instante do tempo C_t igual a zero, nos pontos espaciais da malha variando em $i' = \{1, 2, \ldots, C_s - 1\}$. Ou seja, o valor de $u(x, t = 0)$ no domínio espacial discreto.

$$\gamma[\![WE_{start}]\!] = [\![TValues]\!]_{0,i'}$$

Uma das pré-condições para o processamento da malha computacional está na condição inicial do problema. Para que sejam satisfeitas as condições apresentadas na equação (2.20) se faz necessário especificar o significado do seu comportamento, caso o usuário opte pelo comando WE dentro do domínio sintático Δ.

Para definir os valores da matriz, a proposta é que esta função faça as computações, no passo de tempo = 1 e variando os valores do passo de espaço em $i' = \{1, 2, \ldots, C_s - 1\}$, como segue:

$$S_{pwe}[\![WE]\!] = [\![\lambda_{we}]\!]^2/2 * ([\![TValues]\!]_{0,i+1} + [\![TValues]\!]_{0,i-1}) + (1 - [\![\lambda_{we}]\!]^2 * [\![TValues]\!]_{0,i}) + [\![k]\!] * 0.$$

Este comando também é responsável por processar os demais pontos internos da malha, apresentados a seguir.

5.4.4 Representação do processamento da malha numérica computacional

Aqui será representado o significado do mecanismo que dará o processamento dos pontos internos da malha computacional. Caso o usuário opte por processar uma equação da onda, com o comando WE dentro do domínio sintático Δ, este irá abstrair as operações conforme apresentado na tabela 5.9.

Um aspecto importante a ser mencionado nesta equação da tabela 5.9 está na técnica do processamento das equações hiperbólicas pelo MDF e sua respectiva representação semântica.

Tabela 5.9: Equação Semântica que gera a malha computacional da *Equação da Onda*.

Metalinguagem		Representação
$S_{pwe}[\![WE]\!]$	=	$[\![\lambda_{we}]\!]^2 * [\![TValues]\!]_{(j,i+1)} + 2 * (1 - [\![\lambda_{we}]\!]^2) *$ $[\![TValues]\!]_{(j,i)} + [\![\lambda_{we}]\!]^2 *$ $[\![TValues]\!]_{(j,i-1)} - [\![TValues]\!]_{(j-1,i)}$

Fonte: Autor, 2013.

Para obter as aproximações nestas coordenadas da malha, as computações são realizadas de forma recursiva dentro do domínio do problema. Quando o usuário opta pela resolução de uma equação da onda, $S_{pwe}[\![WE]\!]$ é uma função pré-definida de *LaND* que dará ao usuário a abstração necessária que implementa tal mecanismo (através da geração automática do código equivalente). Do ponto de vista de projeto da linguagem esta equação semântica diz qual o significado da computação necessária para encontrar o valor do ponto na malha de valor semântico $S_{pwe}[\![WE]\!]$.

5.5 Exemplo de programa em *LaND*

Nos programas escritos em *LaND* o usuário deverá apenas conhecer as funções, da linguagem, responsáveis na computação que simula cada etapa do processo de desenvolvimento de uma solução numérica com o método das diferenças finitas.

Similarmente as etapas do processo de construção desta solução numérica, o usuário deve apenas fazer as chamadas das funções pré-definidas de *LaND* responsáveis por cada uma destas etapas. Para efeitos de demonstração, os passos a seguir exemplificam como será um programa em *LaND*, onde seu domínio do problema é uma equação hiperbólica de segunda ordem no espaço e primeira ordem no tempo:

1^0**) Definir um nome ao programa** - Como apresentada na Figura 5.3, sua primeira entrada será um nome para o programa em *LaND*. Dado a sintaxe abstrata (tabela 5.2), seus programas são definidos como módulos **<Module>** seguido do seu nome, um **<Identificador>**, e da palavra reservada **where**. Esta sintaxe é similar aos módulos de Haskell, e sua ideia é de facilitar a integração com outros programas em *LAND* que futuramente venham a ser acoplados a este módulo;

2^0**) Valores para o λ da equação discretizada** - Na seção 5.4.1 foram apresentadas as equações semânticas da função λ da equação discretizada. Para que esta computação seja realizada, *LaND* disponibiliza duas funções que abstraem a complexidade no desenvolvimento do algoritmo responsável por esta ação. Neste momento o usuário faz a chamada da função **size(**x_1**,** x_2**)** onde, o primeiro parâmetro x_1, uma constante numérica, é o limite do espaço e o segundo parâmetro x_2, é o tempo total de simulação. Também é neste mo-

mento em que usuário deve definir os valores para função **slot(**y_1**,** y_2**)** onde, os parâmetros y_1 e y_2, representam respectivamente os números de pontos no espaço e no tempo. Com estas duas funções, e seus valores em seus parâmetros, a malha computacional é definida;

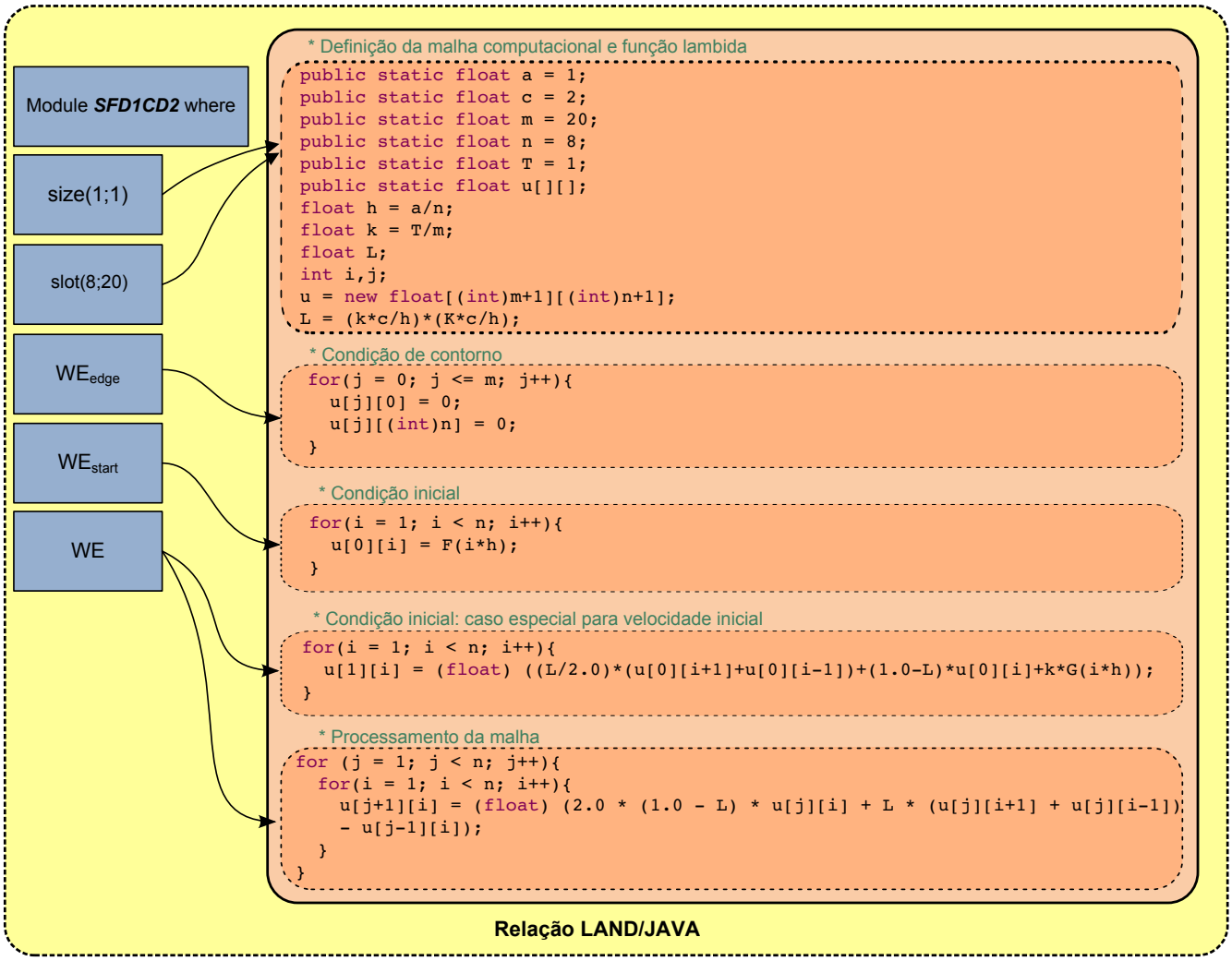
3^0**) Definir as condições de contorno** - Definida a malha computacional e a função λ, a próxima etapa é definir as condições de contorno do problema. Quando o usuário opta pela função WE_{edge}, *LaND* irá simular o processo da condição inicial da equação da onda. Neste caso uma matriz terá seus valores iniciais 0 (zero), nos pontos do espaço inicial e final, para todos os tempos de simulação da malha computacional. Neste caso a corda está fixa nas extremidades;

4^0**) Definir a condição inicial** - Nesta etapa o usuário utiliza uma função de *LaND* que irá simular as condições iniciais do problema. A função WE_{start} define valores para toda a linha inicial da malha computacional (tempo 0 da simulação). O seu limite de espaço, desta matriz, já foi dado no primeiro parâmetro passado pela função *slot* (seção 5.4.1);

5^0**) Definir a condição de velocidade inicial e processamento da malha** - A última ação que o usuário deverá realizar é utilizar a função de *LaND* WE para a velocidade inicial e processamento de toda a malha computacional nos pontos ainda não computados. No caso da velocidade inicial, esta função será responsável por dar valores a segunda linha da malha computacional (tempo 1 da simulação). A necessidade de dar valores para esta coordenada foi discutida na seção 5.4.3. Finalmente, este comando também será responsável por dar os valores de todos os pontos da malha ainda não computados. O comportamento desta computação também já foi discutido na seção 5.4.4.

Para ilustrar o que foi discutido, apresenta-se a Figura 5.3 onde, do lado esquerdo tem-se as composições das funções de *LaND* e do lado direito partes de um programa em Java. Esta figura explica a relação das funções definidas pelo *solver LaND* e seu correlato, um programa em Java. Ambos tratam o mesmo problema, a resolução de um problema pelo MDF.

Figura 5.3: Mapeamento de *LaND* com um programa correlato em Java.



Fonte: Autor, 2013.

De acordo com o apresentado (Figura 5.3), pode-se perceber algumas vantagens nos programas escritos em *LaND*:

- **Abstração das estruturas computacionais**. *LaND* possui um conjunto de funções pré-definidas que abstraem a complexidade no desenvolvimento de estrututas computacionais como matrizes e laços de repetição.

- **Simplicidade na implementação do programa de resolução**. Por se tratar de uma linguagem funcional, *LaND* possui um conjunto de funções pré-definidas que abstraem a implementação dos passos de resolução de um MDF. Esta característica evita ou minimiza erros na implementação do algoritimo de resolução em linguagens tradicionais de programação.
- **Maior legibilidade do código**. Inerente a sua natureza funcional os programas em *LaND* são mais simples, consequentemente possuie maior legibilidade do seu código.
- **Código de fácil manutencão**. Como reflexo de um código mais simples e de maior legibilidade a sua manutenção torna-se também mais simples.

Neste capítulo foi apresentado um modelo formal para *LaND*, uma linguagem que abstrai a complexidade inerente à programação de algoritmos na resolução de problemas com uso do Método das Diferenças Finitas. O formalismo adotado foi da Semântica Denotacional e CSP com demonstrações apoiadas em Grafos para os processos síncronos envolvidos no método de aproximação numérica.

No próximo capítulo serão apresentados as conclusões e sugestões de trabalhos futuros.

6 CONCLUSÃO

NESTE trabalho discutiu-se os problemas envolvidos na construção de algoritmos de métodos numéricos computacionais, especificamente o método das diferenças finitas. A implementação destes algoritmos computacionais pode demandar um tempo e esforço considerável do engenheiro ou cientista no tratamento do problema. Para facilitar o processo de desenvolvimento de soluções computacionais, apresentou-se a alternativa de utilização de especificação formal semântica de *LaND*.

Esta linguagem de programação é caracterizada por minimizar a complexidade envolvida no desenvolvimento do software científico de simulações de EDPs para o Método das Diferenças Finitas.

Também citou-se as contribuições ligadas à especificação semântica formal de *LaND*. A utilização da Semântica Denotacional combinada à CSP representaram uma interessante alternativa para designers de linguagens artificiais computacionais, atribuindo significados ao esclarecer as construções sintáticas e funções de *LaND*.

Esta abordagem semântica tem potencial para criar uma ferramenta de auxílio no processo de construção de um interpretador para *LaND* e, através do operador de paralelismo intrínseco à CSP, pode ser utilizada para geração automática de código paralelo a partir do modelo matemático do problema físico.

Ou seja, com a sintaxe de *LaND* definida, sustentada por este projeto semântico, o objetivo é que o engenheiro ou cientista tenha um suporte ferramental que minimize o esforço na construção dos algoritmos de solução numérica. De fato, seu esforço estará concentrado em definir, em sintaxe *LaND*, os aspectos inerentes à sua especialidade, tais como: condições de contorno, condições iniciais, tamanho da malha numérica computacional e métodos de aproximação (acurácia), deixando a cargo da ferramenta a geração automática do programa equivalente para resolução do problema.

6.1 Trabalhos futuros

Alguns detalhes ficaram em aberto na especificação da linguagem, os quais pode-se citar:

- Tratamento do erro de aproximação numérica. Faz-se necessária a implementação de procedimentos técnicos que evitem os erros de truncamento global, local e arredondamento antes que a aproximação numérica, computada por *LaND*, possa ser aceita como satisfatória.

- Uso de outros métodos numéricos a exemplo do Método dos Elementos Finitos. Diferentemente das aproximações nas derivadas das equações diferenciais que reduzem a um problema de sistemas de equações lineares no MEF a solução das equações diferenciais pode ser resolvida por funções de aproximação que satisfazem condições descritas por equações integrais no domínio do problema. Outra diferença entre os métodos é quanto a topologia de discretização do domínio. Em *LaND*, a geração automática da malha computacional está limitada a uma topologia de malha estruturada onde os intervalos entre os nós adjacentes nas direções “x”e “y” são constantes.

- Detalhamento do significado dos processos CSP esboçando uma extensão da linguagem para abstrair programas no paradigma paralelo. Como consequência desta característica, não será exigido conhecimento prévio dos paradigmas do desenvolvimento de programas paralelos ao usuário, como os requisitos na dependência de dados.

- Uma definição complementar baseada nas abordagens da especificação sintática de linguagens como a *Backus–Naur Form* (BNF), *Extended Backus–Naur Form* (EBNF) ou Diagramas Sintáticos. Esta atividade, juntamente a este trabalho monográfico, possibilitaria a implementação de um compilador para a linguagem.

- A construção de um ambiente de desenvolvimento. De posse do compilador de *LaND*, esta ferramenta de desenvolvimento deverá ser capaz de detectar erros sintáticos na escrita das equações mediante regras gramaticais e visualização automática do código fonte do programa equivalente. Este ambiente poderá ser disponibilizado como um *Plug-in* para IDE Eclipse. A escolha pelo Eclipse está na sua capacidade de extenção à plug-ins através do seu ambiente *Plug-in Development Environment* (PDE) (Potrich, 2006).

- Em última instância este interpretador pode, de forma intrínseca a CSP, ser usado na geração de programas paralelos em uma linguagem alvo arbitrária, como C/C++ ou Fortran já utilizando bibliotecas como MPI ou APIs como OpenMP, bem como a utilização das diretrizes de CUDA nestas linguagens (Sanders & Kandrot, 2010).

REFERÊNCIAS

Allison, L. (1986), *A Practical Introduction to Denotational Semantics*, Cambridge Computer Science Texts, Cambridge University Press.

Almeida, E. & Haeusler, E. H. (1994), Utilização de CSP como ferramenta para especificação semântica de linguagens orientadas a objetos, *in* '14 Seminário Integrado de Software e Hardware'.

Brannan, J. R. & Boyce, W. E. (2008), *Equações Diferenciais: Uma Introdução a Métodos Modernos e Suas Aplicações*, LTC, Rio de Janeiro.

Burden, R. & Faires, J. (2011), *Numerical Analysis*, ninth ed., Richard Stratton.

Canneyt, M. V. (2011), *Free Pascal: Reference guide.* URL ftp://ftp.freepascal.org/fpc/docs-pdf/ref.pdf.

Chaquet, J. M. & Carmona, E. J. (2012), 'Solving differential equations with fourier series and evolution strategies', *Applied Soft Computing* **12**(9), 3051 – 3062. URL http://www.sciencedirect.com/science/article/pii/S1568494612002505.

Claudio, D. C. & Marins, J. M. (2000), *Cálculo Numérico Computacional*, Atlas.

Colvin, R. J. & Hayes, I. J. (2011), 'A semantics for behavior trees using csp with specification commands', *Science of Computer Programming* **76**, 891–914.

Cook, S., Jones, G., Kent, S. & Wills, A. (2007), *Domain-specific Development with Visual Studio Dsl Tools*, first ed., Addison-Wesley Professional.

Fischer, C. (n.d.), Csp-oz: A combination of object-z and csp, Technical report. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.5983.

Hannan, J. & Miller, D. (1990), From operational semantics to abstract machines: preliminary results, *in* 'Proceedings of the 1990 ACM Conference on LISP and Functional Programming', LFP '90, ACM, New York, NY, USA, pp. 323–332.

Hoare, C. A. R. (1969), ‘An axiomatic basis for computer programming’, *Commun. ACM* **12**(10), 576–580. URL http://doi.acm.org/10.1145/363235.363259.

Hoare, C. A. R. (1978), ‘Communicating sequential processes’, *Commun. ACM* **21**(8), 666–677. URL http://doi.acm.org/10.1145/359576.359585.

Hoare, C. A. R. (2004), *Communicating Sequential Processes*, Prentice Hall.

Hoare, C. A. R. & Wirth, N. (1973), ‘An axiomatic definition of the programming language pascal’, *Acta Informatica* **2**, 335–355. URL http://dx.doi.org/10.1007/BF00289504, 10.1007/BF00289504.

Japaridze, G. (2005), ‘In the beginning was game semantics’, *CoRR*.

Kahn, G. (1987), ‘Natual semantics’, *Lecture Notes in Computer Science* **247**, 22–39.

Knabner, P. & Angermann, L. (2003), *Numerical Methods for Eliptic and Parabolic Partial Differential Equations*.

Krishnamurthi, S. (2007), *Programming Languages: Application and Interpretation*, version as of 2007-04-26 — 1st version: 2003-12-15 ed., Brown University.

Lapidus, L. & Pinder, G., F. (1999), *Numerical Solution of Differential Equations in Science and Engineering*, John Wiley and Sons.

MacLennan, B. (2012), ‘Algol 60 grammar in bnf @ONLINE’. URL http://web.eecs.utk.edu/~mclennan/Classes/312/handouts/Algol60-grammar.pdf.

Menezes, P. B. (2008), *Linguagens Formais e Autômatos*, Livros didáticos, Bookman.

Mosses, P. D. (1996), Theory and practice of action semantics, pp. 37–61.

NASA (2001), ‘What is formal methods?’, Online: http://shemesh.larc.nasa.gov/fm/fm-what.html. URL http://shemesh.larc.nasa.gov/fm/fm-what.html, accessed in july 21, 2012.

Nielson, H. R. & Nielson, F. (1992), *Semantics with applications: a formal introduction*, John Wiley & Sons, Inc., New York, NY, USA.

Noonan, R. & Tucker, A. (2008), *Linguagens de programação: princípios e paradigmas*, McGraw Hill - Artmed.

Oliveira, S. & Stewart, D. (2006), *Writing Scientific Software: a guide to good style*, Cambridge.

Plotkin, G. D. (1981), A structural approach to operational semantics, Technical report, University of Aarhus. URL http://citeseer.ist.psu.edu/plotkin81structural.html.

Potrich, E. (2006), Pews editor, um front-end para a linguagem pews, Master's thesis, Universidade Federal do Paraná.

Reed, G. M. & Roscoe, A. W. (1988), 'A timed model for communicating sequential processes', *Theor. Comput. Sci.* **58**(1-3), 249–261. URL http://dx.doi.org/10.1016/0304-3975(88)90030-8.

Ricarte, I. (2008), *Introdução à Compilação*, primeira ed., Elsevier.

Robaina, D. T., Guedes, M. J. M., Drummond, L. M. A., Kischinhevsky, M. & Filho Otton, T. S. (2005), Solução numérica de equações diferenciais parciais parabólicas usando o método hopscotch com refinamento não-uniforme, *in* 'Anais do XXVIII CNMAC – Congresso Nacional de Matemática Aplicada e Computacional', Santo Amaro, SP.

Roscoe, A. (2005), *The Theory and Practice of Concurrency*, Prentice-Hall.

Sanders, J. & Kandrot, E. (2010), *CUDA by Example: An Introduction to General-Purpose GPU Programming*, 1st ed., Addison-Wesley Professional.

Schneider, F. A. (2007), Verificação de Soluções Numéricas em Problemas Difusos e Advectivos com Malhas não-Uniformes, PhD thesis, Universidade Federal do Paraná. URL http://www.ppgmne.ufpr.br/arquivos/teses/8.pdf.

Scott, D. S. (1982), Domains for denotational semantics, *in* 'Proceedings of the 9th Colloquium on Automata, Languages and Programming', Springer-Verlag, London, UK, pp. 577–613. URL http://dl.acm.org/citation.cfm?id=646236.682867.

Scott, D. S. & Strachey, C. (1971), Toward a mathematical semantics for computer languages, *in* J. Fox, ed., 'Proceedings of the Symposium on Computers and Automata', Vol. XXI, Polytechnic Press, Brooklyn, N.Y., pp. 19–46.

Sebesta, R. (2003), *Conceitos de Linguagens de Programação*, Bookman.

Sebesta, R. (2010), *Conceitos de Linguagens de Programação*, Bookman.

Slonneger, K. & Kurtz, B. (1995), *Formal Syntax and Semantics of Programming Languages: A Laboratory Based Approach*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Sá, C. C. & Silva, M. F. (2006), *Haskell: uma Abordagem Prática*, primeira ed., Novatec.

Tennent, R. D. (1976), The denotational semantics of programming languages, *in* 'Proceedings of the Comunnications of the ACM', Vol. 19, ACM, New York, NY, USA, pp. 437–453.

Winskel, G. (1993), *The Formal Semantics of Programming Languagens: an introduction*, MIT Press.

Zhang, Y. & Xu, B. W. (2004), 'A survey of semantic description frameworks for programming languages', *ACM Sigplan Notices* **39**(3), 14–30.

Zill, D. G. & Cullen, M. R. (2001), *Equações Diferenciais*, terceira ed., Makron Books.

Apêndice A

Programa da Equação da Onda em Java

```
1  public class HyperbolicWaveEquation {
2
3   public static float a = 1;
4   public static float c = 2;
5   public static float m = 20;
6   public static float n = 8;
7   public static float T = 1;
8   public static float u[][];
9
10  static float F(float d){
11          return (float) Math.sin(3.14159*d);
12  }
13
14  static float G(float d){
15          return (float) 0.0;
16  }
17
18  public static void main(String args[]){
19
20    float h = a/n;
21    float k = T/m;
22    float L;
23    int i, j;
24    u = new float[(int) m+1][(int) n+1];
25
26    L=(k*c/h)*(k*c/h);
27
28
29    /*
30     * Condicao de contorno
31     */
32
33    for (j=0; j<=m; j++) {
34          u[j][0]=0;
35          u[j][(int) n]=0;
36    }
37
38    /*
39     * Condicao inicial
40     */
41
42    for (i=1; i<n; i++){
```

```
43          u[0][i] = F(i*h);
44      }
45
46      /*
47       * Condicao inicial: caso especial para velocidade inicial.
48       */
49
50      for (i=1; i<n; i++){
51          u[1][i] = (float) ((L/2.0)*(u[0][i+1] + u[0][i-1]) + (1.0 - L) * u[0][i] + k * G(i*h));
52      }
53
54      /*
55       * Processamento da malha.
56       */
57
58      for (j=1; j<m; j++){
59          for (i=1; i<n; i++){
60            u[j+1][i] = (float) (2.0* (1.0 - L) * u[j][i] + L*(u[j][i+1] + u[j][i-1])
61                                - u[j-1][i]);
62          }
63      }
64
65      for (j=0; j<=m; j++){
66        for (i=0; i<=n; i++){
67          System.out.printf("␣␣%.4f␣",u[j][i]);
68        }
69          System.out.println("");
70      }
71   }
72  }
```

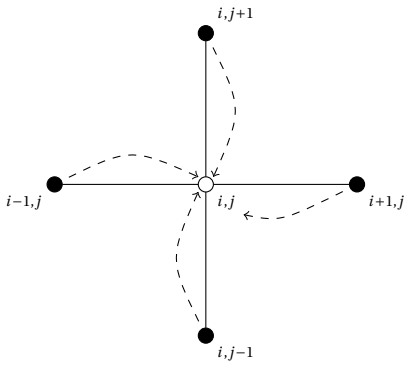
Apêndice B

Outras Especificações de Aproximação

B.1 Domínio Semântico da aproximação por *Central Difference*

Método utilizado nas equações **Elípticas**, como Poisson e Laplace.
Representamos o método da aproximação dos cinco pontos como processos em CSP.

Figura B.1: Grafo de dependência por *Central Difference*.



Fonte: Autor, 2013.

Para este exemplo, e os demais casos que seguem as seções B.2 e B.3, omite-se por questões de simplicidade o refinamento dos respectivos grafos e mapeamentos destes processos.
Neste problema, o tratamento é feito por técnica do tipo *Central Difference*, o conjunto de processos CSP pode ser visto na tabela B.1, onde o seu resultado é a composição parelala dos processos P_{CSP_1}, P_{CSP_2}, P_{CSP_3} e P_{CSP_4} resultando no processo principal $TValues_{(cd)}$.

$$\left(P_{CSP_1} \parallel P_{CSP_2} \parallel P_{CSP_3} \parallel P_{CSP_4}\right) \longrightarrow TValues_{(cd)}^{(x_{(i,j)})}$$

O processo $TValues_{(cd)}$ é dado como a composição em paralelo dos processos P_{CSP_1}, P_{CSP_2}, P_{CSP_3} e P_{CSP_4}, no endereço de memória $(x_{(i,j)})$.

Tabela B.1: Processos CSP para *Central Difference*

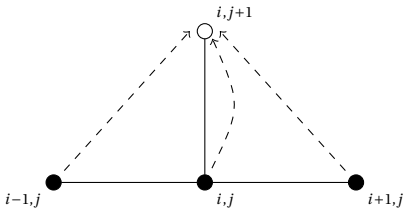
Processo		Representação CSP							
P_{CSP_1}	=	$\big((IN_1!X_{(i,j+1)})$	$\rightarrow$	$(IN_1?X_{(i,j)})\big)$	$\longrightarrow$	$P_{(x_{(i,j)})}$	$\longrightarrow SKIP;$		
P_{CSP_2}	=	$\big((IN_2!X_{(i-1,j)})$	$\rightarrow$	$(IN_2?X_{(i,j)})\big)$	$\longrightarrow$	$P_{(x_{(i,j)})}$	$\longrightarrow SKIP;$		
P_{CSP_3}	=	$\big((IN_3!X_{(i,j-1)})$	$\rightarrow$	$(IN_3?X_{(i,j)})\big)$	$\longrightarrow$	$P_{(x_{(i,j)})}$	$\longrightarrow SKIP;$		
P_{CSP_4}	=	$\big((IN_4!X_{(i+1,j)})$	$\rightarrow$	$(IN_4?X_{(i,j)})\big)$	$\longrightarrow$	$P_{(x_{(i,j)})}$	$\longrightarrow SKIP;$		

Fonte: Autor, 2013.

B.2 Domínio Semântico da aproximação por *Forward Time, Centered Space* (FTCS)

Método explícito utilizado nas equações **Parabólicas**, como a equação do calor: distribuição de temperatura ao longo de uma barra.

Figura B.2: Grafo de dependência por *Forward-Time, Centered-Space*.



Fonte: Autor, 2013.

Neste problema, o tratamento é feito por técnica do tipo *Forward-Time, Centered-Space*, o conjunto de processos CSP pode ser visto na tabela B.2, onde o seu resultado é a composição parelala dos processos P_{CSP_1}, P_{CSP_2} e P_{CSP_3} resultando no processo principal $TValues_{(ftcs)}$.

Tabela B.2: Processos CSP por FTCS

Processo		Representação CSP							
P_{CSP_1}	=	$\big((IN_1!X_{(i-1,j)})$	$\rightarrow$	$(IN_1?X_{(i,j+1)})\big)$	$\longrightarrow$	$P_{(x_{(i,j+1)})}$	$\longrightarrow SKIP;$		
P_{CSP_2}	=	$\big((IN_2!X_{(i,j)})$	$\rightarrow$	$(IN_2?X_{(i,j+1)})\big)$	$\longrightarrow$	$P_{(x_{(i,j+1)})}$	$\longrightarrow SKIP;$		
P_{CSP_3}	=	$\big((IN_3!X_{(i+1,j)})$	$\rightarrow$	$(IN_3?X_{(i,j+1)})\big)$	$\longrightarrow$	$P_{(x_{(i,j+1)})}$	$\longrightarrow SKIP;$		

Fonte: Autor, 2013.

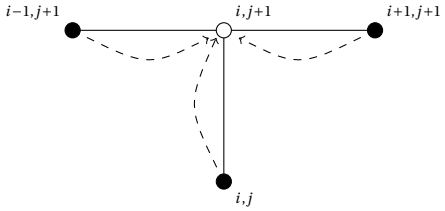
$$\Big(P_{CSP_1} \parallel P_{CSP_2} \parallel P_{CSP_3}\Big) \longrightarrow TValues_{(ftcs)}^{(x_{(i,j+1)})}$$

O processo $TValues_{(ftcs)}$ é dado como a composição em paralelo dos processos P_{CSP_1}, P_{CSP_2} e P_{CSP_3}, no endereço de memória $(x_{(i,j+1)})$.

B.3 Domínio Semântico da aproximação por *Backward Time, Centered Space* (BTCS)

Método implicito também utilizado nas equações **Parabólicas**, que envolvem a distribuição do calor sobre corpos (equação do calor).

Figura B.3: Grafo de dependência por *Backward Time, Centered Space*.



Fonte: Autor, 2013.

Neste problema, o tratamento é feito por técnica do tipo *Backward Time, Centered Space*, o conjunto de processos CSP pode ser visto na tabela B.3, onde o seu resultado é a composição parelala dos processos P_{CSP_1}, P_{CSP_2} e P_{CSP_3} resultando no processo principal $TValues_{(btcs)}$.

Tabela B.3: Processos CSP por BTCS

Processo		Representação CSP						
P_{CSP_1}	=	$\big((IN_1!X_{(i-1,j+1)})$	$\rightarrow$	$(IN_1?X_{(i,j+1)})\big)$	$\longrightarrow$	$TValues_{(x_{(i,j+1)})}$	$\longrightarrow$	$SKIP;$
P_{CSP_2}	=	$\big((IN_2!X_{(i,j)})$	$\rightarrow$	$(IN_2?X_{(i,j+1)})\big)$	$\longrightarrow$	$TValues_{(x_{(i,j+1)})}$	$\longrightarrow$	$SKIP;$
P_{CSP_3}	=	$\big((IN_3!X_{(i+1,j+1)})$	$\rightarrow$	$(IN_3?X_{(i+1,j+1)})\big)$	$\longrightarrow$	$TValues_{(x_{(i,j+1)})}$	$\longrightarrow$	$SKIP;$

Fonte: Autor, 2013.

$$\Big(P_{CSP_1} \parallel P_{CSP_2} \parallel P_{CSP_3}\Big) \longrightarrow TValues_{(btcs)}^{(x_{(i,j+1)})}$$

O processo $TValues_{(x_{(btcs)}}$ é dado como a composição em paralelo, modelado em CSP, dos processos P_{CSP_1}, P_{CSP_2} e P_{CSP_3}, no endereço de memória $(x_{(i,j+1)})$.

Este trabalho foi redigido em LaTeX utilizando uma modificação do estilo IC-UFAL. As referências bibliográficas foram preparadas no JabRef e administradas pelo BibTeX com o estilo LaCCAN. O texto utiliza fonte Fourier-GUT*enberg* e os elementos matemáticos a família tipográfica Euler Virtual Math, ambas em corpo de 12 pontos.