UNIVERSIDADE FEDERAL DO MARANHÃO

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA - CCET

ENGENHARIA DA COMPUTAÇÃO

ERNAMILSON REZENDE DOS SANTOS FILHO

HVM2 E *INTERACTION COMBINATORS*:
UMA REVISÃO NARRATIVA DA LITERATURA

São Luís - MA

2025

ERNAMILSON REZENDE DOS SANTOS FILHO

HVM2 E *INTERACTION COMBINATORS*:
UMA REVISÃO NARRATIVA DA LITERATURA

Monografia apresentada ao curso de Engenharia da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Engenharia da Computação.

Orientador: Prof. Dr. Bruno Feres de Souza

São Luís - MA

2025

ERNAMILSON REZENDE DOS SANTOS FILHO

HVM2 E *INTERACTION COMBINATORS*:
UMA REVISÃO NARRATIVA DA LITERATURA

Monografia apresentada ao curso de Engenharia da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Engenharia da Computação.

Aprovado em 26/02/2025

Prof. Dr. Bruno Feres de Souza
UFMA–Orientador

Prof. Dr. Paulo Rogério de Almeida Ribeiro
UFMA

Prof. Dr. Sérgio Souza Costa
UFMA

São Luís - MA

10 de Fevereiro de 2025

“Como podiam os homens guardar tantas palavras? Era impossível, ninguém conservaria tão grande soma de conhecimentos. Livres dos nomes, as coisas ficavam distantes, misteriosas. Não tinham sido feitas por gente. E os indivíduos que mexiam nelas cometiam imprudência. Vistas de longe, eram bonitas. Admirados e medrosos, falavam baixo para não desencadear as forças estranhas que elas porventura encerrassem.”

(O menino mais novo, em Vidas Secas – Graciliano Ramos)

HVM2 E *INTERACTION COMBINATORS*:

UMA REVISÃO NARRATIVA DA LITERATURA

Resumo: A formalização dos modelos computacionais, baseada nos trabalhos de Turing e Church, estabeleceu as bases para o desenvolvimento de novos paradigmas, como o modelo de computação universal e distribuído proposto por Lafont em Interaction Combinators. Assim, a partir da necessidade de entender o funcionamento da HVM2, um avaliador eficiente e massivamente paralelo para interaction combinators estendidos, este trabalho realiza uma revisão narrativa da literatura estruturada, tomando como base os principais artigos introdutórios sobre os fundamentos teóricos e práticos deste avaliador e dos Interaction Combinators. Essa revisão conecta conceitos fundamentais (Proof-Nets, Interaction Nets) à implementação moderna de computação distribuída na HVM2. A partir dessa análise, é possível identificar os conceitos fundamentais que estruturam as abstrações criadas a partir das proof-nets, culminando na definição de um sistema universal e distribuído, caracterizado por propriedades de alta confluência e localidade. Além disso, a concretização dessa teoria é observada na HVM2, cuja implementação é analisada à luz de suas limitações e desafios técnicos.

Palavras chave: proof-nets; interaction nets; interaction combinators; HVM2; computação distribuída; modelos computacionais.

HVM2 AND *INTERACTION COMBINATORS*:
A NARRATIVE LITERATURE REVIEW

Abstract: The formalization of computational models, based on the works of Turing and Church, laid the foundations for the development of new paradigms, such as the universal and distributed computation model proposed by Lafont in Interaction Combinators. Thus, driven by the need to understand the functioning of HVM2, an efficient and massively parallel evaluator for extended interaction combinators, this work conducts a structured narrative literature review, drawing from key introductory articles on the theoretical and practical foundations of this evaluator and Interaction Combinators. This review connects fundamental concepts (Proof-Nets, Interaction Nets) to the modern implementation of distributed computing in HVM2. Through this analysis, it is possible to identify the core concepts that structure the abstractions derived from proof-nets, culminating in the definition of a universal and distributed system characterized by high confluence and locality properties. Furthermore, the realization of this theory is observed in HVM2, whose implementation is analyzed in light of its limitations and technical challenges.

Keywords: proof-nets; interaction nets; interaction combinators; HVM2; distributed computing; computational models.

LISTA DE FIGURAS

SUMÁRIO

1. Introdução

A existência da HVM (Higher Order Virtual Machine), definida como um avaliador eficiente e massivamente paralelo para combinadores de interação estendidos, é precedida de de diversos aspectos teóricos como *Proof-Nets* (Redes de Prova), *Interaction Nets* (Redes de Interação) e *Interaction Combinators* (Combinadores de Interação). Assim, é necessária a contextualização destes principais pontos que fundamentam a HVM através da abordagem das partes do campo teórico e da aplicação necessárias para a implementação desta.

Portanto, de modo a melhor compreender esta aplicação e seu campo teórico será realizada uma revisão da literatura dos principais trabalhos relacionados aos tópicos citados anteriormente, com intuito de introduzir e destacar as correlações e dependências entre estes.

1.1. Contextualização do Campo Teórico

A partir da formalização de modelos computacionais fundamentalizada por Alan Turing, Alonzo Church e outros teóricos, por meio da Máquina de Turing, do Cálculo Lambda e de outros modelos de computação, se destacou a existência de problemas computacionais insolucionáveis e a existência de modelos de computação com aspecto universal, ou seja, capazes de simular qualquer outro modelo de seu tipo (COPELAND, 2024).

Deste ponto, com o surgimento da lógica linear e o avanço da computação teórica e dos modelos computacionais existentes, pôde-se formalizar a definição das *Interaction Nets*, apresentadas como uma generalização das *Proof-Nets* para Lógica Linear multiplicativa, de Girard. Assim, Lafont propõe em seu trabalho uma linguagem de programação que apresenta como principais aspectos uma semântica de reescrita de grafos, simetria entre construtores e destrutores e uma disciplina de tipo que irá garantir a característica de um paralelismo determinístico e livre de empasses (LAFONT, 1990, p. 1). Assim, pode se considerar que o conceito de modelo computacional abordado a partir das *Interaction Nets*, é dado de forma diferente à de Turing e Church, uma vez que estas são baseadas em *Proof-Nets*.

Em continuação, Lafont apresenta *Interaction Combinators*, que propõe um modelo de computação universal e distribuído através do uso de 3 símbolos e 6 regras. Além disso, são introduzidas as noções de *translation of interaction systems* (tradução de sistemas de interação) e de *universal interaction system* (sistema universal de interação) que possuem influência nas características de paralelismo e de universalidade da proposta (LAFONT, 1997).

Assim, com tudo que fora apresentado é possível discorrer sobre a implementação deste modelo computacional dado pelos *Interaction Combinators* na HVM, onde é definida uma sintaxe para a representação dos combinadores a partir de um sistema de *Interaction Calculus*, uma extensão dos combinadores definidos por Lafont, uma implementação do sistema de interação (reescrita de grafos) e por fim a implementação deste sistema com a definição de um *layout* de memória, multi-threading e como será performada a conexão entre os combinators no programa.

1.2. Justificativa

A escolha do tema e a abordagem utilizada neste trabalho podem ser justificadas pela relevância da computação distribuída no cenário tecnológico atual. Modelos que provêm características de paralelismo e distribuição, como *Interaction Combinators*, podem se tornar uma base inovadora para superar limitações dos modelos computacionais tradicionais. Dessa forma, a HVM é introduzida como uma implementação prática baseada na teoria de tais conceitos e que, devido à sua natureza, acaba por ser um modelo de computação universal que proporciona paralelismo e distribuição a partir da avaliação de *Interaction Combinators*.

Além disso, a fundamentação teórica que baseia a HVM é essencial para que se observe as possibilidades de avanço na computação. Com uma revisão das bases teóricas, que incluem *Proof-Nets*, *Interaction Nets* e *Interaction Combinators*, pode-se visualizar a conexão entre os pontos e como se integram, se desenvolvem e são aplicados estes conceitos.

Sendo assim, este estudo busca, além de ampliar o entendimento sobre o modelo computacional utilizado na HVM, contribuir para a iniciação nas teorias da computação distribuída e modelos computacionais para quem vier a ler. Portanto, este trabalho se justifica pela oportunidade de correlacionar e destrinchar o campo teórico juntamente com a análise da aplicação prática e suas limitações.

1.3. Objetivos

1.3.1. Objetivo Geral

Analisar o contexto teórico relacionado ao modelo universal de computação distribuída dado pelos *Interaction Combinators* e compreender a implementação desse modelo realizada na HVM através de uma revisão narrativa da literatura.

1.3.2. Objetivos Específicos

- Analisar a definição da linguagem proposta em *Interaction Nets*;
- Analisar os combinadores e regras definidos em *Interaction Combinators*;
- Analisar as abordagens teóricas, a trajetória e as perspectivas da HVM;

2. Metodologia

A escolha da metodologia foi baseada na pesquisa de identificação de tipos de métodos de revisão realizada por Maria J. Grant e Andrew Booth, através da análise manuscritos contendo termos e frases que remeteram à revisão. Dos tipos identificados, foi observado que, através da descrição, pontos fortes e fracos, o método mais adequado seria o de revisão da literatura/revisão narrativa, uma vez que se tem como principal foco a amostra dos tópicos teóricos consolidados préviamente e que sua fraqueza percebida, de falta de proposições de maximização do escopo teórico, não é tão agravante dada a proposta do trabalho.

Desse modo, dados os objetivos do trabalho, se chega à conclusão de que a metodologia mais adequada é a revisão da literatura, de modo a disponibilizar uma conexão entre as teorias relacionadas ao tema. Entretanto, a revisão de literatura pode ser categorizada, de acordo com sua finalidade, em 5 diferentes tipos: com intuito de desenvolvimento teórico, de avaliação validativa da teoria, de pesquisar o estado do conhecimento de um tópico particular, de identificar uma problemática e, por fim, de prover uma narrativa histórica do desenvolvimento da teoria e da pesquisa em um determinado tópico (Baumeister, 1997).

Portanto, este trabalho é categorizado como uma revisão da literatura que traz aspectos de pesquisa do estado do conhecimento, através da leitura e análise dos documentos relevantes; e aborda, consequentemente, a progressão histórica da teoria relacionada ao tema por meio da conexão linearizada entre os tópicos.

2.1. Estratégia de Levantamento

O desenvolvimento deste trabalho será dado pelo estudo das fontes primárias relevantes para a satisfação dos objetivos, como os trabalhos originais de Girard, Lafont e Taelin, além de trabalhos subsequentes que tenham relação direta com a implementação da HVM. Sendo assim, através da seguinte listagem, serão definidos os trabalhos a serem utilizados.

Autores	Ano	Trabalho	Tipo de Trabalho
GIRARD, Jean-Yves	1987	Linear Logic	Artigo Acadêmico
LAFONT, Yves.	1990	Interaction Nets	Artigo Acadêmico
LAFONT, Yves.	1995	From Proof-Nets to interaction nets.	Artigo Acadêmico
LAFONT, Yves.	1997	Interaction Combinators	Artigo Acadêmico
TAELIN, Victor.	2024	HVM2: A parallel evaluator for interaction combinators	Artigo Acadêmico

2.2. Abordagem Metodológica

A abordagem metodológica adotada será fundamentada na revisão da literatura narrativa, dada sua flexibilidade para a integração das teorias a serem tratadas e dos objetivos estabelecidos. Além disso, a revisão combina duas das categorias destacadas por Baumeister (1997) com foco na investigação do estado do conhecimento e na construção de uma linha histórica destes.

Assim, a análise tem como foco as fontes primárias, consideradas fundamentais, como os trabalhos de Girard, Lafont e Taelin; garantindo a reflexão do desenvolvimento do campo teórico, como visto na Estratégia de Levantamento (Seção 2.1). Portanto, esta abordagem disponibiliza uma visualização ampla do campo teórico e proporciona uma contextualização linear entre os tópicos abordados.

3. Desenvolvimento Teórico

Nesta seção serão revisionados os tópicos considerados de fundamental importância para a construção teórica relevante ao tema. Assim, partindo de Interaction Nets, abordando a observação de como estas estão relacionadas às Proof-Nets e a proposta da linguagem de programação; conseguinte aos principais pontos de Interaction Combinators e como estes concebem um sistema universal de computação distribuída e, por fim, a análise das características de implementação da HVM.

3.1. Interaction Nets

Em seu trabalho de nome Interaction Nets, Lafont (1990, p.1) busca demonstrar a generalização das Proof-Nets da Lógica Linear de Girard através das *nets* (grafos não direcionado com vértices nomeados, chamados de agentes) e faz isso através da proposição de uma linguagem de programação que define regras e características dos agentes para que seja possível a concepção de uma semântica de reescritura de grafos e uma disciplina de tipos. Desse modo, para melhor compreensão das Interaction Nets, é necessário uma revisão sobre o que se tratam as Proof-Nets.

Uma vez que são dadas diferentes lógicas, se faz necessário que haja uma forma de relacioná-las para que se possam ter provas equivalentes em diferentes sistemas de prova. Assim, ao observar as relações entre a dedução natural e o cálculo de sequentes, Girard (1987, p.28), observa que para a lógica linear não seria diferente, porém uma vez que esta apresenta característica construtivista, as provas desta precisam ser vistas como programas.

O núcleo das Proof-Nets, de acordo com Girard (1987, p.9), consiste no estudo do fragmento multiplicativo da lógica linear, a partir da conjunção ($\otimes$ ou *times*) e disjunção ($\parr$ ou *par*) multiplicativa (que geram *links* entre termos lógicos), de uma "partícula ideal" (que tem como tarefa percorrer toda a estrutura da prova lógica) e de switches (que são conectados aos *links* e conduzem a forma que a partícula irá percorrer estrutura de prova). Logo, é garantido que toda a estrutura lógica será percorrida, dado o conceito de viagem (Girard, 1987, p. 30), e que todas as perguntas (direção para cima no cálculo de sequentes linear) serão respondidas e todas as respostas serão questionadas (direção para baixo no cálculo de sequentes linear) garantindo a solidez da prova.

Desse modo, a partir dessa ideia inicial do que seria uma Proof-net, Girard (1987, p.9) conclui que é necessário um conjunto de métodos e regras para a geração destas de forma que seja possível demonstrar que a noção de proof-nets é equivalente à abordagem de provas dada

pelo cálculo de sequentes. Estes métodos são: Desequencialização de uma prova no cálculo de sequentes, Operações lógicas entre proof-nets e Normalização de proof-nets.

Para melhor compreensão das Interaction Nets, se faz interessante um salto cronológico para o trabalho *From Proof-Nets to Interaction Nets*, onde Lafont (1995) realiza uma explicação mais direta da relação dos conceitos que dão nome ao título de seu trabalho. Este apresenta, inicialmente, as regras de correspondência entre os conectivos da lógica linear (*times* ($\otimes$) e *par* ($\parr$)) na Figura 1, onde são dadas as configurações de negação dos elementos onde cada átomo p possui sua negação $p^{\perp}$, e as regras básicas de dedução do cálculo de sequentes Figura 2, sendo estas: *axiom*, *cut*, *times* e *par* (LAFONT, 1995, p. 225-226). Assim, a partir desses conjuntos de regras, é possível normalizar as provas no cálculo de sequente utilizando o fragmento multiplicativo da lógica linear, características estas que serão presentes nas *proof-nets* nos métodos de obtenção destas: desequencialização de uma prova no cálculo de sequentes linear, operações lógicas entre *proof-nets* e normalização (GIRARD, 1987, p. 10-12).

Figura 1 - Leis extendidas da negação da lógica linear

$$p^{\perp\perp} = p, \qquad (A \otimes B)^{\perp} = B^{\perp} \parr A^{\perp}, \qquad (A \parr B)^{\perp} = B^{\perp} \otimes A^{\perp}.$$

Fonte: Lafont (1995, p. 1)

Figura 2 - Regras de dedução

$$\frac{}{\vdash A, A^{\perp}} \qquad \frac{\vdash A, \Gamma \qquad \vdash A^{\perp}, \Delta}{\vdash \Gamma, \Delta} \qquad \frac{\vdash A, \Gamma \qquad \vdash B, \Delta}{\vdash A \otimes B, \Gamma, \Delta} \qquad \frac{\vdash A, B, \Gamma}{\vdash A \parr B, \Gamma}$$

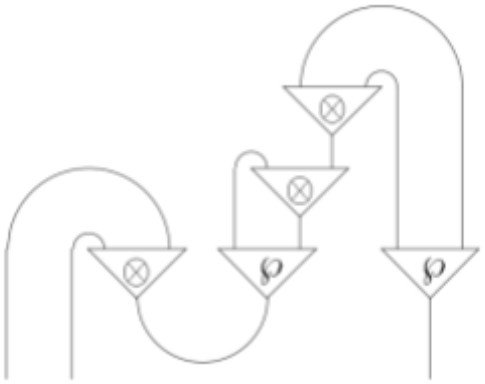
Fonte: Lafont (1995, p. 2)

Desse modo, temos que a partir dessas regras e leis, podem ser realizadas reduções em sequentes (pares de conjuntos finitos de fórmulas), que ao serem normalizadas se vêem desprovidas de termos para corte com apenas axiomas atômicos; sendo esta a característica de uma prova no cálculo linear de sequentes necessária para a consolidação da associação com proof-nets, como é descrito no primeiro teorema acerca de Proof-Nets por Girard (1987, p. 33).

A partir da consolidação dessa associação de formas de provas, agora é possível definir as provas da lógica linear com base em Proof-Nets, Lafont (1995), as apresenta como na Figura 3, com características de um grafo, diferentemente de Girard (1987) que as

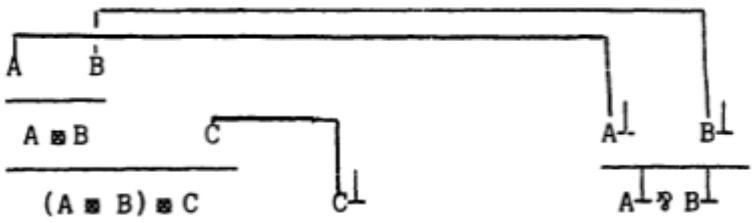
apresenta como na Figura 4, assim,já se observa uma ramificação conceitual de Proof-nets que servirá como base fundadora para a definição das Interaction Nets.

Figura 3 - Net com 3 portas livres



Fonte: Lafont (1995, p. 3)

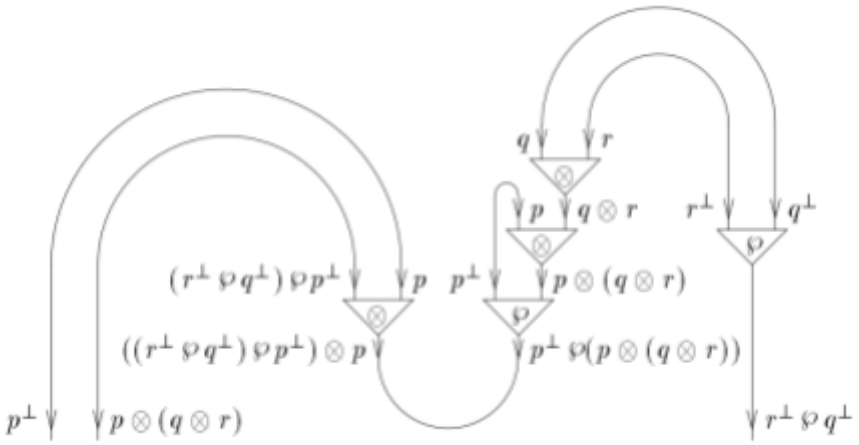
Figura 4 - Exemplo de Proof-Net



Fonte: Girard (1987, p. 35)

Na figura 3, é possível observar a presença de 2 células distintas, diferenciadas pelos conectivos lógicos ⊗ (*times*) e ⅋ (*par*), contendo 3 portas, uma principal (a de baixo) e 2 auxiliares. Além disso, se introduz o conceito de *nets* tipadas, onde existe uma seta que possui uma direção e um rótulo, que ao se inverter a direção também se inverte o equivalente lógico do rótulo. A exemplo:

Figura 5 - Net tipada



Fonte: Lafont (1995, p. 5)

Tendo visto como as *Nets* são formadas, agora se faz necessário observar a característica destas de serem reduzidas à sua forma nominal. Lafont (1995), define uma única regra de redução, vista na Figura 6.

Figura 6 - Redução de Net

Fonte: Lafont (1995, p. 6)

Com essa regra surgem dois lemas que definem a equivalência da redução de duas *nets* que possuem uma origem comum e sobre as três características que fazem com que uma *net* não seja redutível, como: possuir 2 células iguais se conectando pela porta principal, possuir um círculo vicioso (não são redutíveis) e estar na sua forma reduzida.

Além disso, é introduzido o conceito de *switching* que indica como se reduzir células par (⅋) visando reduzir a ocorrência de *nets* mal formadas e cíclicas e que leva ao critério do Teorema 1 que diz que "Uma rede é bem formada se e somente se cada *switching* define um grafo acíclico conectado" (LAFONT, 1995, p. 234). Desse modo, a redução destas se dá a partir da substituição das células par (⅋) na Figura 7 pelas configurações exibidas na Figura 8 e exemplificada como a redução da Figura 5 vista na Figura 9.

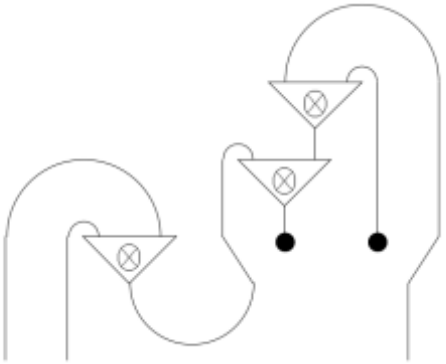
Figura 7 - Possíveis nets com célula par (⅋)

Fonte: Lafont (1995, p. 8)

Figura 8 - Configurações de redução de nets par (⅋)

Fonte: Lafont (1995, p. 9)

Figura 9 - Redução da Figura 5



Fonte: Lafont (1995, p. 9)

Antes que uma *net* seja reduzida, é necessário garantir que esta seja bem formada e para tal tarefa é realizado a análise (*parsing)* da *net*. Se levantam 2 métodos de realização desta análise; o primeiro se trata de uma análises através de um algoritmo exponencial que irá tentar todas as possibilidades da *net* e o segundo, dado por Danos and Regnier, que permite que a operação seja feita em tempo quadrático a partir da introdução do conceito de *parsing box* (LAFONT, 1995, p. 235), que abstrai as regras com intuito de aplicá-las em blocos de *nets*, ao invés do uso direto visualizado anteriormente.

É notório que tudo apresentado até o momento faz o uso dos conectivos *times* e *par*; porém, existem mais quatro conectivos não aditivos a serem debatidos: os multiplicativos unitários **1** e ⊥ (*bottom*) e os exponenciais **!** (*of course*) e **?** (*why not*). Desse modo, Lafont (1995) analisa que o uso de todos os fragmentos multiplicativos unitários sem estruturas

atômicas se trata de um problema NP-completo, fazendo com que não seja possível estender os conectivos respeitando os critérios atrelados ao Teorema 1.

Quanto aos exponenciais, se observa resultados semelhantes aos dos unitários, porém se faz possível o uso dos conectivos, apesar das reduções se tornarem mais longas devido às características das regras de dedução destes conectivos que duplicam provas. Assim, mesmo sendo conforme à propriedade de eliminação do corte, a redução de *nets* que contém estes operadores produzem formas normais fracas e que podem apresentar conflitos entre os diversos tipos de regras, apesar da característica de confluência se manter (LAFONT, 1995, p. 242).

Assim, tendo noção de todo o contexto de abstração inicial sobre as Proof-Nets, é possível que se observe as Interaction Nets com mais clareza. Recapitulando, *interaction nets* são grafos não direcionados com vértices rotulados, chamados de agentes, que interagem exclusivamente por suas portas principais (LAFONT, 1990, p. 95); estas interações ocorrem por um conjunto de regras de reescrita onde pares com portas principais conectadas são reduzidos de acordo com uma regra definida.

Figura 10 - Exemplos de agentes

Fonte: Lafont (1990, p. 95)

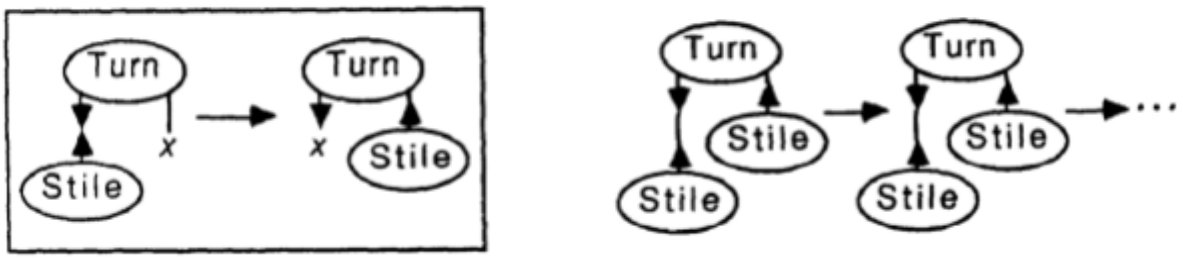
Visando garantir precisão às interações, são definidas propriedades que são indispensáveis para as regras (LAFONT, 1990, p. 95-96):

- Linearidade: Cada variável aparece exatamente duas vezes (uma no lado esquerdo e outra no direito de uma regra).
- Interação Binária: Apenas pares de agentes conectados por suas portas principais podem interagir, destacando a característica de localidade das interações.
- Não ambiguidade: Deve haver, no máximo, uma regra para cada par de símbolos distintos.

Dessa forma, a partir destas três condições são garantidas as características de confluência forte, que indica que a ordem das interações concorrentes é irrelevante para o

resultado final, e da otimização, que indica que os membros direitos das regras não devem conter pares vivos de agentes, evitando conflitos ou loops infinitos.

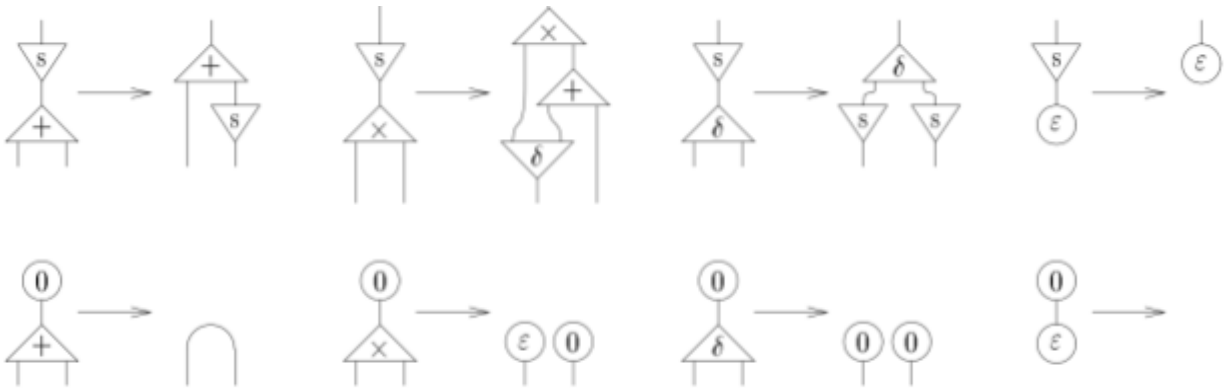
Figura 11 - Exemplos de configuração de loop infinito



Fonte: Lafont (1990, p. 98)

A exemplo de aplicação, temos o sistema de aritmética unária partindo da definição de regras de interação:

Figura 12 - Regras de aritmética unária



Fonte: Lafont (1995, p. 19)

Como dito anteriormente, outra proposta é a disciplina de tipagem que garante computações locais corretas e previne *deadlocks* globais. Esta é dada pela definição do tipo de cada porta que um agente contém, mantendo assim consistência em todas as conexões da rede, além de definir que as regras devem preservar a tipagem durante a redução (LAFONT, 1990, p. 100).

Figura 13 - Agentes tipados

Fonte: Lafont (1990, p. 100)

Outro fator relevante é a categorização das *nets* como simples ou semi-simples, atribuída a partir da ausência de ciclos viciosos. Junto a isso, é definido o conceito de partições das portas auxiliares dos agentes como forma de distinguir os ciclos viciosos bons (alguma forma de recursão) dos ruins (*deadlock*), esse conceito auxilia nesta categorização a partir de um conjunto de operações (LINK, CUT, GRAFT, EMPTY e MIX); uma *net* gerada através destas operações é tida como simples (LAFONT, 1990, p. 102).

Por último, Lafont introduz a sintaxe de programação em *Interaction Nets*, que é estruturada em três componentes fundamentais: a declaração de tipos, a definição de símbolos e as regras de interação. Cada símbolo possui uma tipagem explícita para sua porta principal e auxiliares, assegurando coerência nas interações. No caso de partições não discretas, utiliza-se a notação de chaves para agrupar portas auxiliares de mesmo tipo, como ocorre com Dupl, que associa suas saídas ao tipo nat+. A notação das regras de interação, embora pouco convencional à primeira vista, reflete a estrutura gráfica das nets, organizando portas principais e auxiliares de maneira hierárquica. A aplicação das regras segue um padrão de correspondência entre variáveis nos membros esquerdo e direito, garantindo que as conexões sejam preservadas ao longo das transformações (LAFONT, 1990, p. 104-106).

Portanto, para Lafont (1990), as *Interaction Nets* se definia como uma generalização das *Proof-Nets* e através destas concretizam um modelo computacional pragmático portador de interações locais, sem necessidade de sincronização global e com forte confluência, evidenciando características fundamentais para a programação paralela. Além disso, a partir de uma disciplina de tipos para o tratamento de deadlocks e ciclos viciosos assegurou-se mais ainda a precisão das *nets* ao decorrer das interações.

3.2. Interaction Combinators

O nascimento teórico acerca de Interaction Combinators pode ser resumido como uma generalização das Interaction Nets a partir das suas características obtidas das restrições

definidas para a construção de seus agentes e regras de interação, como o paralelismo provido pela localidade das interações e a forte confluência; isso, junto das questões: “quais são as leis fundamentais da computação?” (LAFONT, 1997, p. 69) e “existe um programa universal capaz de traduzir qualquer outro programa existente?” (LAFONT, 1998, p. 3).

É possível observar as comparações de interaction nets com outros modelos como autômatos celular, connection graphs e máquinas de Turing em um trabalho de Lafont de 1994 denominado The Paradigm of Interaction, que no período de publicação de From Proof-Nets to Interaction Nets ainda era um trabalho em preparação e que possivelmente veio a se tornar o trabalho Interaction Combinators, em 1997, este que será o objeto focal da revisão deste tópico.

As questões acerca da existência de um sistema universal e das leis da computação se entrelaçam na necessidade de entender qual o núcleo da computação com fim de projetar tal sistema capaz de traduzir programas de outros modelos computacionais. Assim, Lafont observa que as leis fundamentais da computação variam conforme o modelo computacional abordado, e ao observar tais modelos é possível observar características a partir dos pontos de vista de computabilidade e computação, termos que podem ser abordados como a capacidade de modelos solucionarem o mesmo problema e como essa execução será dada (a exemplo o paralelismo), respectivamente (LAFONT, 1997, p. 69-70).

Durante esta observação de modelos computacionais são introduzidos os conceitos de tradução de sistemas de interação, sendo a tradução preservante das características essenciais de um determinado modelo computacional, e de um sistema de interação universal, este sendo caracterizado pela propriedade de que qualquer outro sistema computacional pode ser traduzido para ele. A exemplo, as Turing machines podem ser categorizadas como sistemas de interação mas não um sistema universal no sentido abordado, uma vez que sua característica sequencial é limitante de certa forma. Por fim, em conclusão às análises dos modelos são definidas como as duas leis fundamentais da computação: a comutação e a aniquilação (LAFONT, 1997, p. 70).

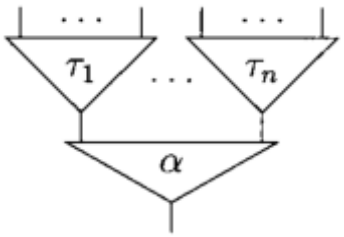
Desse modo, com a tarefa de provar a existência de tal sistema de interação universal, é proposto e provado por Lafont o sistema de *interaction combinators*. Tal sistema é obtido a partir de trabalhos prévios, também desenvolvidos a partir da lógica linear e de *proof-boxes*, e tem como intuito de reduzir a complexidade do modelo computacional mantendo expressividade e parte da generalização de *interaction nets*.

3.2.1. Revisão de *Interaction Nets*

Partindo dessa generalização, Lafont inicializa uma revisão às interaction nets que irá abordar os conceitos de *nets*, interações, exemplos, reduções e traduções. Partindo do conteúdo de *Interaction Nets* (LAFONT, 1990), são introduzidos novas generalizações de *nets* como *trees* (*nets* com apenas 1 porta livre), representadas na Figura 14, e *wiring* (*net* sem células e sem ciclos), representadas na Figura 15; além disso são definidas as características fundamentais de uma *net* (LAFONT, 1997, p. 72), da seguinte forma:

- um conjunto finito X (de portas livres),
- um conjunto finito C (de células),
- um símbolo *l*(c) para cada c ∈ C,
- um conjunto finito W (de *wires*),
- um conjunto ∂w de 0 ou 2 portas para cada w ∈ W

Figura 14 - Representação de uma tree



Fonte: Lafont (1997, p. 72)

Figura 15 - Representação de um wiring

Fonte: Lafont (1997, p. 72)

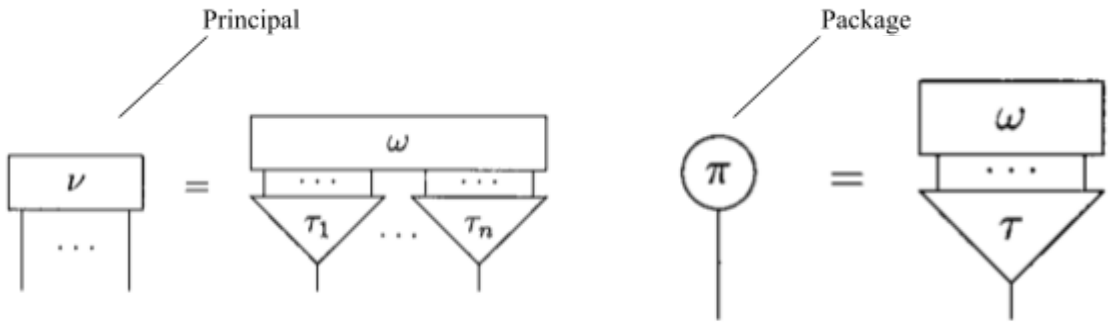
Prosseguindo, se instrui o conceito de *interaction system* como um conjunto de regras de interação que pode ser aplicado sem ambiguidades conservando os princípios da comutatividade e da identidade (LAFONT, 1997, p. 73). Com o contexto, são levantadas proposições acerca das características de forte confluência e da decomposição de *nets* reduzidas contendo portas livres e no meio disso demonstradas possíveis traduções de sistemas como Turing machines, Cellular automata e Aritmética unária.

Todos os exemplos de tradução são acompanhados de regras de interação que permitem a replicação do modelo sobre as *interaction nets*, entretanto na abordagem da aritmética unária é onde se observa a introdução dos novos agentes (símbolos), devido às necessidades de representação desta aplicação, estes são o **δ** (*duplicator*) de aridade 2 e o ε (*eraser*) de aridade 0, onde o termo aridade se refere ao número de portas auxiliares de cada agente, estes serão abordados com maior detalhe posteriormente.

Sabe-se que a computação ocorre através das reduções das *nets*, sendo assim, Lafont (1997) revisita as categorias quanto à redutibilidade: as *nets* redutíveis e as irredutíveis. Assim, é definido que existe um caminho principal pelo qual será realizada a redução, este tem comprimento n e consiste de um número n de células onde a porta principal da anterior se conecta à uma auxiliar da célula posterior. Logo, uma *net* está reduzida quando possui nenhum corte (células principais interagindo) e nenhum ciclo vicioso (LAFONT, 1997, p. 78-79), fator que recorre às definições das regras de interação, as quais os membros à direita devem ser *nets* reduzidas, caso não sejam, há a possibilidade de falha na definição da regra.

Ainda a respeito da redutibilidade, se existe uma *net* reduzida, esta pode ser declarada como principal (*nets* sem portas livres ou com 1 ou mais portas livres) ou package (*nets com* apenas uma porta livre), ilustrado na Figura 16.

Figura 16 - Principal e Package nets



Fonte: Lafont (1997, p. 79-80)

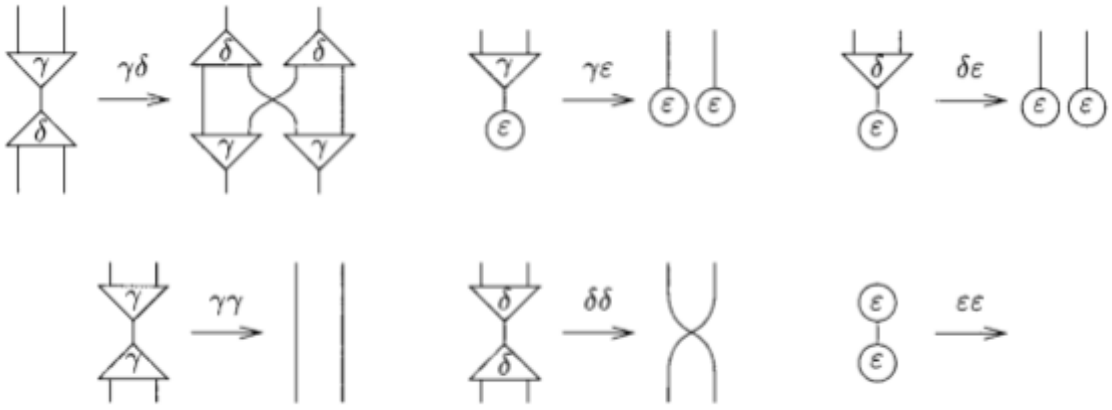
Por fim, na reintrodução das *interaction nets*, são abordadas as traduções. É definido como tradução, por Lafont (1997, p. 80), que: "Sejam Σ para Σ' alfabetos de símbolos com aridades. Uma tradução ϕ de Σ para Σ' mapeia cada símbolo α de aridade n em Σ para uma *net* principal $\phi(\alpha)$ de aridade n, construída com símbolos de Σ'". Desse modo, são introduzidas proposições acerca da equivalência da redução entre *nets* traduzidas e suas

correspondentes originais e sobre a relação da extensão da redução entre tais *nets* (LAFONT, 1997, p. 80-81).

3.2.2. Introdução do Sistema de *Interaction Combinators*

A partir desta recapitulação das *interaction nets* e introdução de novos conceitos, se inicia a introdução do sistema de *interaction combinators*. Este é definido por 3 símbolos, chamados de *combinators*: **γ** (constructor) e **δ** (*duplicator*) de aridade 2, e ε (*eraser*) de aridade 0; além de um sistema de 6 regras, podendo ser de comutação (interação de símbolos diferentes) ou de aniquilação (interação de símbolos iguais), que definem as interações (LAFONT, 1997, p. 82), como observado na Figura 17. Além disso, em decorrência das regras pode-se observar que há cenários onde não é possível terminar a redução da *net*, gerando loops infinitos (LAFONT, 1997, p. 82).

Figura 17 - Regras de interação para os combinators



Fonte: Lafont (1997, p. 81)

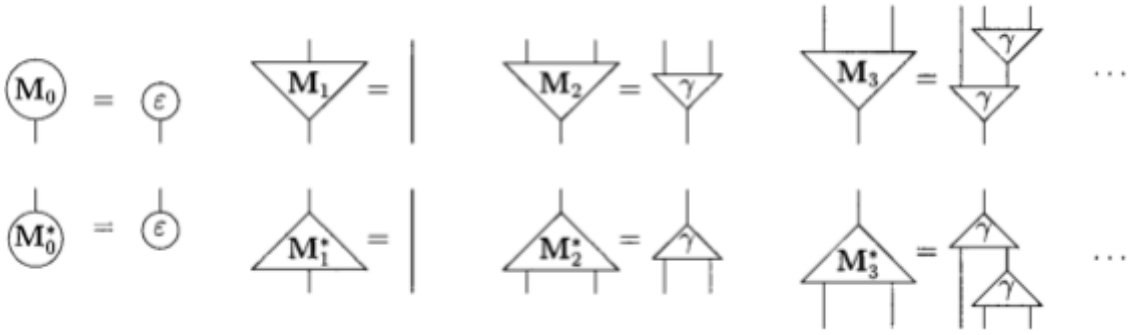
Como forma de garantir a universalidade do sistema, assim garantindo o primeiro teorema definido por Lafont (1997, p. 82) como: "Qualquer sistema de interação pode ser traduzido para o sistema de *interaction combinators*", temos que são introduzidos abstrações extras, também construídas acima da lógica linear. De forma a sumarizar as novas abstrações, temos que:

> Visando provar isto, nós precisamos introduzir inúmeras construções que são inspiradas pelas regras da lógica linear: os *multiplexors* e os *transpositors* (inspirados pelas regras multiplicativas), os *menus* e os *selectors* (inspirados pelas regras aditivas), e os *codes*, *copiers*, e o *decoder* (inspirados pelas regras exponenciais) (LAFONT, 1997, p. 82).

Os *multiplexors* (Figura 18) são introduzidos como abstrações compostas por *constructors* e *erasers* dadas as características da regra de interação **γγ** de disposição das

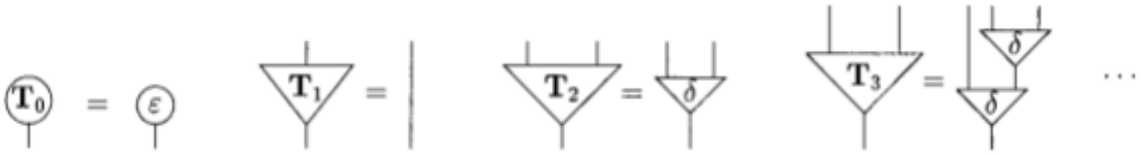
conexões após a interação. Da mesma forma, são introduzidos os *autodual multiplexors* (Figura 19) e sua subcategoria: os *transpositors*, que através da regra de interação **δδ** consegue realizar transposições, ou permutação involutiva, das portas da *net* (LAFONT, 1997, p. 82). Já os *menus* e *selectors* (Figura 20), têm seus funcionamentos baseados na conjunção do conceito de *multiplexors* junto aos *packages* para criação dos *menus* e junto aos *erasers* para o mecanismo de seleção de determinado *package* (LAFONT, 1997, p. 84).

Figura 18 - Implementação dos multiplexors



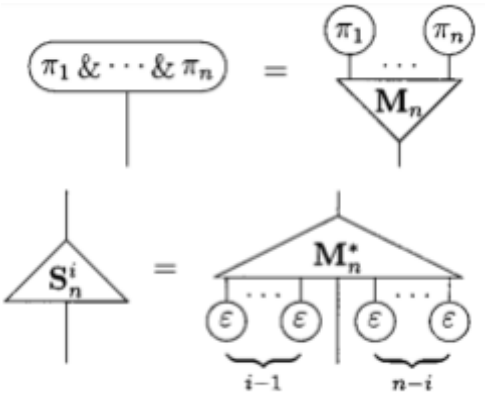
Fonte: Lafont (1997, p. 83)

Figura 19 - Implementação dos autodual multiplexors



Fonte: Lafont (1997, p. 83)

Figura 20 - Implementação dos menus e selectors



Fonte: Lafont (1997, p. 83)

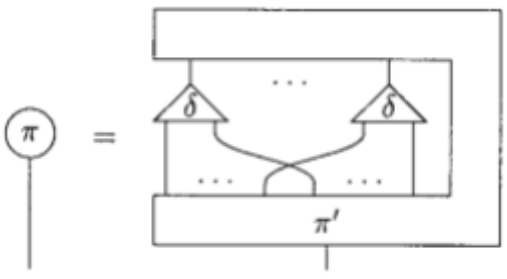
Nesse ponto, parte do mecanismo de tradução de sistemas é definido em um caso restrito, onde são utilizados apenas de *multiplexors*, *transpositors*, *selectors* e *menus* através

da demonstração de tradução de uma célula e regra utilizando apenas as abstrações citadas. Em suma, esta demonstração se dá a partir da representação de uma *net* reduzida em outras duas *nets*, categorizadas pelas aridades de seus símbolos contidos, conectadas a partir de uma permutação de suas portas e de certa forma se pode representar este arranjo utilizando das construções citadas anteriormente (LAFONT, 1997, p. 85-87).

Antes da definição dos constructos restantes, é levantada a questão sobre a possível duplicação de *packages* chegando à conclusão que somente estes que não possuem *duplicators* podem ser duplicados, devido à regra de interação **δδ**. Porém, esta característica é um ponto relevante para o todo, uma vez que a duplicação de *nets* não pode ser dada de forma arbitrária se faz necessário que surjam novos elementos capazes de realizar essa duplicação (LAFONT, 1997, p. 87-88).

Assim, em função da necessidade de se copiar *packages* que contenham **δ** são definidos os conceitos de *codes*, *copiers* e *decoders*. Lafont (LAFONT, 1997, p. 88) introduz o termo *code* da seguinte forma: "Constrói-se, para qualquer pacote **π**, outro pacote **!π** (o *code* de **π**) que pode ser duplicado, e do qual **π** pode ser extraído."; pode-se dizer que **!π** se trata de um *package* sem a presença de símbolos **δ** em sua construção, ilustrado na Figura 21. A partir desse conceito de *code*, *copiers*, representados pelo símbolo C_n de aridade n, podem ser formados para copiar **!π** sem problemas relacionados à **δ** e *decoders*, representados pelo símbolo D de aridade 1, podem ser formados para extrair **π** de **!π** (LAFONT, 1997, p. 88); ambas abstrações com suas características representadas na Figura 22 e o conceito de *code* na Figura 21..

Figura 21 - Isolamento de **δ** em **π**



Fonte: Lafont (1997, p. 88)

Figura 22 - Propriedades de copiers e decoders

Fonte: Lafont (1997, p. 88)

Por fim, são demonstradas através da tradução de uma célula, como se dão as traduções de caso geral a partir da empregabilidade em conjunto das abstrações definidas; além disso, quanto à minimalidade do sistema, são levantados questionamentos sobre as necessidades das regras de interação, que acabam levando à permanência da universalidade mesmo removendo a regra **δ**ε (LAFONT, 1997, p. 89-90)

3.2.3. Semântica dos *Combinators*

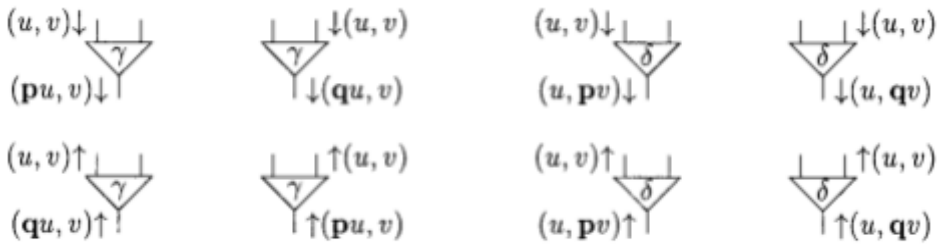
Dado o sistema de *Interaction Combinators*, a forma que este será interpretado é dado por Lafont (1997, p. 90) da seguinte forma: "... nós interpretamos as *nets* de *combinators* como máquinas reversíveis de pilha dupla. Essa interpretação traz consigo noções de equivalência nas *nets* de *combinators*, e sugere um sistema de *combinators* direcionados.". Neste tópico serão abordados como principais tópicos a execução da *net* ao decorrer da sua redução, a equivalência entre as *nets*, a formulação algébrica desta execução e os *combinators* direcionados.

O conceito de execução é dado a partir da ideia de se computar uma *net* sem propriamente reduzi-la, utilizando de uma forma similar do conceito da "partícula ideal" encontrada em *Proof-nets*. Assim, temos que ocorrerá uma "viagem" ao longo da *net* e esta será atribuída de um par *(u,v)*, correspondendo às pilhas (uma para *constructors* e outro para os *duplicators*), e este par conterá uma *string* com elementos contidos no alfabeto {p, q}, onde tais elementos corresponderão como marcadores às portas auxiliares (esquerda (p) ou direita (q)) dos *combinators* (LAFONT, 1997, p. 91).

A viagem é definida através de duas regras (LAFONT, 1997, p. 91), que estão ilustradas na Figura 23:

- se alguém entra em uma célula por uma porta auxiliar, ele empurra a letra correspondente para a pilha relevante e ele sai pela porta principal;
- se alguém entra em uma célula pela porta principal, alguém retira p ou q da pilha relevante, e alguém sai pela porta auxiliar correspondente (no caso de uma célula **δ**) ou pela outra (no caso de uma célula **γ**). Se a pilha relevante estiver vazia, alguém para.

Figura 23 - Regras de execução

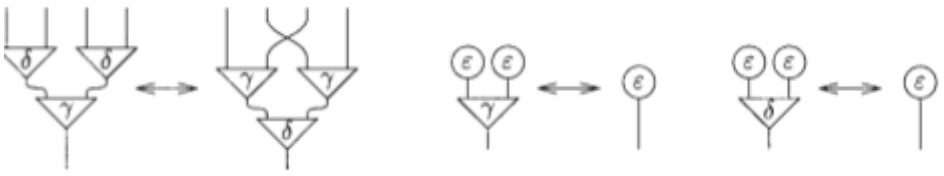


Fonte: Lafont (1997, p. 91)

Outro ponto levantado é o da ideia de uma transição de estados, podendo este ser escrito na forma (x, u, v), onde x denota um marcador de uma porta livre e (u, v) as pilhas, podendo chegar no estado (x', u', v'); também se tem o conceito de ū que é uma *string* obtida a partir da troca de p por q em u (LAFONT, 1997, p. 91). Além disso, são definidas propriedades da execução que garantem determinismo, reversibilidade, monotonicidade e invariância, onde as 3 primeiras são garantidas pelas regras de definição da execução e a última é dada pela verificação da invariância de cada regra do sistema de *Interaction Combinators* ao decorrer da execução (LAFONT, 1997, p. 92-93).

Quanto à equivalência, este conceito é definido a partir da ideia de se estabelecer quando duas *nets*, com as mesmas portas livres, podem ser consideradas estruturalmente idênticas, utilizando um formalismo semelhante ao encontrado em *Proof-nets*. Assim, temos que a equivalência é caracterizada por uma transformação sistemática entre as redes, guiada pelas regras apresentadas na Figura 24, e que preserva suas transições. As portas livres são classificadas em passivas ou ativas (γ, δ ou ε), dependendo das conexões que aceitam, onde essas classificações são fundamentais para determinar as propriedades das redes. Esse processo assegura que a equivalência pode ser avaliada diretamente, sem ambiguidade, garantindo que redes equivalentes compartilham o mesmo comportamento funcional (LAFONT, 1997, p. 94).

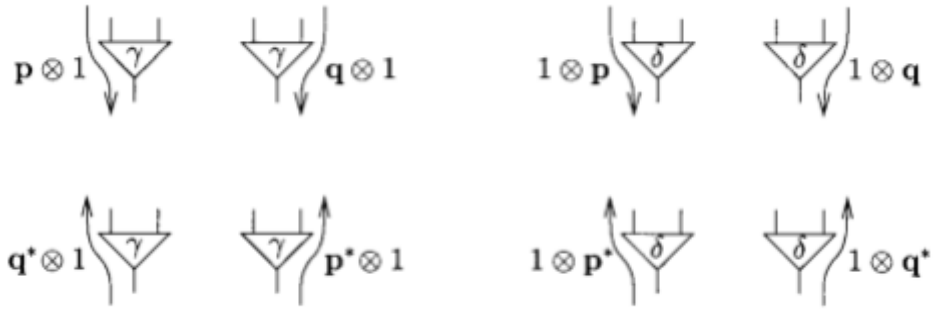
Figura 24 - Regras de equivalência



Fonte: Lafont (1997, p. 94)

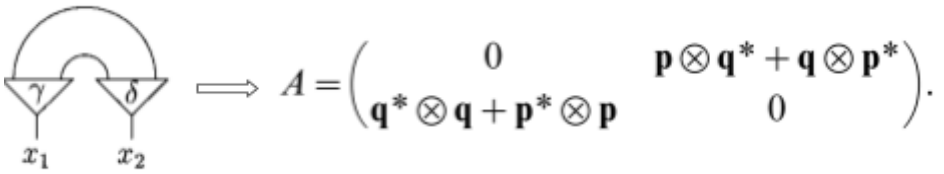
Já a formulação algébrica, introduz uma estrutura formal para a descrição e manipulação do sistema apresentado. Para esta tarefa é introduzido um anel não comutativo contendo 4 símbolos p e q, sendo positivos, e p* e q* sendo negativos; assim, são definidas equações básicas de relacionamento entre tais símbolos que implicam na redução destes levando à uma forma monominal algébrica ou ao 0 (LAFONT, 1997, p. 94-95). De certa forma, estas equações acabam por corresponder às regras de interação, tanto de aniquilação quanto de comutação, ao considerarmos a extensão destas sobre o modelo de pilhas de execução, tendo sua interpretação demonstrada na Figura 25. Além disso, podem ser geradas matrizes sobre os caminhos possíveis entre os estados de execução, visto na Figura 26, e estas matrizes podem ser utilizadas para a decomposição de *nets* seguindo um grupo de regras definidas (LAFONT, 1997, p. 96).

Figura 25 - Interpretação algébrica



Fonte: Lafont (1997, p. 95)

Figura 26 - Interpretação matricial de caminhos

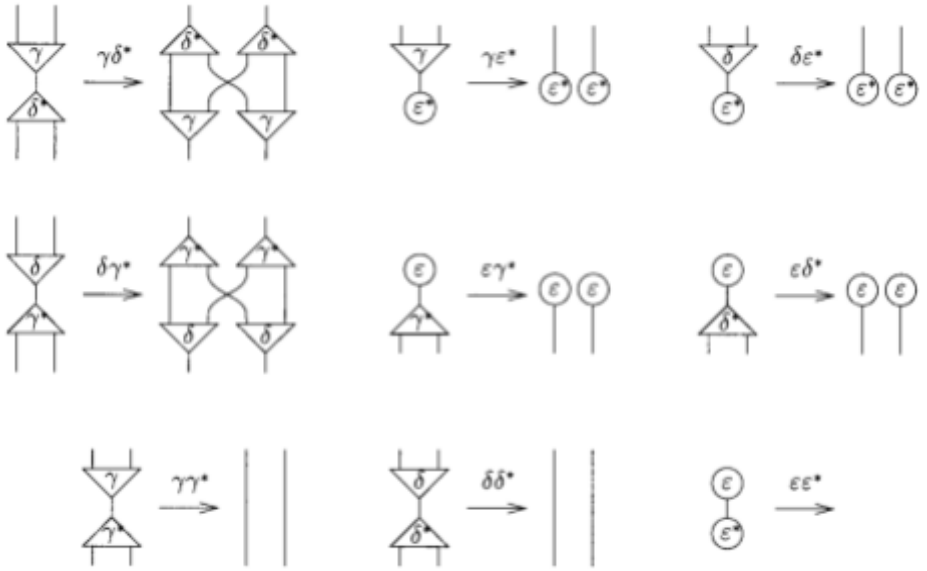


Fonte: Lafont (1997, p. 94)

Por fim, os *combinators* direcionados surgem a partir de um problema da reversibilidade da execução que não é equivalente nas duas direções, característica que é ressaltada pelo anti-automorfismo, observado nas matrizes de *nets* reduzidas geradas a partir da formulação algébrica (LAFONT, 1997, p. 96). Assim, se introduz um novo sistema, dessa vez de *combinators* direcionados com 6 símbolos (os prévios mais suas "negações") e 9 regras (Figura 27), possuindo a mesma expressividade do sistema não direcional mas perdendo parte de sua universalidade. Além disso, um conceito importante desse sistema é introduzido: a cobertura (Figura 28), que auxilia na conversão entre os dois sistemas apesar

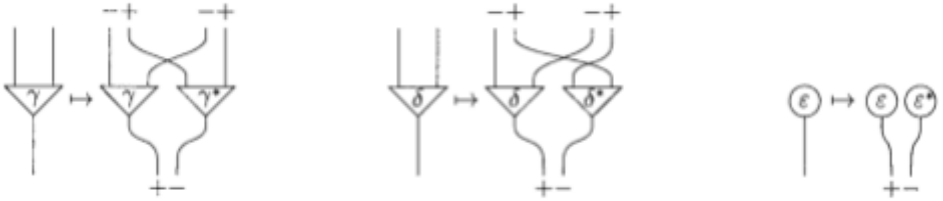
de não ser considerado uma tradução concreta, tomando uma *net* v com n portas existe ṽ, composta por *combinators* direcionados, com 2n portas rotuladas com símbolos + e -, onde uma porta + sempre se conecta com uma porta - (LAFONT, 1997, p. 96).

Figura 27 - Regras de interação para os combinators direcionados



Fonte: Lafont (1997, p. 97)

Figura 28 - Cobertura de células



Fonte: Lafont (1997, p. 98)

3.3. HVM

A HVM surge como um avaliador que se desprende das estruturas convencionais de computação e favorece o uso de Interaction Combinators. Vale ressaltar que, o trabalho abordado nesta revisão se trata sobre a HVM2, a versão de número dois da HVM. Ao se aproveitar dos ganhos matemáticos do modelo utilizado, a HVM2 obtém um paralelismo massivo com necessidade mínima de sincronização, com programas que conseguem crescer, se dividir e serem resolvidos como entidades autônomas ao invés de uma resolução linear.

Além disso, outras características a serem destacadas da HVM2 são: a possibilidade do uso tanto de CUDA Kernels quanto de procedures em C para a implementação dos Combinators, a presença de variáveis atômicas, um sistema com definições globais e números

nativos (TAELIN, 2024, p. 1). Desse modo, este tópico trará como foco a observação de como a HVM2 traduz a teoria em prática.

3.3.1. Sintaxe

As linguagens de programação são frequentemente moldadas pelos modelos computacionais que incorporam, e na HVM2, a sintaxe é mais do que um meio de representação, é uma expressão direta da computação paralela vide as características dos Interaction Combinators. Diferente das linguagens convencionais que dependem de fluxo de controle explícito, o HVM2 é baseado no Cálculo de Interação, por Fernández e Mackie (1999), um sistema textual projetado para representar Interaction Combinators de uma forma que pode ser reescrita mecanicamente (TAELIN, 2024, p. 3). Esse design não é meramente teórico; ele garante que a computação seja inerentemente descentralizada e distribuída, permitindo execução paralela sem a necessidade de primitivas de concorrência explícitas (TAELIN, 2024, p. 4).

Em seu núcleo, os programas são organizados como estruturas semelhantes a árvores, onde a computação surge das interações entre nós e variáveis (TAELIN, 2024, p. 3). O sistema define sete principais tipos de nós, listados na Figura 29, cada um com uma função distinta: erasers (*), references (@), valores numéricos, constructors ((a b)), duplicators ({a b}), operators ($(a b)) e switches (?(a b)); junto ao *node* extra *variable* (TAELIN, 2024, p. 3-4). Além disso, temos a introdução do conceito de *Redex, que* representa a conexão de 2 portas principais de árvores distintas, e do conceito de *Net* que é definido por uma árvore e uma lista de *redexes* separados pelo símbolo “&”; por fim, o conceito de *Book* que consiste no conceito de uma definição *top-level* ou uma *Net* nomeada (TAELIN, 2024, p. 3).

Figura 29 - Sintaxe da HVM2

```
<Node> ::=
    | "*"                     -- (ERA)ser
    | "@" <alphanumeric>      -- (REF)erence
    | <Numeric>               -- (NUM)eric
    | "(" <Tree> <Tree> ")"   -- (CON)structor
    | "{" <Tree> <Tree> "}"   -- (DUP)licator
    | "$(" <Tree> <Tree> ")" -- (OPE)rator
    | "?(" <Tree> <Tree> ")" -- (SWI)tch

<Tree> ::=
    | <alphanumeric>          -- (VAR)iable
    | <Node>

<alphanumeric> ::= [a-zA-Z0-9_.-/]+
```

Fonte: Taelin (2024, p. 3)

Um fator interessante advindo da sintaxe é o fator que as conexões de portas auxiliares para portas principais são implícitas, trazendo um ganho de eficiência no armazenamento dos nós, devido à estrutura de árvore do sistema, enquanto as conexões entre portas auxiliares devem ser explícitas, sendo dadas através de nós VAR, que estão sempre em pares representando um *wiring* entre dois elementos. Outro ponto é o do uso de nós que não constam na definição básica dada por Lafont dos *Combinators*, sendo estes REF(ERENCE), NUM(ERIC), OPE(RATOR) e SWI(TCH), todos estes são introduzidos devido à questões de performance, onde a emulação destes através de *Interaction Combinators* seria mais custosa/ineficiente (TAELIN, 2024, p. 5).

3.3.2. Interações

Na HVM2, a computação se desdobra por meio de um sistema de dez interações fundamentais, cada uma governando como os nós evoluem e se resolvem dentro da *net*. Essas interações definem as regras de transformação, garantindo que a computação progrida de forma eficiente enquanto mantém a natureza paralela e determinística do sistema (TAELIN, 2024, p. 6). Diferente dos paradigmas de programação tradicionais que dependem de sequenciamento explícito, o HVM2 permite que a computação surja organicamente a partir de interações locais entre *nós* conectados, tornando a execução inerentemente escalável em múltiplas threads.

As interações mais básicas lidam com a vinculação de variáveis e a aplicação de funções. A interação *Link* garante que as variáveis permaneçam consistentes ao substituir suas ocorrências na rede, enquanto a interação de *Call* expande references (@) em suas definições correspondentes, permitindo aplicações estruturadas de funções sem a necessidade de fluxo de controle explícito (TAELIN, 2024, p. 6). Juntamente a essas, as interações de *Void* e *Erase* desempenham um papel de coleta de lixo, eliminando estruturas redundantes e garantindo que a computação não acumule *nós* desnecessários (TAELIN, 2024, p. 7).

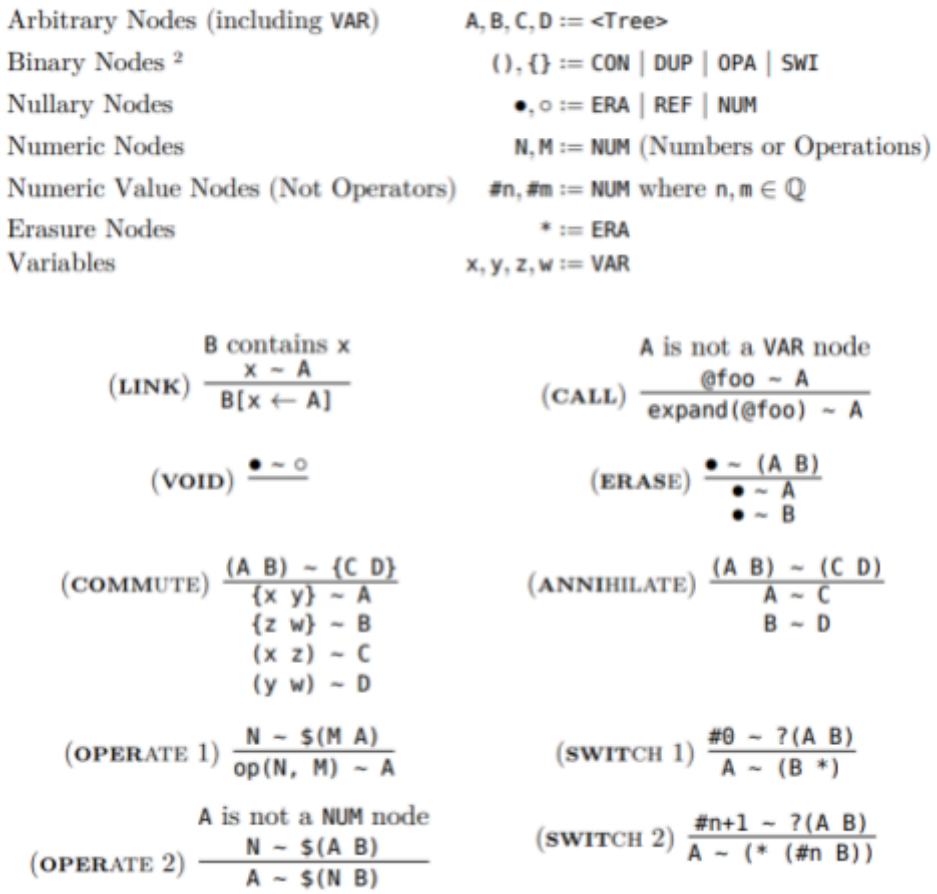
Interações mais complexas impulsionam o comportamento computacional central. A interação de *Commute* permite que diferentes tipos de nós binários troquem de posição, garantindo que as transformações ocorram de maneira ótima e estruturada. A de *Annihilate*, por outro lado, permite que dois nós idênticos se cancelem mutuamente, um processo análogo à redução de funções no cálculo lambda (TAELIN, 2024, p. 7). Essas interações permitem

que o HVM2 execute operações de forma eficiente, reduzindo estruturas de maneira incremental e paralela, em vez de seguir uma ordem de avaliação estrita.

Operações numéricas e ramificações condicionais são tratadas pelas interações de operate e switch, que permitem cálculos aritméticos e controle de fluxo lógico enquanto preservam a natureza paralela da rede (TAELIN, 2024, p. 8). Essas interações garantem que computações envolvendo números, que frequentemente geram um grande gargalo na execução paralela, sejam executadas com o mínimo de sobrecarga, aproveitando operações nativas da máquina sempre que possível.

Ao estruturar a computação em torno dessas interações, de forte confluência, a HVM2 alcança um paralelismo massivo, distribuindo eficientemente o trabalho por milhares de threads enquanto preserva a execução determinística. Em conclusão às interações, podemos observar as definições das interações a partir da definição de regras em estilo Gentzen na Figura 30, além disso é apresentada uma tabela de interação, Figura 31, onde é possível ver as decisões das interações a serem utilizadas com base no tipo dos nós a serem reduzidos.

Figura 30 - Definição das regras de interação



Fonte: Taelin (2024, p. 6)

Figura 31 - Tabela de Interações

A\B	VAR	REF	ERA	NUM	CON	DUP	OPR	SWI
VAR	LINK	LINK	LINK	LINK	LINK	LINK	LINK	LINK
REF	LNIK	VOID	VOID	VOID	CALL	ERAS	CALL	CALL
ERA	LINK	VOID	VOID	VOID	ERAS	ERAS	ERAS	ERAS
NUM	LINK	VOID	VOID	VOID	ERAS	ERAS	OPER	SWIT
CON	LINK	CALL	ERAS	ERAS	ANNI	COMM	COMM	COMM
DUP	LINK	ERAS	ERAS	ERAS	COMM	ANNI	COMM	COMM
OPR	LINK	CALL	ERAS	OPER	COMM	COMM	ANNI	COMM
SWI	LINK	CALL	ERAS	SWIT	COMM	COMM	COMM	ANNI

Fonte: Taelin (2024, p. 8)

3.3.2. Localidade

A questão da substituição de variáveis em sistemas computacionais paralelos é um problema central na literatura de execução concorrente, uma vez que a manipulação de estados compartilhados frequentemente exige mecanismos de sincronização e controle de acesso. No contexto da HVM2, essa ideia é explorada por meio do *Substitution Map* e *Atomic Linker*, um sistema que permite que variáveis sejam resolvidas de maneira atômica e descentralizada, evitando bloqueios globais e maximizando o aproveitamento dos múltiplos núcleos de processamento disponíveis (TAELIN, 2024, p. 9).

Na abordagem adotada na HVM2 a substituição ocorre de maneira descentralizada, por meio de um mapa de substituições global, onde cada variável, ao ser vinculada a um valor, é armazenada temporariamente até que sua segunda ocorrência seja processada (TAELIN, 2024, p. 10). Esse modelo evita a necessidade de percorrer grafos complexos ou manipular pilhas de execução, garantindo maior eficiência em sistemas de computação distribuída.

Outro aspecto relevante a ser analisado é a ausência de bloqueios explícitos no mecanismo de substituição da HVM2. Tradicionalmente, linguagens e sistemas funcionais concorrentes, como GHC e Erlang, fazem uso de estruturas de sincronização, como MVars ou STM (Software Transactional Memory) para garantir a coerência do estado compartilhado (HARRIS et al., 2005). Esses métodos, embora eficazes para certas classes de aplicações, podem apresentar penalidades de desempenho quando escalados para milhares de threads, devido ao custo associado ao controle de transações ou ao acesso concorrente a estruturas de memória protegidas. Em contraste, a solução proposta por Taelin (2024, p. 9-11) elimina a necessidade de bloqueios globais ao adotar um procedimento atômico de vinculação de

variáveis, no qual a primeira aparição de uma variável é registrada e a segunda automaticamente vinculada, sem interferência externa.

Além das vantagens práticas dessa abordagem, há implicações teóricas significativas. A confluência garantida do sistema implica que a ordem na qual os *redexes* são processados não altera o resultado final, um princípio central dos sistemas de reescrita de *Interaction Nets*, conforme demonstrado por Lafont (1997). Isso significa que, diferentemente de outros modelos concorrentes que exigem um rigoroso controle de dependências, a HVM2 permite que as substituições ocorram de forma assíncrona e independente, sem a necessidade de um scheduler global para coordenar a ordem das interações.

Contudo, essa abordagem não está isenta de desafios. A implementação do Substitution Map exige uma estrutura de memória suficientemente eficiente para suportar acessos atômicos de alta frequência, o que pode se tornar um gargalo em arquiteturas com limitações de largura de banda ou acesso desigual à memória global, como ocorre em certos modelos de execução em GPUs (TAELIN, 2024, p. 19). Além disso, a dependência de um esquema de substituição puramente linear pode trazer limitações ao lidar com padrões de duplicação não triviais, um problema já observado em sistemas de reescrita baseados em *Interaction Combinators* (MAZZA, 2007).

3.3.3. Números e Operações

A manipulação eficiente de números em sistemas de reescrita de grafos é um desafio recorrente na literatura sobre avaliação funcional e concorrente. Modelos tradicionais, como os números de Church, embora universais, impõem um custo computacional elevado devido à necessidade de expansões sucessivas para operações simples. Na HVM2, a abordagem adotada busca equilibrar expressividade e desempenho, introduzindo um sistema de números nativo que se integra diretamente ao modelo de Interaction Combinators, garantindo operações aritméticas rápidas e compatíveis com a execução paralela massiva (TAELIN, 2024, p. 12).

Os números na HVM2 são representados por um *node* específico da linguagem (NUM), que encapsula tanto valores numéricos quanto operadores aritméticos. Para maximizar a eficiência, a estrutura de um nó numérico é compacta: um campo de 5 bits para a tag de tipo e um payload de 24 bits para armazenar o valor associado, permitindo a manipulação direta de números inteiros e de ponto flutuante sem a necessidade de estruturas auxiliares (TAELIN, 2024, p. 12). A especificação inclui três tipos numéricos principais:

inteiros sem sinal (U24), inteiros com sinal (I24) e números de ponto flutuante (F24), cada um deles projetado para operar diretamente dentro dos limites de largura de palavra da arquitetura subjacente (TAELIN, 2024, p. 13).

A codificação numérica em HVM2 segue padrões modernos, com três formatos principais: U24, para inteiros sem sinal de 24 bits (0 a 16.777.215); I24, para inteiros com sinal usando complemento de dois (-8.388.608 a 8.388.607); e F24, um ponto flutuante baseado no IEEE 754 binary32, com menor precisão na mantissa, mas ainda adequado para cálculos numéricos eficientes (TAELIN, 2024, p. 13-14).

A HVM2 incorpora operações aritméticas embutidas, como adição, subtração, multiplicação, divisão e comparação, utilizando *opcodes* numéricos compactos para cálculos diretos e eficientes (TAELIN, 2024, p. 14). Uma característica única é a aplicação parcial de operadores, que podem existir como nós independentes na rede de interação, aguardando um segundo operando para serem resolvidos, permitindo a construção modular e paralelizável de expressões matemáticas (TAELIN, 2024, p. 15). A integração do sistema numérico ao modelo de interação garante a distribuição de cálculos sem comprometer a confluência, já que números são tratados como nós e as operações seguem regras fixas, eliminando ambiguidades e permitindo execução simultânea em diferentes partes da estrutura (TAELIN, 2024, p. 15).

3.3.4. Arquitetura

A eficiência da memória é um fator central em HVM2, cuja implementação inicial adota uma arquitetura de 32 bits para otimizar operações atômicas e minimizar sobrecarga computacional. Cada conexão na net é representada por um *Port* (32 bits), enquanto nós binários usam *Pair* (64 bits), garantindo acesso rápido às interações, reduzindo a necessidade de metadados explícitos, tornando a manipulação de dados mais eficiente (TAELIN, 2024, p. 16).

A organização da memória é baseada em três buffers: node, que armazena os nós; vars, que gerencia substituições de variáveis; e rbag, que controla os redexes ativos . O espaço de endereçamento disponível permite 2^{29} elementos (cerca de 4 GB para nós e 2 GB para variáveis), o que é adequado para a maioria das aplicações, mas pode ser um limite em execuções extensas, sugerindo uma futura migração para 64 bits (TAELIN, 2024, p. 17).

A implementação em Rust utiliza um bump allocator, um modelo de alocação sequencial sem fragmentação e melhora o desempenho na manipulação dinâmica de nós e

variáveis, tendo um contexto onde todas as alocações serão de, no máximo, 64 bits (TAELIN, 2024, p. 18). Assim, a arquitetura de 32 bits equilibra eficiência e simplicidade, otimizando o uso da memória, mas com espaço para expansão futura.

3.3.5. Paralelismo e Garbage Collection

A escalabilidade de HVM2 depende de sua capacidade de explorar paralelismo massivo, distribuindo a execução de interações entre múltiplos núcleos sem comprometer a integridade do cálculo. Para isso, o sistema adota um modelo de execução baseado na redução concorrente de redexes, permitindo que múltiplos threads processem interações simultaneamente sem necessidade de um controle centralizado. Como a reescrita de HVM2 é fortemente confluente, a ordem em que os redexes são reduzidos não afeta o resultado final, garantindo que a execução paralela ocorra de forma determinística e eficiente (TAELIN, 2024, p. 18-19).

A distribuição do trabalho entre os núcleos é feita por meio de uma estratégia de compartilhamento de redexes, adaptada conforme o ambiente de execução. Em CPUs, HVM2 utiliza task-stealing queues, onde cada thread mantém uma fila local de redexes, retirando novas interações conforme necessário. Se uma thread fica ociosa, ela pode roubar redexes da fila de outra thread, garantindo uma distribuição equilibrada da carga de trabalho (TAELIN, 2024, p. 19). Em GPUs, onde o número de threads é significativamente maior, a abordagem é diferente: a distribuição ocorre dentro dos blocos de execução, utilizando memória compartilhada para coordenar a atribuição de redexes entre os núcleos (TAELIN, 2024, p. 20).

Um dos desafios desse modelo é evitar desalinhamentos na execução paralela, um problema particularmente relevante em operações recursivas complexas. Para mitigar essa questão, HVM2 adota uma política de marcação de redexes críticos, sinalizando quais interações devem ser distribuídas de forma mais estruturada para evitar divergências entre threads (TAELIN, 2024, p. 20). Esse ajuste melhora significativamente o desempenho em algoritmos como ordenação bitônica, onde a estrutura recursiva pode levar a má distribuição de tarefas entre os núcleos da GPU (TAELIN, 2024, p. 21).

Além disso, para maximizar o aproveitamento da memória compartilhada em GPUs, HVM2 implementa um modelo de interações locais, onde os nós recém-criados permanecem no espaço de memória do bloco de execução sempre que possível. Caso um nó precise ser acessado por outro bloco, um mecanismo de vazamento controlado (LEAK interaction) é

acionado, garantindo que as conexões externas sejam resolvidas sem comprometer a performance global do sistema, reduzindo o custo dos acessos à memória global, permitindo que grandes volumes de interações sejam resolvidos diretamente dentro dos blocos de execução. (TAELIN, 2024, p. 21).

Quanto à coleta de lixo (*garbage collection*) na HVM2, a remoção das estruturas não utilizadas é realizada por meio das interações de ERASE e VOID, que, por fazerem parte do modelo de execução, ocorrem juntamente às outras interações paralelamente e em sincronia com toda a *net*, evitando os aspectos negativos das coletas de lixos encontradas em linguagens convencionais que não se baseiam em *Interaction Combinators*.

3.3.6. Limitações e Conclusão

Ao analisar as principais limitações da HVM2, Taelin (p. 22) aponta que uma das principais restrições decorre do uso de um único tipo de nó duplicador (DUP), o que impõe restrições na duplicação de variáveis não lineares, especialmente no caso de funções de ordem superior (TAELIN, 2024, p. 22). Esse problema já foi observado em sistemas similares e poderia ser mitigado por abordagens como múltiplos nós duplicadores ou a introdução de nós auxiliares para gerenciamento de cópias, seguindo técnicas propostas por Lamping (1990) (TAELIN, 2024, p. 23).

Outra limitação relevante está na *ultra-eager evaluation* adotada pelo sistema, que força a execução de todos os redexes disponíveis imediatamente, sem qualquer mecanismo de adiamento ou controle da demanda computacional. Esse comportamento impede a modelagem direta de estruturas infinitas e pode levar à alocação desnecessária de recursos em cálculos que, em um modelo lazy, seriam executados apenas quando necessários (TAELIN, 2024, p. 23). Embora esse problema possa ser contornado por meio da estruturação do código-fonte, ele representa um obstáculo para a compatibilidade direta com linguagens funcionais tradicionais, como Haskell, que dependem fortemente de *lazy evaluation* (TAELIN, 2024, p. 23).

Além das questões semânticas, HVM2 também apresenta desafios de desempenho em execução sequencial, sendo menos eficiente do que compiladores otimizados para código imperativo, como GHC ou GCC. Isso ocorre porque sua estrutura é baseada em reescrita de grafos, que, embora paralelizável, pode ser mais custosa do que abordagens que utilizam loops e mutabilidade explícita (TAELIN, 2024, p. 23). Da mesma forma, a atual arquitetura de 32 bits limita a capacidade do sistema de endereçar grandes volumes de memória,

restringindo o número máximo de nós e variáveis possíveis em um único processo (TAELIN, 2024, p. 24). A adoção de uma arquitetura de 64 bits resolveria essa questão, mas demandaria adaptações para preservar a eficiência em hardware moderno.

Por fim, Taelin (TAELIN, 2024, p. 24-25) conclui que o desenvolvimento de HVM2 comprova que Interaction Combinators, através de sua solidez e de seus aspectos de universalidade e distribuição, viabilizam uma máquina de avaliação massivamente paralela para linguagens de alto nível, atingindo desempenho quase ideal em hardware de múltiplos núcleos. Além disso, apesar das limitações destacadas, é notório um avanço promissor na exploração da reescrita de grafos para computação concorrente. Com aprimoramentos, pode se tornar um alvo eficiente para compiladores de linguagens funcionais, consolidando-se como uma alternativa competitiva para execução paralela escalável.

4. Discussões

A revisão da literatura apresentada neste trabalho permitiu uma compreensão aprofundada dos fundamentos teóricos que sustentam a HVM2, desde as bases conceituais das Proof-Nets e Interaction Nets até sua implementação prática por meio dos Interaction Combinators. A análise dos trabalhos de Girard, Lafont e Taelin evidencia uma evolução consistente no campo da computação paralela, com a HVM2 emergindo como uma proposta inovadora para a execução massivamente paralela de programas de alto nível.

Um dos pontos centrais discutidos foi a relação entre Proof-Nets e Interaction Nets. Enquanto as Proof-Nets foram introduzidas como uma representação gráfica das provas na lógica linear, as Interaction Nets generalizaram esse conceito, introduzindo um modelo computacional baseado em grafos que permite interações locais e determinísticas. Essa generalização foi crucial para a criação dos Interaction Combinators, que simplificaram ainda mais o modelo, mantendo a propriedade de universalidade e a capacidade de execução paralela.

Além disso, a introdução em paralelo à HVM2 da linguagem Bend, criada pelos mesmos desenvolvedores da HVM, reforça a viabilidade dessa abordagem ao fornecer uma linguagem de alto nível que compila diretamente para HVM2, facilitando seu uso em aplicações práticas. No entanto, o artigo carece de uma explicação detalhada sobre como traduzir outras linguagens para HVM2, um aspecto crucial para sua adoção mais ampla. Essa ausência, aliada à falta de uma seção dedicada a benchmarks, dificulta a avaliação quantitativa de seu desempenho em comparação a outras abordagens (TAELIN, 2024, p. 25).

Outro ponto relevante é a evolução da HVM, que já passou por três versões. A HVM1 introduziu a ideia de um runtime baseado em Interaction Nets, mas com limitações na paralelização. A HVM2, abordada neste trabalho, aprimorou essa abordagem, explorando ao máximo o paralelismo em CPUs e GPUs e implementando um modelo baseado em reescrita de grafos *lock-free*. Já a HVM3, em desenvolvimento, busca resolver limitações presentes na versão atual, incluindo melhorias na duplicação de nós, gerenciamento de memória e avaliação mais eficiente.

Entre as principais limitações da HVM2, destacam-se a restrição a um único nó duplicador (DUP), que dificulta a cópia eficiente de variáveis não lineares, e o modelo de avaliação ultra-eager, que pode resultar em alocações desnecessárias de recursos (TAELIN, 2024, p. 23). Além disso, a arquitetura baseada em 32 bits limita a quantidade de memória

endereçável, representando um potencial gargalo para aplicações que demandam grandes volumes de dados. Embora a transição para 64 bits seja uma solução evidente, sua implementação exigiria otimizações para manter a eficiência, especialmente em GPUs, onde o acesso à memória global pode se tornar um fator limitante (TAELIN, 2024, p. 24).

Apesar dessas restrições, a HVM2 representa um avanço significativo na computação paralela, consolidando-se como um modelo eficiente para a execução paralela de linguagens de alto nível. A capacidade de traduzir funções, tipos algébricos e recursão para *Interaction Combinators* abre espaço para o desenvolvimento de compiladores otimizados para hardware paralelo, permitindo que linguagens funcionais e procedurais aproveitem melhor a escalabilidade oferecida pelas arquiteturas modernas. No entanto, a ausência de uma seção dedicada ao sistema de entrada e saída (I/O) no artigo deixa em aberto questões sobre a aplicabilidade da HVM2 em contextos mais gerais, como integração com sistemas externos e manipulação de dados em tempo real.

Assim, embora a HVM2 já demonstra um modelo promissor para computação paralela, a sua evolução contínua para a HVM3, sendo essencial para superar as limitações e consolidar esse modelo computacional como uma alternativa viável para execução paralela de alto desempenho.

5. Conclusão

Este trabalho buscou analisar o contexto teórico relacionado ao modelo universal de computação distribuída dado pelos Interaction Combinators e compreender a implementação desse modelo na HVM2. A revisão da literatura permitiu traçar uma linha histórica que conecta as Proof-Nets de Girard, as Interaction Nets de Lafont e, finalmente, a HVM2 de Taelin, destacando a evolução dos conceitos e sua aplicação prática.

A HVM2 se destaca como uma máquina de avaliação massivamente paralela que aproveita as propriedades de universalidade e distribuição dos Interaction Combinators para alcançar um desempenho próximo ao ideal em hardware moderno. Sua arquitetura, baseada em reescrita de grafos e interações locais, elimina a necessidade de primitivas de concorrência explícitas, oferecendo uma alternativa escalável e eficiente para a execução de programas de alto nível.

Em conclusão, a HVM2, apesar de enfrentar desafios em sua implementação, representa um avanço significativo no campo da computação paralela, oferecendo um modelo que pode ser aplicado a uma variedade de linguagens de programação. Com aprimoramentos futuros, como a introdução de múltiplos nós duplicadores e a adoção de uma arquitetura de 64 bits, a HVM, a partir do seu desenvolvimento contínuo, tem o potencial de se consolidar como uma plataforma eficiente e escalável para a execução de programas massivamente paralelos, contribuindo para o avanço da computação distribuída e paralela.

Referências

BAUMEISTER, R. F.; LEARY, M. R. Writing narrative literature reviews. **Review of General Psychology**, v. 1, n. 3, p. 311-320, 1997.

COPELAND, B. Jack. **The Church-Turing Thesis**. In: ZALTA, Edward N.; NODELMAN, Uri (ed.). *The Stanford Encyclopedia of Philosophy*. Winter 2024 edition. Stanford: Metaphysics Research Lab, Stanford University, 2024. Disponível em: https://plato.stanford.edu/archives/win2024/entries/church-turing/. Acesso em: 22 out. de 2024.

FERNÁNDEZ, M.; MACKIE, I. A calculus for interaction nets. **Principles and Practice of Declarative Programming**. Berlin: Springer, 1999. p. 170-187.

GIRARD, Jean-Yves. Linear logic. **Theoretical Computer Science**, v. 50, p. 1-102, 1987. Disponível em: https://doi.org/10.1016/0304-3975(87)90045-4.

GRANT, M. J.; BOOTH, A. A typology of reviews: An analysis of 14 review types and associated methodologies. **Health Information & Libraries Journal**, v. 26, n. 2, p. 91-108, 2009.

LAFONT, Yves. Interaction nets. Em: **Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages**, dez. 1990. p. 95-108.

LAFONT, Yves. **Introduction to interaction nets**. Marseille: CNRS - Institut de Mathématiques de Luminy, 1998. Disponível em: https://www.i2m.univ-amu.fr/perso/yves.lafont/papers.html. Acesso em: 13 jan. 2025.

LAFONT, Yves. From proof nets to interaction nets. **London Mathematical Society Lecture Note Series**, p. 225-248, 1995.

LAFONT, Yves. Interaction combinators. **Information and Computation**, v. 137, n. 1, p. 69-101, 1997.

ROWLEY, J.; SLACK, F. Conducting a literature review. **Management Research News**, v. 27, n. 6, p. 31-39, 2004.

TAELIN, Victor. **HVM2: A parallel evaluator for interaction combinators**. 2024. Disponível em: https://github.com/HigherOrderCO/HVM/blob/main/paper/HVM2.pdf. Acesso em: 22 out. 2024.