UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

# Interoperabilidade Lógica via Mapeamentos entre Instituições

por

JULIANA KAIZER VIZZOTTO

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

Prof. Dr. Alfio Martini
Orientador

Prof. Dr. Antônio C. Rocha Costa
Co-orientador

Porto Alegre, agosto de 2001

**CIP — CATALOGAÇÃO NA PUBLICAÇÃO**

Vizzotto, Juliana Kaizer

Interoperabilidade Lógica via Mapeamentos entre Instituições / por Juliana Kaizer Vizzotto. — Porto Alegre: PPGC da UFRGS, 2001.

98 f.: il.

Dissertação (mestrado) — Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2001. Orientador: Martini, Alfio; Co-orientador: Costa, Antônio C. Rocha.

1. Interoperabilidade Lógica. 2. Mapeamentos entre Instituições. 3. Especificação Formal. I. Martini, Alfio. II. Costa, Antônio C. Rocha. III. Título.

*Este trabalho é dedicado ao Pai Angelo,*
*à Mãe Terezinha e à Irmã Dani,*
*pelo Amor de todos*
*os momentos.*

# Agradecimentos

Agradeço a todos que de alguma forma ajudaram na realização deste trabalho, seja direta ou indiretamente. Em especial:

- Ao meu orientador, o prof. Alfio Martini, por toda dedicação, disponibilidade e principalmente por ter acreditado em mim. Valeu Alfio!
- Ao meu co-orientador, o prof. Antônio Carlos da Rocha Costa por me dar esta oportunidade.
- À Graçaliz Dimuro, Renata Reiser, Marilton Aguiar e a todo pessoal de Pelotas por me acompanharem durante toda a caminhada, desde a graduação.
- Ao pessoal do Instituto de Informática da UFRGS, funcionários e professores, por me acolherem.
- Aos professores: Paulo Blauth e Tiarajú.
- Aos alunos: Mônica, Rafael, Júlio e Pilla pela alegria e por todas ajudas do dia a dia!
- As amigas: Fer e Flávia pela Amizade de todas as horas.
- A minha família pelo Amor, carinho e incentivo sempre presentes.
- Ao CNPq por financiar este trabalho.

Por último, agradeço a pessoa que mais me acompanhou ao longo desses meses, ao meu namorado André Du Bois. *Obrigado por todo apoio e por ser tão especial*!

# Sumário

# Lista de Abreviaturas

| | |
|---|---|
| **FOL** | Lógica de Primeira Ordem Multi-Sortida |
| **Horn** | Lógica Horn sem Sorts |
| **MSEqtl** | Lógica Equacional Multi Sortida |
| **OSEqtl** | Lógica Equacional Ordenada por Sorts |

# Lista de Figuras

# Resumo

A integração estruturada e consistente de diversas especificações (ou visões ) de um sistema é hoje uma questão essencial na moderna abordagem para especificação e desenvolvimento de *software*. Neste contexto, precisamos de uma teoria que fale sobre formalismos de especificação e que ao mesmo tempo nos ofereça conceitos e construções para estabelecer-mos relações entre eles.

Com este trabalho temos o objetivo de discutir noções rigorosas para idéia de lógica, técnicas fundamentais para relacioná-las e mostrar a utilização destes conceitos para abordar a questão da interoperabilidade formal, especialmente de provas. Como formalização para idéia de lógica utilizaremos as *Instituições* de Goguen & Burstall e a extensão de Meseguer para *General Logics*. Como técnica para relacionar lógicas trabalharemos com os mapeamentos *Plain* e *Simples* de Meseguer. Atenção especial é dada à discussão das propriedades destes mapeamentos com vista à reutilização de componentes lógicos, especialmente da relação de consequência entre fórmulas.

Contribuições desta dissertação incluem um tratamento acessível para os conceitos fundamentais necessários para estudar lógicas e sua integração, uma exposição uniforme e detalhada de uma família de sistemas lógicos e uma apresentação categórica desta integração via mapeamentos.

**Palavras-chave:** Interoperabilidade Lógica, Mapeamentos entre Instituições, Especificação Formal.

**TITLE:** "LOGICAL INTEROPERABILITY VIA MAPS"

# Abstract

The integration in a sound and structured way of several specifications (or views) of a system is a key research area in (modern) software specification and development. In this context, we need a theory to speak not only about specification logics, but also one that can offer us concepts and constructions to establish relations between these formalisms.

In this work we have the purpose of discussing rigorous notions for the idea of logic, presenting fundamental techniques to relate them, and introducing important constructions to approach the problem of formal interoperability, most notably of proofs. As a formalization of the informal idea of a logical system, we use Goguen & Burstall's concept of *Institutions* and Meseguer's further extension to *General Logics*. Meseguer's *Plain* and *Simple Maps of Institutions* are the tools we use to relate and map logics. Besides, special attention is given for the discussion of the essential properties of these maps concerning the borrowing of logical components, specially of consequence relations.

Contributions of this thesis include a smooth presentation of fundamental techniques to study and relate logics, an uniform and detailed exposition of a number of logical systems, and a categorical formalization of relations between these formalisms by way of maps between institutions.

**Keywords:** Maps between Institutions, Logical Interoperability, Formal Specification.

# 1 Introdução

## 1.1 Motivação

A complexidade dos sistemas de computação tem crescido consideravelmente nos últimos anos. Os requisitos aumentam em escala e funcionalidade. Desta maneira, o desenvolvimento de sistemas requer, cada vez mais, o suporte de metodologias rigorosas.

Neste sentido, salientamos a utilização de métodos formais, que são linguagens, técnicas e ferramentas para especificar e verificar sistemas, baseadas em fundamentos matemáticos [CLA 96]. A utilização de métodos formais visa principalmente aumentar o entendimento dos sistemas, revelando inconsistências, ambigüidades e falhas que podem, caso contrário, não serem detectadas. Além disso, especificações formais facilitam a modularização e o reuso na produção de *software*. Também propiciam rápida prototipação e metodologias para verificação formal.

Devido a necessidade de capturar vários aspectos dos sistemas (i.e., aspectos algébricos e operacionais, tratamento de dados, segurança e concorrência) surge uma diversidade de formalismos lógicos para especificação formal. Alguns exemplos clássicos de lógicas para especificação incluem: lógica de predicados e suas variantes incluindo lógica equacional multi-sortida e ordenada por sorts [GOG 92], lógica de reescrita [MES 90], CCS [MIL 80] e lógicas temporais [ARR 96].

No entanto, a existência desses diversos formalismos, embora um sinal de vitalidade e criatividade intelectual [MES 89], nos impõe ao mesmo tempo algumas dificuldades. Primeiramente, não temos como comparar resultados e ferramentas desenvolvidas em diferentes formalismos. Também, as linguagens de especificação são, na maioria das vezes, utilizadas independentemente, sem a possibilidade de "integração" na solução de um mesmo problema.

Neste contexto, precisamos de uma teoria que fale sobre formalismos para especificação e que ao mesmo tempo nos ofereça conceitos e construções para estabelecermos relações entre eles. Mais precisamente, necessitamos de uma "estrutura" que capture as propriedades básicas de qualquer lógica para especificação.

Este dissertação é baseada no trabalho de Goguen & Burstall [GOG 92] sobre *Instituições*[1] e na extensão desta teoria por Meseguer no clássico *General Logics* [MES 89].

As *Instituições* foram introduzidas para axiomatizar a noção *informal* de lógica, consistindo nos ingredientes mínimos que qualquer formalismo deve ter, i.e., a idéia de *sintaxe* (linguagem), *semântica* (modelos/álgebras) e de *satisfação*. Formalmente, este conceito consiste de uma categoria de *assinaturas*, de dois funtores, um determinando o conjunto de *sentenças* e o outro a categoria de *modelos* para cada assinatura, de uma relação de satisfação entre modelos e sentenças respeitando os morfismos entre assinaturas. Esta é a visão da lógica como uma linguagem para falar sobre modelos.

Em [MES 89], é introduzida uma abordagem unificadora em que um sistema lógico integra aspectos dedutivos (dedução & provas) e semânticos (modelos), i.e., é formalizado como uma *lógica* junto com um *cálculo de provas*, sendo que uma lógica é definida como uma instituição juntamente com um sistema de conseqüência.

[1] Do inglês institution

Esta noção é necessária para abordar a questão da interoperabilidade de relação de consequência, i.e., de provas, como será mostrado adiante. Entretanto, os relacionamentos entre sistemas lógicos são capturados essencialmente no aspecto semântico de uma lógica, mais precisamente a nível de instituição.

Na literatura existem outras noções interessantes para a idéia de sistema lógico, entre elas podemos citar o conceito de Instituição-$\pi$ [FIA 87], em que a formalização de lógica é definida em função da relação de conseqüência. Desta forma, baseada em uma abordagem dedutiva. Em [MES 89], essa noção é apresentada com o nome de *sistema de consequência*.

Desta maneira, podemos construir especificações com sistemas lógicos arbitrários. Contudo, trabalhar com uma única lógica, não é suficiente em muitas situações. Como já mencionado, normalmente precisamos de vários sistemas lógicos na mesma tarefa de especificação e desenvolvimento (i.e., para capturar uma diversidade de aspectos dos sistemas).

Essa necessidade de trabalhar com diversos formalismos lógicos na mesma tarefa introduz a questão da interoperabilidade lógica, ou seja, como conectar de uma maneira matemática rigorosa as diferentes lógicas para especificação. Neste contexto, esta questão pode ser vista sob diversos aspectos, entre eles,

- a necessidade de representar (ou "codificar") uma lógica $\mathcal{L}$ em outra, digamos $\mathcal{U}$, normalmente entendida como uma lógica universal, no sentido de que várias lógicas podem ser representadas nela de uma maneira natural (ver Figura 1.1);

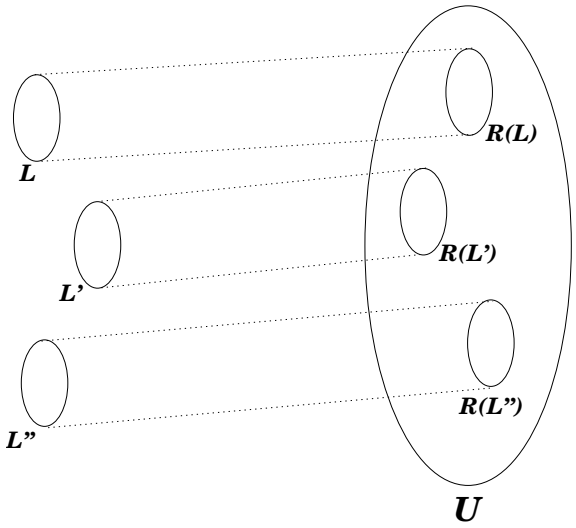



FIGURA 1.1 – Lógica Universal

- a capacidade de utilizar de forma integrada, com um tratamento matemático rigoroso, as diferentes formalizações de um sistema e as diversas estruturas (*softwares*, provas, relações de satisfação e conseqüência) que suportam tais formalizações.

Neste sentido, existem várias abordagens para tratar essa questão. Uma maneira é a construção de novos métodos (sistemas lógicos, ferramentas, linguagens) que reunam como um todo, diferentes características de diferentes lógicas. Exemplos de linguagens que foram construídas englobando características de linguagens já

existentes são: Lotos (CCS + especificação algébrica) [EIJ 89], RAISE [MIL 90], Z + CSP [FIS 97]. Entretanto, estamos certos que, devido ao alto custo e o tempo gasto no desenvolvimento de sistemas e na construção de novos métodos, a reutilização é uma questão essencial. Então torna-se clara a necessidade de uma metodologia que possibilite conectar (reutilizar) as lógicas já existentes, capacitando assim, a utilização de uma diversidade de sistemas lógicos (independentes) na solução de um mesmo problema.

Nesta dissertação, discutimos conceitos que vão de encontro com esta abordagem (reutilização), mais especificamente apresentamos a conexão entre as lógicas e seus componentes capturada através de mapeamentos entre instituições.

Com base nessas idéias (mapeamentos) existe uma série de trabalhos que vem desenvolvendo metodologias para tratar de diversas aplicações. Cerioli & Meseguer tem o objetivo de utilizar (reutilizar) componentes (modelos, cálculos, sistemas de provas, relações) de uma lógica em outra. Diaconescu [DIA 98, DIA 96] busca a conexão entre lógicas através de mapeamentos, para dar semântica para a linguagem CafeOBJ. Em [TAR 96], Tarlecki propõe a utilização de diferentes formalismos em diferentes visões do mesmo problema. Também, Martini, Wolter e Haeusler [MAR 2001], discutem a situação de conectar módulos de especificações (com semânticas possivelmente heterogêneas).

Com isso, podemos dizer que o conteúdo presente aqui está inserido na área de estudo de metodologias matemáticas para o *relacionamento*, *integração* e *reutilização* de lógicas.

## 1.2 Objetivos e Contribuições

Os objetivos deste trabalho incluem

- discutir noções rigorosas da idéia de *lógica* e de *lógica para especificação*, bem como compará-las. Especificamente concentrando esta dissertação nas instituições de Goguen & Burtall, como formalização modelo-teórica para idéia de lógica para especificação e no clássico *General Logics* de Meseguer, tratando o conceito de sistema lógico como uma abordagem unificadora;

- introduzir técnicas fundamentais para relacionar lógicas, apresentando esta idéia através de *mapeamentos entre instituições*;

- abordar esses conceitos para tratar a questão da interoperabilidade formal, especialmente de provas.

Por exemplo, considere a Figura 1.2, onde temos a conexão de dois sistemas lógicos (Instituição + Sistema de Conseqüência + Cálculo) no nível de instituição. O desenho representa uma situação em que a relação de conseqüência é *transportada* através de mapeamentos entre instituições. Mais precisamente, temos um reuso de provas acontecendo através de um mapeamento junto com as relações internas das lógicas (ver setas pontilhadas). Note que o reuso discutido aqui é a nível de deduções ($\Gamma \vdash \varphi$, i.e., a fórmula $\varphi$ é derivável de um conjunto de fórmulas $\Gamma$) não considerando o "escore" de prova (aplicações das regras do cálculo para construir a prova). Mais especificamente, considere por exemplo que para a *lógica de primeira ordem* temos diversos cálculos de provas, i.e., Hilbert, cálculo de seqüentes, dedução natural, cada

um deles com uma noção diferente de prova. Contudo, a relação de provabilidade entre as fórmulas continua a mesma.

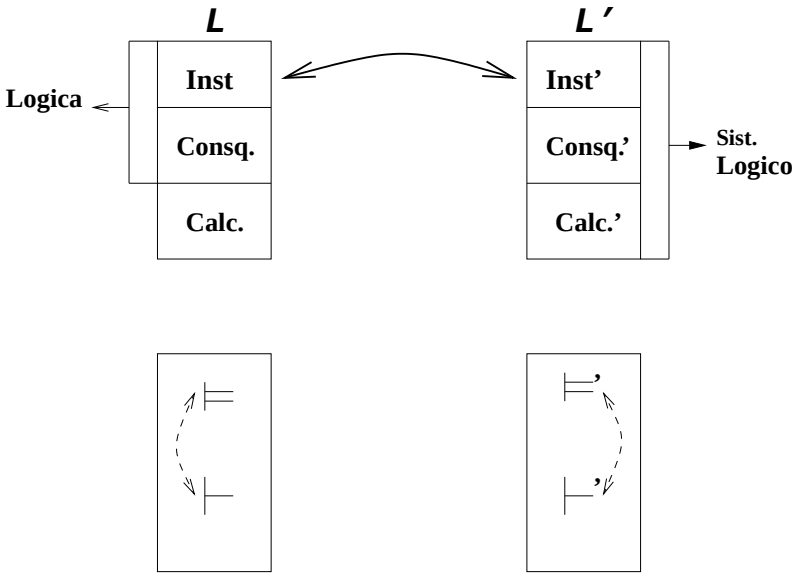



FIGURA 1.2 – Reuso

Um caminho normalmente percorrido para a interoperabilidade de provas pode ser verificado no seguinte diagrama.

$$\begin{array}{ccc} \Gamma \models \varphi & \xleftrightarrow{\quad 2 \quad} & \alpha(\Gamma) \models' \alpha(\varphi) \\ \updownarrow 1 & & \updownarrow 3 \\ \Gamma \vdash \varphi & & \alpha(\Gamma) \vdash' \alpha(\varphi) \end{array}$$

Onde a seta 2, que representa o mapeamento entre instituições, *induz* um mapeamento entre a relação de conseqüência. Considerando que as provas nas duas lógicas estão nos nodos abaixo ($\vdash$), então um mapeamento entre instituições junto com as relações de completude e coerência, representadas pelas equivalências 1 e 3, possibilita o reuso.

É neste sentido que o título desse trabalho "Interoperabilidade Lógica via Mapeamentos entre Instituições" aborda a questão da integração de formalismos lógicos.

Assim, contribuições desta dissertação incluem um tratamento acessível para conceitos fundamentais necessários para estudar lógicas e sua integração, uma exposição detalhada de uma série de sistemas lógicos e uma apresentação categórica desta integração via mapeamentos.

Além disso, é interessante mencionar que este trabalho está inserido no contexto do projeto CNPq-NSF MEFIA: *Mathematical and Engineering Foundations for Interoperability via Architecture*.

Na seqüência colocamos uma breve discussão do material encontrado em cada capítulo da dissertação.

# Capítulo 2

Discutimos sobre as noções de lógica tradicional e lógica para especificação. Apresentamos as duas abordagens de estudo da lógica, ou seja, modelo-teórica ($\models$), que estuda a relação de satisfação de fórmulas por modelos e a prova-teórica ($\vdash$) que enfatiza a relação de conseqüência entre um conjunto de fórmulas e uma fórmula derivável deste conjunto. Assim, o objetivo deste capítulo é comparar essas duas noções (tradicional & especificação), analisando seus aspectos principais. Além disso, no final desta parte do texto apresentamos uma tabela comparativa (resumida) na qual pode ser verificada a principal diferença quando trabalhamos com especificações, i.e., a presença de assinaturas nas definições de lógica para especificação. O material desta parte do texto é baseado principalmente no clássico *General Logics* [MES 89] e também em outros trabalhos importantes como [GOG 92] e [COU 97].

# Capítulo 3

Apresentamos as *instituições* como uma formalização modelo teórica para a idéia de lógica para especificação. Primeiramente, temos uma discussão informal sobre os ingredientes básicos que são capturados por essa idéia. Neste sentido, salientamos que o conceito de instituições nos permite construir estruturas com as quais podemos escrever especificações. A seguir, introduzimos a definição formal consistindo de uma categoria de assinaturas ($\mathsf{Sign}$), de um funtor ($Sen : \mathsf{Sign} \to \mathsf{Set}$), de um funtor ($Mod : \mathsf{Sign} \to \mathsf{Set}$) e de uma relação de satisfação. O funtor *Sen* e o funtor *Mod* determinam respectivamente, o conjunto de sentenças e a categoria de modelos para cada assinatura. O conceito de satisfação entre modelos e sentenças, indexado por assinaturas, determina a invariância da troca de notação. Um observação importante que deve ser notada é que o funtor *Mod* é contravariante, já que uma determinada relação entre dois objetos sintáticos determina essencialmente uma tradução semântica no sentido oposto. Também, citamos diversos exemplos de instituições. Além disso, esse capítulo possui algumas definições derivadas que serão utilizadas no andamento do texto. Essas estruturas são a noção de conseqüência semântica e alguns conceitos que envolvem teorias e seus modelos. Esta apresentação é baseada em textos clássicos como [GOG 92, GOG 84].

# Capítulo 4

Apresentamos detalhadamente duas instituições que são mais relevantes para este texto, i.e., *lógica equacional multi sortida* e *lógica equacional ordenada por sorts*. São discutidos, particularmente para a primeira lógica, os conceitos de assinaturas e morfismos entre assinaturas, bem como a idéia de álgebra e homomorfismos entre álgebras. Seguindo a definição de instituição introduzimos em detalhes o funtor *Sen*, o funtor *Mod* e a relação de satisfação indexada por assinaturas. Em relação ao funtor *Sen* são vistos os conceitos de termos e fórmulas nessas lógicas, bem como a tradução destes induzida pelo morfismo entre assinaturas. Na seção respectiva ao funtor *Mod* cuidado especial é dado ao *funtor esquecimento*, que pode ser visto como uma seta entre as categorias de modelos de assinaturas que estão relacionadas por morfismos em $\mathsf{Sign}$. Um conceito interessante que pode ser analisado nessa instituição, são as atribuições correspondentes, já que elas têm papel essencial

para provar a relação de satisfação. Para a *lógica equacional ordenada por sorts* apresentamos somente a categoria de assinaturas devido a similaridade dos demais conceitos com a instituição discutida anteriormente. Esses exemplos são adaptados principalmente de [MAR 99, WOL 94].

# Capítulo 5

Temos uma apresentação detalhada da *lógica Horn sem sorts* (*Prolog* puro - conjunção e implicação) vista como uma instituição. Também, discutimos minuciosamente a categoria Sign, o funtor *Sen*, o funtor *Mod* e a relação de satisfação. Na instituição *Horn* um ponto essencial a ser observado é a idéia que uma "lógica sem sorts" pode ser entendida como uma lógica com assinaturas que possuem um único sort. Desta forma este único sort pode ser pensado como uma representação do *universo* (domínio). Além disso, neste capítulo, os predicados são tratados com mais atenção, já que os outros conceitos são similares ao capítulo anterior. Esta explicação busca ser tão detalhada quanto possível e também é baseada em [MAR 99, WOL 94].

# Capítulo 6

Analisamos dois tipos de mapeamentos entre instituições, mais precisamente, *mapeamento Puro*[2] e *mapeamento Simples*[3]. Um mapeamento entre duas lógicas é essencialmente uma estrutura que traduz assinaturas e sentenças de $\mathcal{L}$ para $\mathcal{L}'$ e que mapeia modelos de $\mathcal{L}'$ para $\mathcal{L}$. Como mapeamento plain temos uma seta que relaciona (*representa*) uma instituição $\mathcal{L}$ em outra $\mathcal{L}'$, tal que as assinaturas de $\mathcal{L}$ são mapeadas para assinaturas de $\mathcal{L}'$. Já um mapeamento simples traduz assinaturas de $\mathcal{L}$ para *teorias* de $\mathcal{L}'$. Isto acontece pois a tradução da informação na assinatura da lógica origem pode requerer axiomas na lógica destino. Apresentamos detalhadamente um mapeamento puro entre a *MSEqtl* e a *OSEqtl* e um mapeamento simples entre a *MSEqtl* e a lógica *Horn*. Atenção especial é dada para as propriedades lógicas desses mapeamentos. Esses dois tipos de setas foram introduzidas em [MES 89]. Este capítulo é essencialmente influenciado por [MAR 99].

# Capítulo 7

Apresentamos um estudo de caso: *reutilizando um provador de teoremas através de mapeamentos entre instituições*. Esta parte da dissertação tem como objetivo utilizar mapeamentos entre instituições como abordagem para tratar um simples aspecto da *interoperabilidade lógica*, ou seja, uma situação problema em que nos deparamos com a necessidade de *reutilizar* o provador de teoremas OBJ [GOG 92a]. Primeiramente introduzimos o *problema*, i.e., ao trabalharmos com especificações em *MSEqtl* surgiu a necessidade de executarmos provas sob essas especificações. Seguindo, discutimos a idéia de utilizar o provador de teoremas OBJ para executar tais provas, já que conhecemos esse sistema e o temos instalado em nossas máquinas. Finalmente, mostramos um mapeamento puro entre as duas lógicas em questão como uma estrutura matemática adequada para modelar a situação. A execução das provas na ferramenta *OSEqtl* é possível no sentido de que todos e, somente todos, os

[2]Do inglês *Plain*
[3]Do inglês *Simple*

axiomas prováveis nas especificações *MSEqtl* são provados em *OSEqtl*. Temos essa garantia, pois esse mapeamento (*MSEqtl* $\rightarrow$ *OSEqtl*) possui a propriedade lógica de ser uma extensão conservativa. Também é importante salientar que para o reuso acontecer é necessário, além do mapeamento entre as instituições, considerarmos as relações internas das lógicas (completude e coerência, ver Figura 1.2). Além disso, buscamos salientar que mesmo situações simples requerem um tratamento matemático cuidadoso (i.e., um mapeamento adequado e que tem a propriedade de ser *conservativo* junto com relações internas das lógicas).

# Capítulo 8

Finalmente, discutimos as conclusões e direções de trabalhos futuros.

# Notas para o Leitor

Na notação de teoria das categorias utilizada neste texto consideramos que $|\ \mathsf{C}\ |$ descreve uma classe de objetos de uma categoria $\mathsf{C}$, agora, se o objeto $A$ está na categoria $\mathsf{C}$ temos $A \in |\ \mathsf{C}\ |$. Se escrevemos $f \in \mathsf{C}$ e $f$ é um morfismo, significa que o morfismo $f$ está na categoria $\mathsf{C}$.

Se $A$ e $B$ são objetos em uma categoria $\mathsf{C}$, representamos um mapeamento $m$ entre esses dois objetos como $A \mapsto B$.

Uma função $f$ com domínio em $A$ e codomínio em $B$ é escrita $f : A \rightarrow B$.

Funtores são setas entre categorias. Sejam $\mathsf{C}$, $\mathsf{D}$, $\mathsf{E}$ categorias e $F : \mathsf{C} \rightarrow \mathsf{D}$, $G : \mathsf{D} \rightarrow \mathsf{E}$ dois funtores. Representamos a composição desses funtores por $F;\ G : \mathsf{C} \rightarrow \mathsf{E}$.

Uma transformação natural é uma seta entre dois funtores. Sejam $\mathsf{C}$ e $\mathsf{D}$ duas categorias e $F : \mathsf{C} \rightarrow \mathsf{D}$, $G : \mathsf{C} \rightarrow \mathsf{D}$ dois funtores. Então, uma transformação natural $\eta$ entre esses dois funtores é escrita $\eta : F \Rightarrow G : \mathsf{C} \rightarrow \mathsf{D}$.

Em teoria das categorias, existem algumas categorias especiais. São categorias bem conhecidas que tem nomes específicos. Nesta dissertação, nos referimos a algumas dessas categorias, e.g., $\mathsf{Cat}$ e $\mathsf{Set}$. Em $\mathsf{Cat}$, os objetos são categorias e as setas são funtores. Em $\mathsf{Set}$, os objetos são conjuntos e os morfismos são funções totais.

Para as definições e proposições apresentadas neste trabalho utilizamos o seguinte critério: quando o conceito requer uma prova mais cuidadosa então é uma proposição seguida de prova, caso contrário é uma definição. Se a definição requer alguma verificação, então essa é inserida em uma nota.

# 2 Lógica Tradicional & Lógica para Especificação

Há muito tempo a ciência da computação tem encontrado na lógica verdadeiras soluções para os seus problemas. Além de programação em lógica, provadores de teoremas, especificação e verificação de programas outras áreas têm uma interessante interação com lógica, incluindo teoria dos tipos, concorrência, inteligência artificial, banco de dados, semântica operacional e compiladores [MES 89].

De qualquer forma, quando falamos sobre lógica, podemos pensar em duas abordagens: a modelo-teórica ($M \models \varphi$), preocupada com a relação de satisfação entre um modelo $M$ e uma fórmula $\varphi$, e a prova-teórica ($\Gamma \vdash \varphi$), que axiomatiza a relação de conseqüência entre um conjunto de sentenças $\Gamma$ e uma sentença $\varphi$ derivável de $\Gamma$. Também é bom mencionar que a lógica como um instrumento matemático pode ser compreendida sob no mínimo dois aspectos. Um deles é a visão tradicional da lógica como uma linguagem de expressões para falar sobre estruturas, sendo essas o universo que dá significado aos objetos dessa linguagem. O outro é uma extensão natural deste primeiro que leva em conta a noção de assinaturas e a relação entre assinaturas. Isto é essencial sob o ponto de vista do uso da lógica como linguagem para descrição e especificação de sistemas de forma modular e estruturada.

O objetivo deste capítulo é apresentar uma descrição sistemática das estruturas que compõem lógicas. Na primeira seção apresentamos isto sob uma visão tradicional. Na seção seguinte adaptamos o material desenvolvido anteriormente, levando em conta assinaturas e seus relacionamentos.

## 2.1 Lógica - Uma Visão Tradicional

Normalmente, quando falamos em lógica temos em mente *sintaxe* e *semântica*. Neste contexto, podemos estudá-la essencialmente em seus aspectos semânticos, ou seja, uma abordagem orientada à teoria dos modelos (sistemas de satisfação). Por outro lado, também podemos estudar lógica focalizando seus aspectos dedutivos, enfatizando a relação de conseqüência entre conjuntos de fórmulas e fórmulas. No entanto, segundo Meseguer [MES 89], em muitos casos nenhuma dessas abordagens é suficiente para fornecer uma axiomatização das linguagens lógicas. Assim, vários autores integram ambos aspectos semânticos e dedutivos.

É interessante mencionar também, que existem lógicas formais somente com teoria de prova, sem semântica definida, e.g., LF (*Edinburgh Logical Framework*) [AVR 89], Isabelle [PAU 2001]. Essas lógicas são geralmente utilizadas na área de *teoria de tipos de alta ordem*.

Além disso, com a evolução dos sistemas de computação e principalmente com as direções de pesquisa em inteligência artificial teve o surgimento de lógicas modernas, que ainda não tem semântica, i.e., lógicas não-monotônicas, paraconsistentes.

Nesta seção apresentamos a lógica vista sob seu aspecto sintático (sistema de conseqüência) e seu aspecto semântico (sistema de satisfação). Também abordamos uma noção que integra essas duas idéias.

### 2.1.1 Sistema de Conseqüência

Informalmente, um sistema de conseqüência é composto por um conjunto de fórmulas e uma relação de conseqüência , entre conjuntos de fórmulas e fórmulas.

Por exemplo, na *lógica equacional multi sortida* (daqui para frente simplesmente *MSEqtl*, veja seção 3.1 e 4.1), usualmente temos uma assinatura consistindo basicamente de símbolos de sorts e de símbolos de funções. Com base nesta assinatura temos a noção de *fórmulas* da lógica *MSEqtl*.

Sabendo que um conjunto $\Gamma$ é um conjunto de tais fórmulas *MSEqtl*, então as regras de um cálculo de provas desta lógica especificam a seguinte relação de conseqüência ,

$$\Gamma \vdash_{MSEqtl} \varphi$$

Esta relação na *MSEqtl* ($\vdash_{MSEqtl}$) satisfaz três propriedades básicas, que podem ser generalizadas como as propriedades que toda relação de conseqüência deve satisfazer [MES 89]. São elas:

- *reflexividade*, i.e., se assumirmos uma sentença sempre podemos prová-la;
- *monotonicidade*, i.e., podemos provar com mais hipóteses, o que já provamos com menos;
- *transitividade*, i.e., se utilizarmos como hipótese adicional alguma coisa já provada, não deve ser possível chegarmos à mais conclusões que aquelas que poderíamos chegar com a hipótese original.

Essas idéias são capturadas na seguinte

**Definição 2.1 (Sistema de Conseqüência )** Um sistema de conseqüência é uma dupla $\mathcal{SC} = \langle Form, \vdash \rangle$ tal que:

- $Form$ é um conjunto de fórmulas;
- $\vdash \subseteq \wp(Form) \times Form$ é a relação de conseqüência tal que as seguintes propriedades são satisfeitas:
  - *reflexividade*: para qualquer $\varphi \in Form, \{\varphi\} \vdash \varphi$;
  - *monotonicidade*: se $\Gamma \vdash \varphi$ e $\Gamma \subseteq \Gamma'$ então $\Gamma' \vdash \varphi$;
  - *transitividade*: se $\Gamma \vdash \varphi_i$, para $i \in I$, e $\Gamma \cup \{\varphi_i \mid i \in I\} \vdash \psi$, então $\Gamma \vdash \psi$.

### 2.1.2 Sistema de Satisfação

Informalmente, um sistema de satisfação é composto por um conjunto de fórmulas, uma classe de modelos e uma relação de satisfação de fórmulas por modelos.

Voltando ao exemplo da *MSEqtl*, dado um conjunto de fórmulas, podemos associar uma classe de modelos $\mathcal{M}_{MSEqtl}$ a esse conjunto. Onde cada modelo $M$ consiste de um conjunto para cada símbolo de sort juntamente com uma interpretação para cada símbolo de função.

Desta maneira, dado um modelo $M \in \mathcal{M}_{MSEqtl}$ e uma fórmula $\varphi \in Form_{MSEqtl}$, temos a noção de *satisfação* de uma fórmula $\varphi$ por um modelo $M$, escrita $M \models \varphi$.

**Definição 2.2 (Sistema de Satisfação)** Um sistema de satisfação é uma tripla $\mathcal{SS} = \langle Form, \mathcal{M}, \models \rangle$ onde:

- $Form$ é um conjunto de fórmulas;
- $\mathcal{M}$ é uma classe de modelos;
- $\models \subseteq \mathcal{M} \times Form$ é a relação de satisfação.

Um sistema de satisfação induz uma relação de conseqüência entre conjunto de fórmulas e fórmulas, que é formalizada na seguinte

**Definição 2.3 (Conseqüência Semântica)** Sejam $\mathcal{SS} = \langle Form, \mathcal{M}, \models \rangle$ um sistema de satisfação, $\Gamma \subseteq Form$ um conjunto de fórmulas e $\varphi \in Form$ uma fórmula. Temos que $\varphi$ é conseqüênciasemântica de $\Gamma$, escrita $\Gamma \models \varphi$, se e somente se, para todo $M \in \mathcal{M}$, o seguinte vale

$$M \models \Gamma \Rightarrow M \models \varphi.$$

### 2.1.3 Lógica = Sistema de Satisfação + Sistema de Conseqüência

Em muitos casos é insuficiente entender uma lógica somente como um sistema de satisfação ou como um sistema de consequência.

Geralmente, uma lógica tem essencialmente dois componentes: um sistema de conseqüência e um sistema de satisfação.

**Definição 2.4 (Lógica)** Uma lógica é uma quádrupla $\mathcal{L} = \langle Form, \mathcal{M}, \vdash, \models \rangle$ tal que:

- $\langle Form, \vdash \rangle$ é um sistema de conseqüência ;
- $\langle Form, \mathcal{M}, \models \rangle$ é um sistema de satisfação.

Uma lógica é consistente se $\Gamma \vdash \varphi \Rightarrow \Gamma \models \varphi$. Uma lógica é completa se $\Gamma \models \varphi \Rightarrow \Gamma \vdash \varphi$.

## 2.2 Lógica - Uma Visão para Especificação

A idéia discutida nesta seção é que podemos desejar trocar de assinatura $\Sigma$ (no caso da lógica *MSEqtl*, uma assinatura é um par $\langle S, F \rangle$, onde $S$ é um conjunto de sorts e $F$ é uma família de símbolos de funções, ver definição 4.1) e mover para uma nova assinatura $\Sigma'$. Em alguns casos a assinatura $\Sigma'$ pode ser uma extensão da sintaxe original. Mas, por outro lado, $\Sigma'$ pode ser uma assinatura diferente, contendo novos símbolos, sendo que os símbolos velhos foram traduzidos para esses novos. Neste sentido, a idéia geral é podermos ter um morfismo, $\phi : \Sigma \to \Sigma'$, entre assinaturas. Na *MSEqtl*, um morfismo entre assinaturas consiste de um par de funções, uma para símbolos de sorts e outra, que preserva a aridade e o mapeamento de sorts, para símbolos de operações. Essas funções são utilizadas para mapear os símbolos velhos para símbolos novos. Essas assinaturas formam uma categoria que chamamos de Sign que tem assinaturas como objetos e morfismos entre assinaturas como setas. Mais ainda, temos que morfismos entre assinaturas induzem a uma

tradução a nível de sentenças. Assim, podemos associar a cada sentença $\varphi$ relativa a $\Sigma$ uma sentença correspondente relativa a $\Sigma'$, pela substituição dos velhos símbolos por novos de acordo com o morfismo entre assinaturas. Isso pode ser expressado de maneira estrutural e compacta dizendo que o processo que associa a cada assinatura $\Sigma$ seu conjunto de sentenças $Sen(\Sigma)$ é um funtor $Sen : \mathsf{Sign} \to \mathsf{Set}$, com origem na categoria de assinaturas e destino na categoria de conjuntos. Contudo, é essencial que a relação de conseqüência seja *estável sob a troca de notação* pelo morfismo entre assinaturas, ou seja: se podemos provar uma conclusão de alguns axiomas (sentenças na lógica), então, para qualquer tradução $\phi$, podemos provar a conclusão traduzida dos axiomas traduzidos.

A observação crucial é de que assinaturas nada mais são do que conjuntos de declarações de símbolos não lógicos, como é normalmente introduzido em qualquer apresentação de um sistema lógico. A questão essencial a ser entendida é que quando falamos em lógica para especificação estamos interessados em considerar de uma vez só todos esses conjuntos e também as funções que os relacionam. Em outras palavras precisamos reunir tudo isto em uma categoria.

Desta maneira, nesta seção, apresentamos uma adaptação do material desenvolvido na seção anterior, considerando a indexação dos componentes por assinaturas.

Esta parte do trabalho é essencialmente baseada no clássico *General Logics* [MES 89].

### 2.2.1 Sistema de Conseqüência

**Definição 2.5 (Sistema de Conseqüência )** Um sistema de conseqüência é uma tripla $\mathcal{SC} = \langle \mathsf{Sign}, Sen, \vdash \rangle$, onde $\mathsf{Sign}$ é uma categoria cujos objetos são chamados assinaturas, $Sen$ um funtor $Sen : \mathsf{Sign} \to \mathsf{Set}$ e $\vdash$ uma função associando a cada $\Sigma \in \mathsf{Sign}$ uma relação binária $\vdash_\Sigma \subseteq \wp\,(Sen(\Sigma)) \times Sen(\Sigma)$ chamada $\Sigma$-conseqüência , tal que as seguintes propriedades são satisfeitas:

1. Reflexividade: para qualquer $\varphi \in Sen(\Sigma), \{\varphi\} \vdash_\Sigma \varphi$;
2. Motonicidade: se $\Gamma \vdash_\Sigma \varphi$ e $\Gamma \subseteq \Gamma'$ então $\Gamma' \vdash_\Sigma \varphi$;
3. Transitividade: se $\Gamma \vdash_\Sigma \varphi_i$, para $i \in I$, e $\Gamma \cup \{\varphi_i \mid i \in I\} \vdash_\Sigma \psi$, então $\Gamma \vdash_\Sigma \psi$;
4. Tradução: se $\Gamma \vdash_\Sigma \varphi$, então para qualquer $\phi : \Sigma \to \Sigma' \in \mathsf{Sign}, Sen(\phi)(\Gamma) \vdash_{\Sigma'} Sen(\phi)(\varphi)$.

### 2.2.2 Instituição

**Definição 2.6 (Instituição)** Uma instituição é uma quadrupla $\langle \mathsf{Sign}, Sen, Mod, \models \rangle$, onde $\mathsf{Sign}$ é uma categoria cujos objetos são chamados assinaturas, $Sen$ um funtor $Sen : \mathsf{Sign} \to \mathsf{Set}$, $Mod$ um funtor $Mod : \mathsf{Sign} \to \mathsf{Cat}$ e $\models$ uma função associando a cada assinatura $\Sigma$ uma relação binária $\models_\Sigma \subseteq \mid Mod(\Sigma) \mid \times Sen(\Sigma)$, chamada satisfação, tal que para qualquer $M' \in Mod(\Sigma')$, para qualquer $\phi : \Sigma \to \Sigma'$ e para todo $\varphi \in Sen(\Sigma)$, o seguinte vale

$$M' \models_{\Sigma'} Sen(\phi)(\varphi) \Leftrightarrow Mod(\phi)(M') \models_\Sigma (\varphi).$$

Sabendo que a partir de uma instituição podemos derivar um sistema de conseqüência temos a seguinte

***Proposição 2.1*** *Um sistema de conseqüência derivado de uma instituição é uma tripla* $\langle \mathsf{Sign}, Sen, \vdash \rangle$, *tal que* $\vdash_\Sigma =^{def} \models_\Sigma$, *para todo* $\Sigma \in |\ \mathsf{Sign}\ |$ *(segundo a definição 3.2,1), onde* $\models_\Sigma \subseteq \wp\,(Sen(\Sigma) \times Sen(\Sigma))$.

*Prova.* Precisamos demonstrar as quatro condições da definição 2.5.

1. Reflexividade: seja $M \in |\ Mod(\Sigma)\ |$. Então se $M \models_\Sigma \{\varphi\}$ segue que $M \models_\Sigma \varphi$ (pelo corolário 3.1,1, no próximo capítulo), e estamos prontos.

2. Monotonicidade: seja $M \in |\ Mod(\Sigma)\ |$, tal que $M \models_\Sigma \Gamma'$. Mas pelo corolário 3.1,1 temos que $M \models_\Sigma \Gamma$, já que $\Gamma \subseteq \Gamma'$. Mas, pela premissa temos que se $M \models_\Sigma \Gamma$ então $M \models \varphi$, que era o que tinha que ser provado.

3. Transitividade: seja $M \in |\ Mod(\Sigma)\ |$, tal que $M \models_\Sigma \Gamma$. Pelas premissas temos que $M \models_\Sigma \Gamma$ e $M \models_\Sigma \varphi_i, i \in I$, e além disso que se $M \models_\Sigma \Gamma \cup \{\varphi_i \mid i \in I\}$, $M \models_\Sigma \psi$. Pelo corolário 3.1,2 $M \models_\Sigma \Gamma \cup \{\varphi_i \mid i \in I\}$ se e somente se $M \models_\Sigma \Gamma$ e $M \models_\Sigma \{\varphi_i \mid i \in I\}$. Mas como $M \models_\Sigma \Gamma$, pela hipótese, e $M \models_\Sigma \Gamma \Rightarrow M \models_\Sigma \varphi_i, i \in I$, segue que $M \models_\Sigma \Gamma$ e $M \models_\Sigma \varphi_i, i \in I$. Por *modus ponens* temos que $M \models_\Sigma \psi$.

4. Segue direto do lema 3.1 (monotonicidade de $Sen(\phi)$).

### 2.2.3 Lógica = Instituição + Sistema de Conseqüência

Uma lógica para especificação pode ser axiomatizada em uma formalização que englobe ambos aspectos dedutivos quanto os modelo teóricos. Assim, uma lógica tem dois componentes, que são: um *sistema de conseqüência* e uma *instituição* que compartilham as mesmas assinaturas e sentenças. Também, a lógica deve ser *consistente*, i.e.,

$$\Gamma \vdash \varphi \Rightarrow \Gamma \models \varphi.$$

Portanto, temos a seguinte

**Definição 2.7 (Lógica)** Uma lógica é uma 5-upla $\mathcal{L} = \langle \mathsf{Sign}, Sen, Mod, \vdash, \models \rangle$, tal que:

1. $\langle \mathsf{Sign}, Sen, \vdash \rangle$ é um sistema de conseqüência ;

2. $\langle \mathsf{Sign}, Sen, Mod, \models \rangle$ é uma instituição, e

3. a seguinte condição de consistência (*soundness*) é satisfeita: para qualquer $\Sigma \in \mathsf{Sign}$, $\Gamma \subseteq Sen(\Sigma)$ e $\varphi \in Sen(\Sigma)$,temos que $\Gamma \vdash_\Sigma \varphi \Rightarrow \Gamma \models_\Sigma \varphi$. Uma lógica é *completa*, se: $\Gamma \vdash_\Sigma \varphi \Leftarrow \Gamma \models_\Sigma \varphi$.

O conceito de instituição junto com a noção de sistema de conseqüência derivado induz a noção de lógica derivada.

***Corolário 2.1*** *Dada uma instituição* $\mathcal{I} = \langle \mathsf{Sign}, Sen, Mod, \models \rangle$ *e um sistema de conseqüência derivado* $\langle \mathsf{Sign}, Sen, \vdash \rangle$ *podemos definir uma lógica* $\mathcal{L}_\mathcal{I} = \langle \mathsf{Sign}, Sen, Mod, \models, \vdash \rangle$, *tal que*

$$\Gamma \vdash_\Sigma \varphi \Leftrightarrow \Gamma \models_\Sigma \varphi.$$

### 2.2.4 Sistema Lógico

Na definição de lógica vista até aqui, a estrutura (escore) das provas é abstraida. A única coisa que sabemos é que a partir de um conjunto de sentenças $\Gamma$ podemos derivar uma sentença $\varphi$. Nenhuma informação sobre a estrutura interna de geração dessa derivação é conhecida. A essência desta idéia é que a relação de conseqüência ($\vdash$) permanece *invariante* sob os cálculos diferentes que podem ser utilizados para uma lógica. Por exemplo, para *MSEqtl* temos diversos cálculos de provas, i.e., Hilbert, cálculo de sequentes, dedução natural, cada um deles com uma noção diferente de prova. Contudo, a *lógica MSEqtl* sempre permanece a mesma, pois todo e qualquer cálculo produz a mesma relação de conseqüência ($\vdash$). É neste sentido que é interessante termos uma definição de lógica sem a preocupação com um cálculo de provas particular. Segundo Meseguer, esta idéia é diretamente relevante para a ciência da computação, pois ela mostra que podemos trocar a semântica operacional (cálculo de provas) de uma linguagem de programação sem alterar sua semântica matemática, no sentido que a nova e a velha semântica operacional têm a mesma relação de conseqüência . No entanto, se precisamos escolher um cálculo específico para uma lógica temos a idéia de *sistema lógico*, que é axiomatizado como uma lógica junto com um cálculo de provas. Mais especificamente, como uma lógica $\mathcal{L}= \langle \mathsf{Sign}, Sen, Mod, \vdash, \models \rangle$ juntamente com um cálculo que gera a relação de conseqüência $\vdash$.
Na tabela a seguir resumimos os componentes de uma lógica de acordo com as duas visões discutidas anteriormente.

**Lógica Clássica & Lógica para Especificação**

| Lógica | | Lógica para Especificação |
|---|---|---|
| Sistema de Satisfação $\langle Form, M, \models \rangle$ | $\mapsto$ | Instituição $\langle \mathsf{Sign}, Sen, Mod, \models \rangle$ |
| Sistema de Conseqüência $\langle Form, \vdash \rangle$ | $\mapsto$ | Sistema de Conseqüência $\langle \mathsf{Sign}, Sen, \vdash \rangle$ |
| Sist. de Conseq. derivado $\langle Form, \models \rangle$ | $\mapsto$ | Sist. de Conseq. derivado $\langle \mathsf{Sign}, Sen, \models \rangle$ |
| Lógica $\langle Form, \vdash, \models, M \rangle$ | $\mapsto$ | Lógica $\langle \mathsf{Sign}, Sen, Mod, \models, \vdash \rangle$ |

## 2.3 Notas Bibliográficas

Na abordagem modelo-teórica, Goguen & Burstall [GOG 84] propõem uma visão unificadora de lógica para especificação com o conceito de Instituição. Em Meseguer [MES 89] encontramos uma extensão desta teoria com objetivo de levar em consideração outros componentes importantes de uma lógica, como por exemplo deduções e provas. Em [COU 97] é apresentado um estudo bastante detalhado sobre lógicas e seus relacionamentos sem a indexação por assinaturas.

# 3 Instituições

Este capítulo consiste basicamente na apresentação de *instituições* como uma formalização modelo teórica para a idéia de lógica para especificação. Primeiramente, temos uma discussão informal sobre os ingredientes básicos que são capturados por essa idéia. Neste sentido, salientamos que o conceito de instituição nos permite construir estruturas com as quais podemos escrever especificações. Seguindo, introduzimos a definição formal consistindo de uma categoria de assinaturas (Sign), de um funtor ($Sen : \mathsf{Sign} \to \mathsf{Set}$), de um funtor ($Mod : \mathsf{Sign} \to \mathsf{Set}$) e de uma relação de satisfação entre modelos e sentenças indexada por assinaturas. Os funtores *Sen* e *Mod* determinam respectivamente, o conjunto de sentenças e a categoria de modelos para cada assinatura. Além de citarmos diversos exemplos de instituições apresentamos, nesse capítulo, algumas definições derivadas que serão utilizadas no andamento do texto. Essas estruturas são essencialmente a noção de conseqüência semântica e alguns conceitos que envolvem teorias, i.e., categoria de teorias -Th-, subcategoria de modelos e funtor modelo generalizado. Os conceitos discutidos nesta parte da dissertação são baseados em textos clássicos como [GOG 92, GOG 84].

## 3.1 Definições Básicas

Na discussão da parte introdutória deste trabalho verificamos que as lógicas para especificação possuem características em comum, i.e., o conceito de *sintaxe*, *semântica* e a noção de *satisfação*. Neste sentido, essas idéias são capturadas pela definição de instituição.

De forma a motivar a noção de instituição considere o seguinte esquema para números naturais na *MSEqtl*.

$\Sigma Nat1, MSEqtl$

Sorts
- $nat$

Opns
- $0 :\to nat$
- $s : nat \to nat$
- $plus : nat, nat \to nat$

Formalmente, temos o conceito de assinatura dos naturais com zero, sucessor e adição. Agora, para esta assinatura temos a idéia de álgebra, i.e., uma estrutura que interpreta cada símbolo de sort como um conjunto e cada símbolo de operação como uma função. Assim, uma álgebra para esta assinatura *MSEqtl*, na notação representada pelo esquema abaixo

$[\![\ ]\!], Alg$

Funções

---

Definições

pode ser uma estrutura que interpreta o símbolo de sort *nat* da assinatura como o conjunto dos números naturais $\mathbb{N}$, o símbolo de constante 0 como a constante 0

dos naturais, o símbolo de operação $s$ como a função $succ$ e o símbolo $plus$ como a função $sum$ definida no esquema abaixo

$[\![Nat1, \Sigma]\!], Alg$

Conjuntos

$\mathbb{N}$

Fun

$0 : \{*\} \to \mathbb{N}$

$succ : \mathbb{N} \to \mathbb{N}$

$sum : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

---

$succ(x) = x + 1$

$sum(x, y) = x + y$

Note que o domínio para a função constante 0 é um conjunto unitário. Mais precisamente, considere, por exemplo, que temos um conjunto com dois elementos ($\{0, 1\}$). Seja $\{*\}$ um conjunto unitário fixo. Observe que é possível definir somente duas funções de $\{*\}$ para tal conjunto, uma que associa o único elemento $*$ à constante 0 e outra a constante 1. Portanto, existem duas funções que correspondem aos dois valores constantes, i.e., $0 : \{*\} \to \{0, 1\}$ e $1 : \{*\} \to \{0, 1\}$. Desta forma, temos uma maneira de *apontar* para o elemento desejado.

Quando trabalhamos com especificações, em muitos casos, precisamos mover de uma assinatura para outra adicionando ou removendo símbolos de sorts ou operações. Então, precisamos da noção de morfismo entre assinaturas. Por exemplo, considere que desejamos enriquecer a assinatura *MSEqtl* dos naturais com o símbolo de operação *mult*, i.e.,

$\Sigma Nat2, MSEqtl$

Sorts

$nat$

Opns

$0 :\to nat$

$s : nat \to nat$

$plus : nat, nat \to nat$

$mult : nat, nat \to nat$

Desta maneira, um morfismo ($\phi$) entre essas duas assinaturas pode ser pensado como uma função que relaciona cada símbolo de sort e símbolos de operações da especificação original com a nova. Neste exemplo, temos que todos os símbolos da primeira assinatura são mapeados para os mesmos na nova, visto que esta seta pode ser pensada como uma inclusão.

Agora, observe uma álgebra para a nova assinatura no esquema abaixo. Note que temos a mesma estrutura da assinatura original só que com uma função para interpretar o novo símbolo *mult*, que no caso é visto como a multiplicação dos naturais.

$[\![Nat2, \Sigma]\!], Alg$

Conjuntos

$\mathbb{N}$

Fun

$0 : \{*\} \to \mathbb{N}$

$succ : \mathbb{N} \to \mathbb{N}$

$sum : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

$times : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

---

$succ(x) = x + 1$

$sum(x, y) = x + y$

$times(x, y) = x * y$

Neste sentido, temos um morfismo entre as duas álgebras ($Mod(\phi)$) que é induzido pelo morfismo entre assinaturas ($\phi$). Contudo, observe no diagrama abaixo que esta seta traduz modelos no sentido contrário, i.e., o conjunto e funções que interpretam a nova assinatura são, agora, a origem do morfismo.

$$\langle \mathbb{N}, 0, succ, sum, times \rangle$$

$$\Big\downarrow Mod_{(\phi)}$$

$$\langle \mathbb{N}, 0, succ, sum \rangle$$

Assim, uma seta entre as estruturas de interpretação traduz os modelos que interpretam os símbolos mapeados pelo morfismo entre as assinaturas, mais especificamente, temos que a álgebra da assinatura destino é *também* uma álgebra da assinatura origem do morfismo entre assinaturas. Portanto, neste caso o funtor $Mod(\phi)$ *esquece* a função *times*. Além disso, para cada assinatura temos a noção de sentenças. Um morfismo entre sentenças é no mesmo sentido que $\phi$, relacionando sentenças da assinatura origem com as da assinatura destino, de acordo também com $\phi$.

Considere novamente a primeira assinatura *MSEqtl* de números naturais. Podemos desejar *mover* para uma assinatura de números inteiros, também com 0, sucessor e adição (ver esquema abaixo).

$\Sigma Int, MSEqtl$

Sorts

$int$

Opns

$0 :\to int$

$s : int \to int$

$plus : int, int \to int$

Então, um morfismo ($\phi$) entre essas duas assinaturas *traduz* o símbolo de sort *nat* para *int*. Os nomes dos símbolos de operações são mantidos os mesmos, no entanto seus sorts respectivos trocam de *nat* para *int*.

Uma álgebra para esta assinatura pode ser uma estrutura que interpreta o símbolo de sort *int* como o conjunto dos números inteiros $\mathbb{Z}$, e os símbolos de operações 0, $s$ e *plus* como a constante inteira 0, como a função sucessor em inteiros *succ* e como a função adição em inteiros *sum*, respectivamente. Logo, temos essa álgebra no seguinte esquema.

$[\![Int, \Sigma]\!], Alg$

Conjuntos

$\mathbb{Z}$

Fun

$0 : \{*\} \to \mathbb{Z}$

$succ : \mathbb{Z} \to \mathbb{Z}$

$sum : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$

---

$succ(x) = x + 1$

$sum(x, y) = x + y$

Um morfismo entre as estruturas de interpretação de assinaturas é sempre contravariante, i.e., no sentido contrário do morfismo entre assinaturas. Observe a tradução das álgebras, dessas duas assinaturas, no diagrama abaixo.

$$\begin{array}{c} \langle \mathbb{Z}, 0, succ, sum \rangle \\ \Big\uparrow Mod_{(\phi)} \\ \langle \mathbb{Z}, 0, succ, sum \rangle \end{array}$$

Novamente, nesta tradução, transparece a idéia de que um modelo da assinatura destino (respectiva a $\phi$) é também um modelo da assinatura origem. Particularmente neste caso, a álgebra destino de $Mod(\phi)$ é exatamente a mesma origem.

No primeiro exemplo apresentamos a idéia de um morfismo entre assinaturas que é uma inclusão (injeção), no segundo, o morfismo pode ser pensado como uma bijeção, assim o próximo exemplo que apresentaremos é um morfismo sobrejetor. Para isso, considere a seguinte assinatura *MSEqtl* de números naturais e valores e operações booleanas, no esquema a seguir.

$\Sigma NatB, MSEqtl$

Sorts

$nat, bool$

Opns

$0 :\to nat$

$true :\to bool$

$false :\to bool$

$s : nat \to nat$

$not : bool \to bool$

$and : bool, bool \to bool$

$plus : nat, nat \to nat$

Agora, queremos *relacionar* esta assinatura com uma assinatura somente de valores e operações booleanas, i.e.,

$\Sigma Bool$, *MSEqtl*

Sorts
- $bool$

Opns
- $true :\rightarrow bool$
- $false :\rightarrow bool$
- $not : bool \rightarrow bool$
- $and : bool, bool \rightarrow bool$

Definimos o morfismo $\phi$ de $\Sigma NatB$ para $\Sigma Bool$ como uma função que relaciona ambos símbolos de sort *nat* e *bool* da primeira assinatura com o *único* símbolo *bool* da segunda. Os símbolos de constantes 0 e *true* são também ambos mapeados para *true* e o símbolo *false* é traduzido para *false* em $\Sigma Bool$. Os símbolos $s$ e *not* vão para *not* na segunda assinatura. Finalmente, *and* e *plus* tem imagem *and*.

Uma álgebra para $\Sigma Bool$, interpreta o símbolo de sort *bool* como o conjunto de valores booleanos $\mathbb{B}$ tal que $\mathbb{B} = \{TRUE, FALSE\}$. Os símbolos de constantes *true* e *false* são as constantes $TRUE \in \mathbb{B}$ e $FALSE \in \mathbb{B}$, respectivamente. Os símbolos de operações *not* e *and* são as usuais operações booleanas $\neg$ (negação) e $\wedge$ (conjunção) (ver esquema abaixo).

$[\![Bool, \Sigma]\!]$, *Alg*

Conjuntos
- $\mathbb{B}$

Fun
- $TRUE : \{*\} \rightarrow \mathbb{B}$
- $FALSE : \{*\} \rightarrow \mathbb{B}$
- $\neg : \mathbb{B} \rightarrow \mathbb{B}$
- $\wedge : \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$

Desta forma, de acordo com o morfismo entre assinaturas, o morfismo $Mod(\phi)$ entre as álgebras é um mapeamento como representado no diagrama abaixo.

$$\langle \mathbb{B}, TRUE, FALSE, \neg, \wedge \rangle$$

$$Mod_{(\phi)} \downarrow$$

$$\langle \mathbb{B}, \mathbb{B}, TRUE, TRUE, FALSE, \neg, \neg, \wedge, \wedge \rangle$$

Note, neste último exemplo, que o relacionamento entre modelos não é mais uma seta que *esquece* ou traduz o *próprio* modelo, mas sim um morfismo que *duplica* o carregador e algumas constantes e operações. Esta duplicação acontece de acordo com a tradução ($\phi$) dos símbolos da assinatura.

É interessante mencionar que o morfismo entre assinaturas ($\phi$) é uma n-upla de *funções*, no caso de uma assinatura *MSEqtl* é uma dupla de funções, uma que mapeia sorts e outra que traduz símbolos de operações *preservando* a aridade. Para ilustrar esta situação considere que desejamos relacionar a última assinatura de valores booleanos $\Sigma Bool$, apresentada acima, com a seguinte assinatura de números naturais

$\Sigma Nat3, MSEqtl$

Sorts
$nat$
Opns
$0 :\rightarrow nat$
$s : nat \rightarrow nat$

Podemos pensar no morfismo $\phi$ com uma função que traduz o sort *bool* de $\Sigma Bool$ para o sort *nat* de $\Sigma Nat3$, os símbolos de constantes *true* e *false* para o símbolo 0 de $\Sigma Nat3$ e *not* para $s$. Até aqui tudo está de acordo, entretanto observe que na assinatura origem temos o símbolo de operação *and*, o qual tem aridade 2, e que na assinatura destino não temos nenhum símbolo de operação com aridade igual a 2. Como descrito acima o mapeamento $\phi$ para símbolo de operações é uma função que preserva a aridade então, neste caso não é possível estabelecer um morfismo entre essas duas assinaturas.

Toda essa discussão sobre relacionamentos entre assinaturas, álgebras e sentenças verificada nos exemplos apresentados, pode ser formalizada pela seguinte

**Definição 3.1 (Instituição)** Uma *instituição* é uma quádrupla $\mathcal{I} = (\mathsf{Sign}, Sen, Mod, \models)$ onde,

- $\mathsf{Sign}$ é uma categoria cujos objetos são *assinaturas* $\Sigma$ e as setas são morfismos entre assinaturas;
- $Sen : \mathsf{Sign} \rightarrow \mathsf{Set}$ é um funtor, associando à cada assinatura $\Sigma$ um conjunto de *sentenças* $Sen(\Sigma)$;
- $Mod : \mathsf{Sign}^{\mathsf{op}} \rightarrow \mathsf{Cat}$ é um funtor, atribuindo para cada assinatura uma correspondente categoria de modelos $Mod(\Sigma)$ e
- $\models$ é uma *relação de satisfação* indexada $\models_\Sigma \subseteq |Mod(\Sigma)| \times Sen(\Sigma)$, onde $|Mod(\Sigma)|$ denota a classe de todos os objetos na categoria $Mod(\Sigma)$ tal que a seguinte propriedade vale para qualquer seta $\phi : \Sigma_1 \rightarrow \Sigma_2$ em $\mathsf{Sign}$, para qualquer $M_2 \in |Mod(\Sigma_2)|$ e para qualquer $\varphi_1 \in Sen(\Sigma_1)$:

  $M_2 \models_{\Sigma_2} Sen(\phi)(\varphi_1) \iff Mod(\phi)(M_2) \models_{\Sigma_1} \varphi_1$ (Cond. de Satisfação) □

  Esta relação pode ser observada no seguinte diagrama

$$\begin{array}{ccc} \Sigma_1 & & Mod(\Sigma_1) \xleftarrow{\models_{\Sigma_1}} Sen(\Sigma_1) \\ \downarrow \phi & & Mod_{(\phi)} \uparrow \qquad\qquad \downarrow Sen_{(\phi)} \\ \Sigma_2 & & Mod(\Sigma_2) \xleftarrow[\models_{\Sigma_2}]{} Sen(\Sigma_2) \end{array}$$

Considere agora a Figura 3.1. Ela é composta essencialmente por três células e setas que estabelecem relacionamentos entre elas. A célula superior denota a categoria de assinaturas $\mathsf{Sign}$. A célula inferior à esquerda representa a categoria das categorias $\mathsf{Cat}$ e a inferior à direita a categoria dos conjuntos $\mathsf{Set}$. Temos duas setas que partem da célula superior, a que tem destino na célula inferior à esquerda e a que chega na inferior à direita. Esta primeira representa o funtor $Mod : \mathsf{Sign} \rightarrow \mathsf{Cat}$

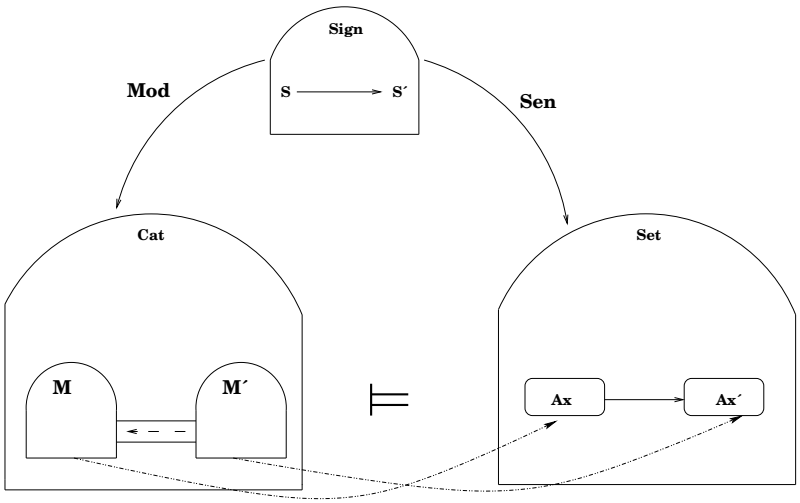



FIGURA 3.1 – Instituição

e a segunda o funtor $Sen$ : Sign $\rightarrow$ Set. Em Cat, as duas subcélulas denotam as categorias de modelos das respectivas assinaturas em Sign. Já em Set, as subcélulas são os conjuntos de sentenças dessas assinaturas. Note que a direção da seta entre as categorias de modelos é oposta a direção da seta que relaciona os conjuntos de sentenças, ilustrando a contravariância entre sintaxe e semântica. No centro do desenho, o símbolo $\models$ é a relação de satisfação. Esta relação junto com as setas pontilhadas abaixo denotam a invariância da relação de satisfação, i.e., a partir de um morfismo entre assinaturas em Sign, a satisfação de fórmulas por modelos é respeitada coerentemente.

A definição de instituição é tão geral quanto possível, descrevendo as características essenciais dos sistemas lógicos usuais. Por exemplo, as seguintes lógicas são instituições: *lógica de primeira ordem e suas variantes* [GOG 92], *lógicas temporais (linear, branching)* [ARR 96], *lógicas modais, cálculo lambda, lógica de alta ordem* [GOG 92], *Lisp puro, Prolog puro* [GOG 2001].

Considerando, novamente, a primeira assinatura *MSEqtl* para os números naturais ($\Sigma Nat1$), podemos construir a partir dela uma *teoria* (i.e., especificação) acrescentando os seguintes axiomas

$$[x : nat]plus(x, 0) = x : nat \quad A_1$$
$$[x, y : nat]plus(x, s(y)) = s(plus(x, y)) : nat \quad A_2$$

Essa teoria, com $\Gamma$ = conjunto de axiomas (equações), na notação do esquema abaixo

$$\begin{array}{|l} Th \\ \hline \Sigma \\ \hline \Gamma \\ \hline \end{array}$$

pode ser verificada na sequência

*ThNat*1, *MSEqtl*

Sorts
$nat$

Opns
$0 :\rightarrow nat$
$s : nat \rightarrow nat$
$plus : nat, nat \rightarrow nat$

$[x : nat]plus(x, 0) = x : nat \quad A_1$
$[x, y : nat]plus(x, s(y)) = s(plus(x, y)) : nat \quad A_2$

Nesta especificação, os nomes $A_1$ e $A_2$ no final de cada axioma são rótulos que utilizaremos a seguir, para identificar de qual axioma estamos falando. Observe que uma álgebra para esta teoria pode ser a mesma introduzida para a assinatura $\Sigma Nat1$, no início da seção, já que ela não é somente uma álgebra da assinatura, mas também uma álgebra da especificação. Uma álgebra $A$ de uma especificação, é uma álgebra $A$ da assinatura que satisfaz todos os axiomas (neste caso equações) em $\Gamma$. Como tratamos de uma especificação em *MSEqtl* temos equações como axiomas, entretanto em outras lógicas podemos ter também outros tipos de fórmulas. Isso motiva o conceito de satisfação de um axioma em uma álgebra. Um axioma (neste caso equação) é satisfeito por uma álgebra $A$ se e somente se a avaliação dos lados direito e esquerdo da equação produz o mesmo resultado em $A$. Por exemplo, considere a senteça $A_1$ da especificação *ThNat*1. Sejam $A_{ThNat1}$ a álgebra da especificação e $\alpha_{nat} : \{x\} \rightarrow \mathbb{N}$ uma atribuição árbitrária. Para verificar que a sentença $A_1$ é satisfeita pela álgebra $A_{ThNat1}$, temos que provar que: $\overline{\alpha}_{nat}(plus(x, 0)) = \overline{\alpha}_{nat}(x)$, para a avaliação $\overline{\alpha}$ segundo a definição 4.13. Sabendo que a avaliação é um homomorfismo, temos:

$$\begin{aligned} \overline{\alpha}_{nat}(plus(x,0)) &= sum(\overline{\alpha}_{nat}(x), \overline{\alpha}_{nat}(0)) && \text{(definição 4.13)} \\ &= \overline{\alpha}_{nat}(x) + 0 && \text{( definição de } sum) \\ &= \overline{\alpha}_{nat}(x) && \text{(definição de + )} \end{aligned}$$

## 3.2 Definições Derivadas

As estruturas derivadas apresentadas nesta seção são essencialmente a noção de consequência semântica e algumas definições que envolvem teorias. Na área de especificação algébrica *teorias* são representadas como especificações, i.e., assinatura junto com axiomas (ver *ThNat*1). Essas estruturas são principalmente importantes para as aplicações (ponto de vista do *usuário*), i.e., para construir especificações. É interessante salientar que existe um outro conceito de teoria sem a noção de representação, que chamamos de *teoria lógica* que é um conjunto de axiomas fechado sob a noção de consequência lógica. A maioria dos conceitos apresentados nesta seção são adaptados de [MAR 99].

Em instituições temos a noção de *satisfação*, i.e., abordagem modelo-teórica. Contudo, a partir de uma instituição podemos derivar uma relação de consequência entre um conjunto de fórmulas $\Gamma$ e uma fórmula $\varphi$, a qual chamamos de *consequência semântica*, que pode ser formalizada na seguinte

**Definição 3.2 (Consequência Semântica)**

1. Um modelo $M \in |Mod(\Sigma)|$ satisfaz um conjunto de sentenças $\Gamma \subseteq Sen(\Sigma)$, escrito $M \models_\Sigma \Gamma$, se e somente se $\forall\, \varphi \in \Gamma : M \models_\Sigma \Gamma$.

2. Uma sentença $\varphi \in Sen(\Sigma)$ é uma consequência semântica de um conjunto de sentenças $\Gamma \in sen(\Sigma)$, escrita $\Gamma \models_\Sigma \varphi$, se e somente se, $\forall\, M \in |Mod(\Sigma)| : M \models_\Sigma \Gamma \Longrightarrow M \models_\Sigma \varphi$.

3. Um conjunto de sentenças $\Delta \subseteq Sen(\Sigma)$ satisfaz a consequência semântica de um conjunto de sentenças $\Gamma \subseteq Sen(\Sigma)$, escrito $\Gamma \models_\Sigma \Delta$, se e somente se $\forall\, \varphi \in \Delta : \Gamma \models_\Sigma \varphi$.

***Corolário 3.1***

1. *Dados* $\Gamma, \Delta \subseteq Sen(\Sigma)$ *e* $\Delta \subseteq \Gamma$, *temos que* $M \models_\Sigma \Gamma \Rightarrow M \models_\Sigma \Delta$.

2. *Dados* $\Gamma, \Delta \subseteq Sen(\Sigma)$ *e* $M \in| Mod(\Sigma) |$, *temos que* $M \models_\Sigma \Gamma \cup \Delta \Leftrightarrow M \models_\Sigma \Gamma$ *e* $M \models_\Sigma \Delta$.

3. *Dados* $\Gamma_1, \Gamma_2, \Gamma_3 \subseteq Sen(\Sigma)$, *temos que*

   *se* $\Gamma_1 \models_\Sigma \Gamma_2$ *e* $\Gamma_2 \models_\Sigma \Gamma_3$, *então* $\Gamma_1 \models_\Sigma \Gamma_3$.

*Prova.* Temos que provar as três afirmações do corolário. Começamos com o item 1 e na sequência temos os itens 2 e 3.

1. Seja $M \models_\Sigma \Gamma$ pela hipótese. Então, pela definição 3.2,1, temos que $\forall\, \varphi \in \Gamma : M \models_\Sigma \varphi$. Como $\Delta \subseteq \Gamma$, segue que $\forall\, \varphi \in \Delta : M \models_\Sigma \varphi$. Novamente pela definição 3.2,1, temos que $M \models_\Sigma$, o que era para ser mostrado.

2. Primeiro, da esquerda para direita ($\Rightarrow$). Note que $M \models_\Sigma \Gamma \cup \Delta \Rightarrow M \models_\Sigma \Gamma$, pelo lema 3.1. Da mesma forma $M \models_\Sigma \Gamma \cup \Delta \Rightarrow M \models_\Sigma \Delta$. Logo, $M \models_\Sigma \Delta$ e $M \models_\Sigma \Gamma$.

   Seguindo a prova, da direita para a esquerda ($\Leftarrow$). Por hipótese temos que $M \models_\Sigma \Delta$ e $M \models_\Sigma \Gamma$. Agora, note que se $M \models_\Sigma \Delta \cup \Gamma$ então $\forall\, \varphi \in \Delta \cup \Gamma : M \models_\Sigma \varphi$. Pela definição de conjuntos temos que considerar dois casos, já que $\varphi \in \Delta \cup \Gamma$ então $\varphi \in \Delta$ ou $\varphi \in \Gamma$ (ou ambos). Segue de $\varphi \in \Delta$ e pela hipótese $M \models_\Sigma \Delta$ que $M \models_\Sigma \varphi$. Logo, por vacuidade $M \models_\Sigma \varphi$.

3. Seja $M \in| Mod(\Sigma) |$ um modelo arbitrário. Temos que mostrar que se $M \models_\Sigma \Gamma_1$ então $M \models_\Sigma \Gamma_3$. Mas por hipótese sabemos que se $M \models_\Sigma \Gamma_1$ então $M \models_\Sigma \Gamma_2$. Novamente pela hipótese, como $M \models_\Sigma \Gamma_2$ então $M \models_\Sigma \Gamma_3$, o que conclui a prova já que $M$ é arbitrário.

Seguindo, apresentamos mais algumas estruturas sintáticas e semânticas que são baseadas em uma instituição arbitrária $\mathcal{I}$.

Além da definição de *teoria* apresentamos o *funtor modelo generalizado*, que tem como origem a *categoria de teorias* e como destino $\mathsf{Cat}$. Contudo, precisamos primeiramente definir um importante operador, que define uma *subcategoria cheia* de $Mod(\Sigma)$. Nesta subcategoria vamos ter os $\Sigma$-modelos que satisfazem um conjunto de sentenças $\Gamma$.

**Definição 3.3 (Subcategoria de Modelos)** Dada uma assinatura $\Sigma \in |\ \mathsf{Sign}\ |$ e um conjunto de sentenças $\Gamma \subseteq Sen(\Sigma)$, definimos a categoria $mod(\Sigma)(\Gamma)$ como a *subcategoria cheia* de $Mod(\Sigma)$ determinada por aqueles modelos $M \in |\ Mod(\Sigma)\ |$ que satisfazem todas as sentenças em $\Gamma$, isto é, a parte dos objetos é definida como

$$|\ mod(\Sigma)(\Gamma)\ | =^{def} \{M \in |\ Mod(\Sigma)\ |\ |\ \ M \models_\Sigma \Gamma\}.$$

A preservação de consequência lógica pelos morfismo entre assinaturas é representada no seguinte

***Lema 3.1*** *Sejam $\Gamma_1, \Gamma_2 \subseteq Sen(\Sigma_1)$ e $\phi : \Sigma \to \Sigma_2$ um morfismo entre assinaturas em* Sign. *Então o seguinte vale:*

$$\Gamma_1 \models_{\Sigma_1} \Gamma_2 \Rightarrow Sen(\phi)(\Gamma_1) \models_{\Sigma_2} Sen(\phi)(\Gamma_2).$$

*Prova.* Por duas aplicações da condição de satisfação e pela hipótese, segue que, para todo $M_2 \in |\ Mod(\Sigma_2)\ |$, $M_2 \models_{\Sigma_2} Sen(\phi)(\Gamma_1) \Leftrightarrow Mod(\phi)(M_2) \models_{\Sigma_1} \Gamma_1 \Rightarrow Mod(\phi)(M_2) \models_{\Sigma_1} \Gamma_2 \Leftrightarrow M_2 \models_{\Sigma_2} Sen(\phi)(\Gamma_2)$.

**Definição 3.4 (Teoria)** Uma representação de uma *teoria* é um par $T = \langle \Sigma, \Gamma \rangle$, tal que $\Sigma \in |\ \mathsf{Sign}\ |$, $\Gamma \subseteq Sen(\Sigma)$.

**Definição 3.5 (Categoria Th)** A categoria Th de teorias e morfismos que preservam axiomas tem:

- $T = \langle \Sigma, \Gamma \rangle$ como objetos;
- setas $\langle \phi, \models \rangle : \langle \Sigma, \Gamma \rangle \to \langle \Sigma', \Gamma' \rangle$, tal que $\phi : \Sigma \to \Sigma'$ é um morfismo entre assinaturas em Sign e $\Gamma' \models_{\Sigma'} Sen(\phi)(\Gamma)$;
- o morfismo identidade é definido como: $id : \langle id_\Sigma, \models \rangle : \langle \Sigma, \Gamma \rangle \to \langle \Sigma, \Gamma \rangle$, tal que $id_\Sigma : \Sigma \to \Sigma$ é um morfismo em Sign e $\Gamma \models_\Sigma Sen(id)(\Gamma)$. Como $Sen$ é um funtor $\Gamma \models_\Sigma id_{Sen}(\Gamma) = \Gamma$;
- composição: sejam $\langle \phi, \models \rangle : \langle \Sigma, \Gamma \rangle \to \langle \Sigma', \Gamma' \rangle$ e $\langle \psi, \models \rangle : \langle \Sigma', \Gamma' \rangle \to \langle \Sigma'', \Gamma'' \rangle$ setas em Th, então temos que $\Gamma' \models_{\Sigma'} Sen(\phi)(\Gamma)$ e $\Gamma'' \models_{\Sigma''} Sen(\psi)(\Gamma')$, tal que a composição é $\Gamma'' \models_{\Sigma''} Sen(\psi \circ \phi)(\Gamma)$.

**Nota 3.1** Temos que verificar se $\Gamma'' \models_{\Sigma''} Sen(\psi \circ \phi)(\Gamma)$ é uma seta em Th. Sejam $\Gamma' \models_{\Sigma'} Sen(\phi)(\Gamma)$ e $\Gamma'' \models_{\Sigma''} Sen(\psi)(\Gamma')$. Assim pelo lema 3.1 (monotonicidade de $Sen(\psi)$ em relação a $\models$), temos que $Sen(\psi)(\Gamma') \models_{\Sigma''} Sen(\psi)(Sen(\phi)(\Gamma))$. Agora, pela transitividade de $\models_{\Sigma''}$ (ver corolário 3.1.3), temos que $\Gamma'' \models_{\Sigma''} Sen(\psi)(\Gamma')$ e $Sen(\psi)(\Gamma') \models_{\Sigma''} Sen(\psi)(Sen(\phi)(\Gamma))$ implica que $\Gamma'' \models_{\Sigma''} Sen(\psi)(Sen(\phi)(\Gamma))$. Agora pela definição de composição do funtor $Sen$ temos $\Gamma'' \models_{\Sigma''} Sen(\psi \circ \phi)(\Gamma)$, que é o que devia ser provado.

Cada morfismo entre teorias possui uma denotação, que é um funtor que mapeia modelos da teoria destino para modelos da teoria origem. Devido a condição de satisfação de uma instituição, podemos definir um funtor como uma restrição do respectivo funtor que é a denotação do morfismo entre assinaturas. Em detalhes, temos essa idéia capturada pela seguinte

***Proposição 3.1 (Funtor Modelo Generalizado)*** *Dada uma teoria* $\langle \Sigma, \Gamma \rangle \in |$ $\mathsf{Th}$ $|$, *a atribuição* $Mod_{\models}(\langle \Sigma, \Gamma \rangle) =^{def} mod(\Sigma)(\Gamma)$ *determina um funtor* $Mod_{\models} : \mathsf{Th}^{op} \to \mathsf{Cat}$, *chamado funtor modelo generalizado.*

**Nota 3.2** Para verificar a proposição acima consideramos $\phi : \langle \Sigma_1, \Gamma_1 \rangle \to \langle \Sigma_2, \Gamma_2 \rangle$ um morfismo entre teorias em $\mathsf{Th}$. Então, pela definição de morfismo entre teorias, temos que $\Gamma_2 \models_{\Sigma_2} Sen(\phi)(\Gamma_1)$. Assim, para qualquer $M_2 \in Mod_{\models}\langle \Sigma_2, \Gamma_2 \rangle)$, segue que $M_2 \models_{\Sigma_2} Sen(\phi)(\Gamma_1)$, e pela condição de satisfação , temos que $Mod(\phi)(M_2) \models_{\Sigma_1} \Gamma_1$, isto é $Mod(\phi)(M_2) \in | Mod_{\models}(\langle \Sigma_1, \Gamma_1 \rangle) |$. Isto significa que o funtor

$$Mod(\phi) : Mod(\Sigma_2) \to Mod(\Sigma_1)$$

se restringe ao funtor $Mod_{\models}(\phi) : Mod_{\models}\langle \Sigma_2, \Gamma_2 \rangle \to Mod_{\models}\langle \Sigma_1, \Gamma_1 \rangle)$.

$$\begin{array}{ccccc}
\langle \Sigma_1, \Gamma_1 \rangle & & Mod_{\models}(\langle \Sigma_1, \Gamma_1 \rangle) & \hookrightarrow & Mod(\Sigma_1) \\
\phi \downarrow & & Mod_{(\phi)\models} \uparrow & & Mod_{(\phi)} \uparrow \\
\langle \Sigma_2, \Gamma_2 \rangle & & Mod_{\models}(\langle \Sigma_2, \Gamma_2 \rangle) & \hookrightarrow & Mod(\Sigma_2)
\end{array}$$

## 3.3 Notas Bibliográficas

A teoria das instituições foi primeiramente introduzida por Goguen & Burstal [GOG 84], para descrever de uma maneira uniforme uma diversidade de sistemas lógicos utilizados para especificação. Esta idéia foi essencial no trabalho com a linguagem de especificação Clear [BUR 80], suportando modularização e parametrização para estruturar o reuso de especificações formais. O trabalho em instituições foi posteriormente enriquecido por seus autores em [GOG 92], na área de teoria abstrata dos modelos. Além disso, essas idéias têm também infuenciado outros trabalhos de autores diferentes, como por exemplo: *Foundations* [POI 89], $\pi-$*institutions* [FIA 87], *General Logics* [MES 89]. Uma interessante apresentação unificada de sistemas lógicos para especificação pode ser encontrada em [WOL 94].

# 4 Instituições I

Neste capítulo mostramos em detalhes duas instituições que são especialmente relevantes neste texto, i.e.,*lógica equacional multi-sortida* e *lógica equacional ordenada por sorts*. Para a primeira lógica, introduzimos o funtor *Sen*, o funtor *Mod* e a relação de satisfação indexada por assinaturas. Em relação ao funtor *Sen* são vistos os conceitos de termos e fórmulas nessas lógicas, bem como a tradução destes induzida pelo morfismo entre assinaturas. Na seção respectiva ao funtor *Mod* cuidado especial é dado ao *funtor esquecimento*, que pode ser visto como uma seta entre as categorias de modelos de assinaturas que estão relacionadas por morfismos em Sign. Uma idéia interessante que pode ser analisada cuidadosamente nessa instituição é o conceito de atribuições correspondentes, já que elas têm papel essencial para provar a relação de satisfação. Para a *lógica equacional ordenada por sorts* apresentamos somente a categoria de assinaturas devido a similaridade dos demais conceitos com a instituição discutida anteriormente. Salientamos que as seções são estruturadas com uma discussão informal sobre cada uma das lógicas no primeiro momento, e então seguindo, temos as definições formais.

## 4.1 Instituição da *Lógica Equacional Multi-Sortida*

De forma a introduzir uma discussão informal da instituição da *lógica equacional multi-sortida* (daqui em diante, simplesmente *MSEqtl*), considere as seguintes especificações.

*ThString*, *MSEqtl*

Sorts
$string, nat$

Opns
$0 :\rightarrow nat$
$vazia :\rightarrow string$
$s : nat \rightarrow nat$
$cons : nat, string \rightarrow string$
$first : string \rightarrow nat$
$rest : string \rightarrow string$

$[x : nat, w : string]\ first(cons(x\ w)) = x : nat\ A_1$
$[x : nat, w : string]\ rest(cons(x\ w)) = w : string\ A_2$

Primeiramente, temos uma especificação da estrutura de dados *string* de números naturais, com *string* e *nat* como símbolos de sorts. Observe que os únicos símbolos de operações sobre o sort *nat* são 0 e $s$. O símbolo de operação *cons* especifica o construtor para o sort *string*. Além disso, note que os axiomas sobre os símbolos *first* e *rest* especificam *o que* deve fazer uma possível função que interpretá-los. Os nomes $A_1$ e $A_2$ no final de cada equação, no esquema acima, são rótulos que serão utilizados a seguir para identificar de qual equação estamos falando. Agora, considere a seguinte especificação de *listas* de números inteiros.

*ThList, MSEqtl*

Sorts

$list, int$

Opns

$0 :\rightarrow int$
$empty :\rightarrow list$
$s : int \rightarrow int$
$plus : int, int \rightarrow int$
$tail : list \rightarrow list$
$head : list \rightarrow int$
$cons : int, list \rightarrow list$

$[x : int]\ plus(x, 0) = x : int\ B_1$
$[x, y : int]\ plus(x, s(y)) = s(plus(x, y)) : int\ B_2$
$[x : int, l : list]\ head(conc(x\ l)) = x : int\ B_3$
$[x : int, l : list]\ tail(conc(x\ l)) = l : list\ B_4$

Nesta teoria, para o sort *int*, além dos símbolos de operações 0 e $s$ temos também *plus*. Novamente, o símbolo *cons* especifica o construtor, só que agora para o sort *list*. Note que *empty* é o único símbolo de constante para *list*. Novamente, marcamos cada equação com um rótulo $(B_1, B_2, B_3, B_4)$ para identificação.

Observando essas duas teorias, podemos desejar estabelecer um *morfismo* $(\phi)$ entre as assinaturas $(\Sigma_{String}, \Sigma_{List})$ das duas especificações. Note, nos esquemas apresentados, que a assinatura está antes do traço que marca o início das equações. Mais precisamente, um mapeamento entre assinaturas traduz sorts e símbolos de funções de $\Sigma_{String}$ para $\Sigma_{List}$. Então, temos as seguintes traduções

$\phi(string) = list$
$\phi(nat) = int$
$\phi(0) = 0$
$\phi(vazia) = empty$
$\phi(s) = s$
$\phi(cons) = cons$
$\phi(first) = head$
$\phi(rest) = tail$

Neste contexto, podemos pensar também em traduções de sentenças do conjunto de sentenças de *ThString* para o conjunto *ThList*. Assim, temos um $\overline{\phi}$, que troca os nomes de acordo com o morfismo entre assinaturas $(\phi)$, tal que:

$\overline{\phi}([x : nat, s : string], first(cons(x\ w)) = x : nat) =$
$= ([x : int, l : list], head(cons(x\ l)) = x : int)$
$\overline{\phi}([x : nat, s : string], rest(cons(x\ w)) = w : string) =$
$= ([x : int, l : list], tail(cons(x\ l)) = l : list)$

Uma álgebra para especificação *ThString* é a seguinte estrutura,

$[\![String, MSEqtl]\!], Alg$

Conjuntos

$\mathbb{N}, \mathbb{N}^*$

Fun

$0 : \{*\} \to \mathbb{N}$
$\lambda : \{*\} \to \mathbb{N}^*$
$succ : \mathbb{N} \to \mathbb{N}$
$:: : \mathbb{N} \times \mathbb{N}^* \to \mathbb{N}^*$
$hd : \mathbb{N}^* \to \mathbb{N}$
$tl : \mathbb{N}^* \to \mathbb{N}^*$

---

$hd(n :: y) = n$
$tl(n :: y) = y$

que interpreta o símbolo de sort *nat* como o conjunto dos números naturais $\mathbb{N}$, o sort *list* como *palavras* de naturais, o símbolo de operação 0 como a constante 0 dos naturais, o símbolo *vazia* como a palavra vazia de números naturais $\lambda$, o símbolo de operação *first* como a função *hd* e finalmente *rest* como a função *tl*.

Como já mencionado no capítulo anterior, quando trabalhamos com especificações e estruturas de interpretação para elas, temos o conceito de satisfação de fórmulas (da especificação) pela álgebra. Logo, a satisfação de fórmulas em *ThString* é tal que: seja $A_{String}$ a estrutura de interpretação *MSEqtl* definida acima, e sejam

$$\alpha_{nat} : \{x\} \to \mathbb{N}$$
$$\alpha_{str} : \{w\} \to \mathbb{N}^*$$

atribuições arbitárias. Para provar que a sentença $A_1$ é satisfeita pela álgebra, significa verificar que $\overline{\alpha}_{str}(first(cons(x\ w)) = \overline{\alpha}_{nat}(x)$, para a avaliação $\overline{\alpha}$ segundo a definição 4.13. Então, temos:

$$\begin{array}{lll} \overline{\alpha}_{str}(first(cons(x\ w)) & = hd(\overline{\alpha}_{str}(cons(x\ w))) & \text{(definição 4.13)} \\ & = hd(\overline{\alpha}_{nat}(x) :: \overline{\alpha}_{str}(w)) & \text{(definição 4.13)} \\ & = \overline{\alpha}_{nat}(x) & \text{(definição de } hd) \end{array}$$

Agora para a sentença $A_2$ é preciso verificar que $\overline{\alpha}_{str}(rest(cons(x\ \ w))) = \overline{\alpha}_{str}(w)$. Assim, temos:

$$\begin{array}{lll} \overline{\alpha}_{str}(rest(cons(x\ w)) & = tl(\overline{\alpha}_{rest}(cons(x\ w)) & \text{(definição 4.13)} \\ & = tl(\overline{\alpha}_{nat}(x) :: \overline{\alpha}_{str}(w)) & \text{(definição 4.13)} \\ & = \overline{\alpha}_{str}(w) & \text{(definição de } tl) \end{array}$$

Na sequência, considere a álgebra para especificação *ThList* no seguinte esquema.

$[\![List, MSEqtl]\!], Alg$

Conjuntos

$\mathbb{Z}, \mathbb{Z}^*$

Fun

$0 : \{*\} \to \mathbb{Z}$
$\lambda : \{*\} \to \mathbb{Z}^*$
$succ : \mathbb{Z} \to \mathbb{Z}$
$sum : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$
$hd : \mathbb{Z}^* \to \mathbb{Z}$
$tl : \mathbb{Z}^* \to \mathbb{Z}^*$
$:: : \mathbb{Z} \times \mathbb{Z}^* \to \mathbb{Z}^*$

---

$succ(n) = n + 1$
$sum(n, m) = n + m$
$hd(n :: l) = n$
$tl(n :: l) = l$

Temos que o símbolos de sorts *int* e *list* são interpretados pelo conjunto dos números inteiros e por palavras de inteiros, respectivamente. Nesta estrutura de interpretação, as funções 0 e *succ* que denotam 0 e $s$ são definidas sob números inteiros, diferentes das definidas na álgebra anterior que eram sob naturais.

Para este segundo exemplo, novamente temos a noção de satisfação dos axiomas da teoria $ThList$ pela álgebra $A_{List}$ definida acima. Assim, sejam

$$\alpha_{nat} : \{x, y\} \to \mathbb{Z}$$
$$\alpha_{list} : \{l\} \to \mathbb{Z}^*$$

atribuições arbitárias. Para provar que a sentença $B_1$ é satisfeita pela álgebra significa verificar que $\overline{\alpha}_{nat}(plus(x, 0)) = \overline{\alpha}_{nat}(x)$. Logo, temos:

$$\begin{aligned}
\overline{\alpha}_{nat}(plus(x,0)) &= sum(\overline{\alpha}_{nat}(x), \overline{\alpha}_{nat}(0)) && \text{(definição 4.13)}\\
&= \overline{\alpha}_{nat}(x) + \overline{\alpha}_{nat}(0) && \text{(definição de } sum)\\
&= \overline{\alpha}_{nat}(x) + 0 && \text{(definição 4.13)}\\
&= \overline{\alpha}_{nat}(x) && \text{(definição de } +)
\end{aligned}$$

Para a sentença $B_2$ temos $\overline{\alpha}_{nat}(plus(x, s(y))) = \overline{\alpha}_{nat}(s(plus(x, y)))$. Assim,

$$\begin{aligned}
\overline{\alpha}_{nat}(plus\ (x, s(y))) &= sum(\overline{\alpha}_{nat}(x), \overline{\alpha}_{nat}(s(y))) && \text{(definição 4.13)}\\
&= \overline{\alpha}_{nat}(x) + \overline{\alpha}_{nat}(s(y)) && \text{(definição de } sum)\\
&= \overline{\alpha}_{nat}(x) + (succ(\overline{\alpha}_{nat}(y))) && \text{(definição 4.13)}\\
&= \overline{\alpha}_{nat}(x) + \overline{\alpha}_{nat}(y) + 1 && \text{(definição de } succ)
\end{aligned}$$

Agora, para o lado direito de $B_2$ temos:

$$\begin{aligned}
\overline{\alpha}_{nat}(s(plus(x, y))) &= succ(\overline{\alpha}_{nat}(plus(x, y))) && \text{(definição 4.13)}\\
&= succ(sum(\overline{\alpha}_{nat}(x), \overline{\alpha}_{nat}(y)) && \text{(definição 4.13)}\\
&= succ(\overline{\alpha}_{nat}(x) + \overline{\alpha}_{nat}(y)) && \text{(definição de } sum)\\
&= \overline{\alpha}_{nat}(x) + \overline{\alpha}_{nat}(y) + 1 && \text{(definição de } succ)
\end{aligned}$$

que é o que tinha que ser provado. Seguindo, para $B_3$ temos $\overline{\alpha}_{list}(head(conc(x\ l))) = \overline{\alpha}_{nat}(x)$. Assim,

$$\begin{aligned}
\overline{\alpha}_{list}(head(conc(x\ l))) &= hd(\overline{\alpha}_{list}(conc(x\ l))) && \text{(definição 4.13)}\\
&= hd(\overline{\alpha}_{nat}(x) :: \overline{\alpha}_{list}(l)) && \text{(definição 4.13)}\\
&= \overline{\alpha}_{nat}(x) && \text{(definição de } hd)
\end{aligned}$$

Finalmente, para $B_4$, precisamos verificar que $\overline{\alpha}_{list}(tail(conc(x\ l))) = \overline{\alpha}_{list}(l)$. Então:

$$\begin{array}{lll} \overline{\alpha}_{list}(tail(conc(x\ l))) & = tl(\overline{\alpha}_{list}(conc(x\ l)) & \text{(definição 4.13)} \\ & = tl(\overline{\alpha}_{nat}(x) :: \overline{\alpha}_{list}(l)) & \text{(definição 4.13)} \\ & = \overline{\alpha}_{list}(l) & \text{(definição de } tl) \end{array}$$

Desta maneira, provamos que essas duas álgebras são modelos das respectivas especificações.

Agora, voltando à idéia inicial de *relacionamento* entre as especificações, podemos definir um morfismo, $Mod(\phi) : A_{List} \rightarrow A_{String}$, (induzido pelo morfismo entre assinaturas) entre essas duas álgebras que é dado no *sentido contrário*, i.e., de $A_{List}$ para $A_{String}$. Portanto, tratamos da seguinte tradução:

$$\begin{array}{c} A_{List} = \langle \mathbb{Z}, \mathbb{Z}^*, 0, \lambda, succ,\ ::, sum, hd, tl \rangle \\ \Big\downarrow {\scriptstyle Mod(\phi)} \\ \langle \mathbb{Z}, \mathbb{Z}^*, 0, \lambda, succ,\ ::, hd, tl \rangle \end{array}$$

onde a função *sum* é *esquecida*. É importante notar que esta seta traduz os modelos de $A_{List}$ que satisfazem os axiomas relacionados por $\overline{\phi}$.

Contudo, é essencial que a satisfação de fórmulas por modelos seja *estável sob a troca de notação* pelo morfismo entre assinaturas, i.e., se temos que um modelo satisfaz os axiomas traduzidos, então, para qualquer tradução $\phi$, temos que o modelo traduzido satisfaz os axiomas origem. Assim, na tradução das especificações acima temos:

$$A_{List} \models_{\Sigma_{List}} Sen(\phi)(\varphi) \Leftrightarrow Mod(\phi)(A_{List}) \models_{\Sigma_{String}} \varphi$$

Esta discussão motiva pensarmos em assinaturas como objetos em uma categoria e morfismos entre assinaturas como setas nesta categoria. Também, podemos expressar de maneira estruturada, o processo que associa cada assinatura com seu conjunto de fórmulas e com sua classe de modelos, i.e., através de dois funtores. Os dois com origem na categoria de assinaturas e com destino em Set e Cat, respectivamente.

Isto tudo pode ser formalizado nas definições apresentadas nesta seção.

### 4.1.1 Categoria $\mathsf{Sign}_{MSEqtl}$ de Assinaturas

A categoria de assinaturas *MSEqtl*, tem assinaturas *MSEqtl* como objetos e morfismos entre assinaturas como setas. Uma assinatura *MSEqtl* consiste de um conjunto com nomes de sorts e de uma família de conjuntos com símbolos de operações indexada pela aridade e sort codomínio desses símbolos. Neste sentido, aridade pode ser pensada como uma *palavra* de sorts. Essa idéia pode ser verificada em detalhes na seguinte

**Definição 4.1 (Assinatura)** Uma assinatura *MSEqtl* (equacional) é um par $\Sigma = \langle S, F \rangle$, onde:

- $S$ é um conjunto de nomes de sorts,
- $F$ é um família de conjuntos de nomes de operadores, indexada por $S^* \times S$. Assim, $F = \langle F_{w,s} \mid w \in S^*, s \in S \rangle$. Um operador $f \in F_{w,s}$ com aridade $w$ e sort $s$ é escrito $f : w \to s$.

Um morfismo entre tais assinaturas *MSEqtl* consiste de uma função que traduz símbolos de sorts e uma família de funções que mapeia símbolos de operações. Mais precisamente, este morfismo dá para cada sort e para cada símbolo de operação da assinatura origem um respectivo na assinatura destino.

**Definição 4.2 (Morfismo entre Assinaturas *MSEqtl*)** Dadas duas assinaturas $\Sigma_1 = \langle S_1, F_1 \rangle$ e $\Sigma_2 = \langle S_2, F_2 \rangle$. Um morfismo $\phi : \Sigma_1 \to \Sigma_2$ é um par $\langle \phi^S, \phi^F \rangle$, onde:

- $\phi^S$ é um mapeamento $\phi^S : S_1 \to S_2$ de sorts
- $\phi^F$ é uma família de mapeamentos de símbolos de funções indexada por $S^* \times S$, tal que: $\phi^F = \langle \phi_{w,s} : F_{1,w,s} \to F_{2,\phi^*(w),\phi(s)} \rangle$, onde $\phi^*$ é a extensão de $\phi^S : S \to S$ para palavras de $S^*$, isto é, $\phi^*(s_1 \dots s_n) = \phi(s_1) \dots \phi(s_n)$ para todo $s_1 \dots s_n \in S^*$, lembrando que $\phi^*(\lambda) = \lambda$

**Nota 4.1** Para a notação ficar mais clara, escrevemos $\phi(s)$ no lugar de $\phi^S(s)$, $\phi(f)$ no lugar de $\phi^F(f)$ e $f : s_1 \dots s_n \to s$ como $f : w \to s$ se $w = s_1 \dots s_n$.

Tendo os conceitos de assinaturas e morfismo entre assinaturas *MSEqtl* podemos definir então o primeiro ingrediente da instituição desta lógica, i.e., categoria $\mathsf{Sign}_{MSEqtl}$.

**Definição 4.3 (Categoria $\mathsf{Sign}_{MSEqtl}$)** A **categoria** $\mathsf{Sign}_{MSEqtl}$ tem:

- assinaturas equacionais ($\langle S, F \rangle$) (segundo a definição 4.1) como objetos,
- morfismos entre assinaturas como morfismos,
- a composição dos morfismos é a composição dos mapeamentos nos componentes, i.e., sejam

  $$\phi : \Sigma_1 \to \Sigma_2 = \langle \phi^S, \phi^F \rangle \text{ e } \psi : \Sigma_2 \to \Sigma_3 = \langle \psi^S, \psi^F \rangle$$

  morfismos em $\mathsf{Sign}$, onde

  $$\phi^S : S_1 \to S_2 \text{ e } \phi^F : F_1 \to F_2$$

  que é definida como:

  $$\phi^F = \langle \phi_{w,s} : F_{1,w,s} \to F_{2,\phi^*(w),\phi(s)} \rangle$$

  para todo $w \in S_1^*$ e $s \in S_1$, onde $\phi^* : S_1^* \to S_2^*$. Igualmente para o morfismo $\psi$, temos

  $$\psi^S : S_2 \to S_3 \text{ e } \psi^F : F_2 \to F_3$$

  que é definida como:

  $$\psi^F = \langle \psi_{w,s} : F_{2,w,s} \to F_{3,\psi^*(w),\psi(s)} \rangle$$

para todo $w \in S_2^*$ e $s \in S_2$, onde $\psi^* : S_2^* \to S_3^*$. Então, a composição desses mapeamentos é tal que:

$$\psi \circ \phi : \Sigma_1 \to \Sigma_3 =^{Def} (\psi^S \circ \phi^S, \psi^F \circ \phi^F)$$

definida para cada componente dos morfismos. Assim, a definição da composição para a componente $S$ é

$$(\psi^S \circ \phi^S)(s) = \psi(\phi(s)) \in S_3$$

para todo $s \in S_1$, e para a componente $F$ é

$$(\psi^F \circ \phi^F)_{w,s} =^{Def} (\psi_{\phi^*(w),\phi(s)} \circ \phi_{w,s}).$$

Note que, pela definição da composição na componente $F$, temos que:

$$(\psi^F \circ \phi^F)_{w,s} : F_{1,w,s} \to F_{3,\psi^*(\phi^*(w)),\psi(\phi(s))}.$$

Isto pode ser verificado nos seguintes diagramas. Onde, para $\phi$, temos

$$\begin{array}{ccc} S_1 & S_1^* \times S_1 & F_{1,w_1,s_1} \\ \phi \downarrow & \downarrow \phi^* \times \phi & \downarrow \phi_{w_1,s_1} \\ S_2 & S_2^* \times S_2 & F_{2,\phi^*(w_1),\phi(s_1)} \end{array}$$

para $\psi$ temos

$$\begin{array}{ccc} S_2 & S_2^* \times S_2 & F_{2,w,s} \\ \psi \downarrow & \downarrow \psi^* \times \psi & \downarrow \psi_{w_2,s_2} \\ S_3 & S_3^* \times S_3 & F_{3,\psi^*(w_2),\psi(s_2)} \end{array}$$

finalmente, para composição $(\psi \circ \phi)$

$$\begin{array}{ccc} S_1 & S_1^* \times S_1 & F_{1,w_1,s_1} \\ (\psi\circ\phi) \downarrow & \downarrow (\psi\circ\phi)^* \times (\psi\circ\phi) & \downarrow \psi_{\phi^*(w_1),\phi(s_1)} \circ \phi_{w_1,s_1} \\ S_3 & S_3^* \times S_3 & F_{3,\psi^*(\phi^*(w_1)),\psi(\phi(s_1))} \end{array}$$

- os morfismos identidade $id_\Sigma : \Sigma_1 \to \Sigma_1$ são pares de mapeamentos identidades $id_\Sigma = \langle id^S, id^F \rangle$. Assim, sejam

$$id_\Sigma : \Sigma_1 \to \Sigma_1 = \langle id^S, id^F \rangle \text{ e } \phi : \Sigma_1 \to \Sigma_2 = \langle \phi^S, \phi^F \rangle$$

morfismos em Sign, onde

$$id^S : S_1 \to S_1 \text{ e } id^F : F_1 \to F_1$$

que é definida como:

$$id^F = \langle id^F_{w,s} : F_{1,w,s} \to F_{1,id^*(w),id(s)} \rangle$$

para todo $w \in S_1^*$ e $s \in S_1$, onde $id^* : S_1^* \to S_1^*$. Igualmente para o morfismo $\phi$, temos

$$\phi^S : S_1 \to S_2 \text{ e } \phi^F : F_1 \to F_2$$

que é definida como:

$$\phi^F = \langle \phi^F_{w,s} : F_{1,w,s} \to F_{2,\phi^*(w),\phi(s)} \rangle$$

para todo $w \in S_1^*$ e $s \in S_1$, onde $\phi^* : S_1^* \to S_2^*$. Então, temos que a composição desses morfismos é definida como:

$$\phi \circ id_\Sigma : \Sigma_1 \to \Sigma_2 =^{def} \langle \phi^S \circ id^S, \phi^F \circ id^F \rangle$$

i.e., para cada componente dos morfismos. Logo, a definição da composição para a componente $S$ é

$$(\phi^S \circ id^S)_s = \phi^S (id^S)_s = \phi_s$$

para todo $s \in S_1$. Finalmente, para a componente $F$ temos:

$$(\phi^F \circ id^F)_{w,s} =^{def} (\phi_{id^*(w),id(s)} \circ id_{w,s}) = \phi_{w,s}.$$

Novamente, temos os diagramas representativos, tal que para $id_\Sigma$, temos:

$$\begin{array}{ccc} S_1 & S_1^* \times S_1 & F_{1,w_1,s_1} \\ {\scriptstyle id}\downarrow & \downarrow{\scriptstyle id^* \times id} & \downarrow{\scriptstyle id_{w_1,s_1}} \\ S_1 & S_1^* \times S_1 & F_{1,id^*(w_1),id(s_1)} \end{array}$$

O diagrama de $\phi$ pode ser observado no ítem acima. Então, a composição é

$$\begin{array}{ccc} S_1 & S_1^* \times S_1 & F_{1,w_1,s_1} \\ {\scriptstyle (\phi \circ id)}\downarrow & \downarrow{\scriptstyle (\phi \circ id)^* \times (\phi \circ id)} & \downarrow{\scriptstyle \phi_{id^*(w_1),id_{s_1}} \circ id_{w_1,s_1}} \\ S_2 & S_2^* \times S_2 & F_{2,\phi^*(id^*(w_1)),\phi(id(s_1))} \end{array}$$

que pode ser simplificada como:

$$\begin{array}{ccc} S_1 & S_1^* \times S_1 & F_{1,w_1,s_1} \\ {\scriptstyle (\phi \circ id)}\downarrow & \downarrow{\scriptstyle (\phi \circ id)^* \times (\phi \circ id)} & \downarrow{\scriptstyle \phi_{w_1,s_1}} \\ S_2 & S_2^* \times S_2 & F_{2,\phi^*(w_1),\phi(s_1)} \end{array}$$

Seguindo a definição de instituição, apresentamos agora, o segundo ingrediente da lógica *MSEqtl*, i.e., funtor modelo ($Mod_{MSEqtl}$).

### 4.1.2 Funtor Modelo $Mod_{MSEqtl} : \mathsf{Sign}^{op}_{MSEqtl} \to \mathsf{Cat}$

O funtor $Mod_{MSEqtl}$ (daqui em diante, representado simplificadamente $Mod$) atribui para cada assinatura $\Sigma \in |\ \mathsf{Sign}_{MSEqtl}\ |$ uma categoria de $\Sigma$-álgebras como objetos e homomorfismos entre $\Sigma$-álgebras como morfismos. No caso da lógica *MSEqtl* os modelos são álgebras então, escrevemos $\Sigma$-álgebras no lugar de $\Sigma$-modelos .

Uma $\Sigma$-álgebra interpreta cada símbolo de sort da assinatura como um conjunto e cada símbolo de operação como uma função.

**Definição 4.4 ($\Sigma$-álgebra)** Seja $\Sigma = \langle S, F\rangle$ uma assinatura. Uma $\Sigma$-álgebra $A = \langle A(S), A(F)\rangle$ é dada por:

- $A(S) = \langle A_s \mid s \in S\rangle$ que é uma família de conjuntos indexada por $S$, os quais são chamados "carregadores".
- $A(F) = \langle A(F)_{w,s} \mid w \in S^*, s \in S\rangle$ é uma família de funções indexada por $S^* \times S$, com uma função

$$A_{f,w,s} : A_w \to A_s$$

para cada símbolo de operação $f : w \to s \in F$, onde $A(s_1 \ldots s_n) = A_{s_1} \times \ldots \times A_{s_n}$ no caso de $w = s_1 \ldots s_n$

Como visto na definição acima, $\Sigma$-álgebras em geral têm vários conjuntos carregadores. Então uma família de mapeamentos entre esses conjuntos é chamada homomorfismo entre $\Sigma$-álgebras, se os mapeamentos são compatíveis com todas as constantes e funções da álgebra. Assim, essa idéia é capturada na seguinte

**Definição 4.5 ($\Sigma$- Homomorfismo)** Dadas duas $\Sigma$-álgebras $A$ e $B$, um $\Sigma$- homomorfismo $h : A \to B$ é uma família de mapeamentos indexada por $S$.

$$h = \langle h_s : A_s \to B_s \mid s \in S\rangle,$$

tal que $h$ é compatível com as operações, isto é,

$$h_s(A_f(a_1, \ldots, a_n)) = B_f(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$$

$$\begin{array}{ccc} A_w & \xrightarrow{h_w} & B_w \\ {\scriptstyle A_{f,w,s}}\downarrow & & \downarrow{\scriptstyle B_{f,w,s}} \\ A_s & \xrightarrow{h_s} & B_s \end{array}$$

para cada símbolo de operação $n$-ário $f : s_1 \ldots s_n \to s \in F$ e todo $a_1 \in A_{s_1}, \ldots, a_n \in A_{s_n}$.

**Nota 4.2** Denotamos $(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$ como $h_w(a)$ com $a \in A_w$. Assim, podemos escrever, simplificadamente, a condição de *homorfismo*: $h_s \circ A_f = B_f \circ h_w$

A definição do funtor $Mod$ em objetos atribui para cada assinatura $\Sigma \in |\ \mathsf{Sign}\ |$ uma categoria de $\Sigma$-álgebras.

***Proposição 4.1 (Categoria** $Mod_{MSEqtl}$**)*** *Dada um assinatura* $\Sigma$*. A categoria* $Mod(\Sigma)_{MSEqtl}$ *consiste de:* $\Sigma$*-álgebras como objetos e* $\Sigma$*-homomorfismos entre as álgebras como morfismos.*

*Prova.* Temos que provar que os homorfismos entre as álgebras preservam identidade e composição. Primeiramente, o homomorfismo identidade (mapeamento identidade nos carregadores) é realmente a *identidade*, pois a definição é baseada em mapeamentos. Segundamente, a composição de homomorfismos deve ser um homomorfismo. Sejam $h = (h_s : A_s \to B_s \mid s \in S)$ e $g = (g_s : B_s \to C_s \mid s \in S)$ dois homomorfismos, definimos a composição desses $\Sigma$-homomorfismos como a composição de mapeamentos em Set, isto é:

$$\langle (g \circ h)_s \mid s \in S \rangle = (g_s \circ h_s : A_s \to C_s \mid s \in S)$$

Assim, temos que provar que a composição é compatível com as operações. Então, sejam os dois homomorfismos $h$ e $g$ definidos acima, temos que o seguinte diagrama comuta:

$$\begin{array}{ccccc} A_w & \xrightarrow{h_w} & B_w & \xrightarrow{g_w} & C_w \\ {\scriptstyle A_{f,w,s}}\downarrow & & \downarrow{\scriptstyle B_{f,w,s}} & & \downarrow{\scriptstyle C_{f,w,s}} \\ A_s & \xrightarrow[h_s]{} & B_s & \xrightarrow[g_s]{} & C_s \end{array}$$

ou seja,

$$\begin{array}{lll} (g \circ h)_s \circ A_f & = (g_s \circ h_s) \circ A_f & \text{(pela definição de composição)} \\ & = g_s \circ B_f \circ h_w & (h \text{ é um homomorfismo}) \\ & = C_f \circ g_w \circ h_w & (g \text{ é um homomorfismo}) \\ & = C_f \circ (g \circ h)_w & \text{(pela definição de composição)} \end{array}$$

para todo $s \in S$ e $f : w \to s \in F$ e $w = s_1 \ldots s_n$

Agora, pela definição do funtor *Mod* em setas, temos para cada morfismo entre assinaturas $\phi$, um funtor $Mod(\phi)$, entre as categorias das $\Sigma$-álgebras relacionadas por $\phi$. Contudo, se o morfismo entre assinaturas é de $\Sigma$ para $\Sigma'$, então o funtor $Mod(\phi)$ é da categoria de $\Sigma'$-álgebras para a de $\Sigma$-álgebras. Nos exemplos apresentados no início do capítulo podemos observar a *contravariância* desse funtor. Chamamos $Mod(\phi)$ de *funtor esquecimento*, que é mostrado em detalhes, na seqüência. Neste trabalho, utilizamos o nome *funtor esquecimento* que é uma herança dos estudos em *Álgebra Universal*, entretanto a terminologia *funtor restrição* também é adequada.

**Definição 4.6 (Funtor Esquecimento (*forgetful*)** $Mod(\phi)_{MSEqtl}$**)** Dado um morfismo entre assinaturas $\phi : \Sigma_1 \to \Sigma_2$ em Sign, definimos um **funtor esquecimento** $Mod(\phi) : Mod(\Sigma_2) \to Mod(\Sigma_1)$ em $\Sigma_2$-álgebras e $\Sigma_2$-homomorfismos.

1. Para toda $\Sigma_2$-álgebra $A_2$ temos um $\Sigma_1$-álgebra $Mod(\phi)(A_2)$, onde

$$\begin{array}{ll} Mod(\phi)(A_2)_{s_1} & =^{Def} A_{2,\phi(s_1)} \text{ para todo } s_1 \in S_1 \text{ e} \\ Mod(\phi)(A_2)_{f_1} & =^{Def} A_{2,\phi(f_1)} \text{ para todo } f_1 \in F_1 \end{array}$$

2. Para todo $\Sigma_2$-homomorfismo $h_2 : A_2 \to B_2$ obtemos um $\Sigma_1$-homomorfismo

$Mod(\phi)(h_2) : Mod(\phi)(A_2) \to Mod(\phi)(B_2)$ definido por
$Mod(\phi)(h_2)_{s_1} =^{Def} h_{2,\phi(s_1)}$ para todo $s_1 \in S_1$

O diagrama seguinte representa a definição do funtor esquecimento em objetos e setas.

$$(A_2)_{s_1} \xleftarrow{Mod(\phi)} A_{2,\phi(s_1)}$$

$$(A_2)_{f_1} \xleftarrow{Mod(\phi)} A_{2,\phi(f_1)}$$

$$(h_2)_{s_1} \xleftarrow{Mod(\phi)} h_{2,\phi(s_1)}$$

**Nota 4.3** Precisamos verificar se o mapeamento preserva as identidades e é compatível com a composição, ou seja, é um funtor.

1. $Mod(\phi)$ preserva as identidades. Seja $A_2$ uma $\Sigma_2$-álgebra que tem como identidades $id_{A_2} : A_2 \to A_2$. Para todo $s_1 \in S_1$, temos:

$$\begin{array}{lll} Mod(\phi)(id_{A_2})_{s_1} & = id_{A_2,\phi(s_1)} & \text{(pela definição 4.6,1)} \\ & = id_{Mod(\phi)(A_2)_{s1}} & \text{(pela definição 4.6,1 )} \end{array}$$

2. $Mod(\phi)$ é compatível com a composição de homomorfismos. Dados dois $\Sigma_2$-homorfismos $h_2 : A_2 \to B_2$ e $g_2 : B_2 \to C_2$ para todo $s_1 \in S_1$, temos:

$$\begin{array}{lll} Mod(\phi)(g_2 \circ h_2)_{s_1} & = (g_2 \circ h_2)_{\phi(s_1)} & \text{(pela def. 4.6,2)} \\ & = g_{2,\phi(s_1)} \circ h_{2,\phi(s_1)} & \text{(comp. em Set)} \\ & = Mod(\phi)(g_2)_{s_1} \circ Mod(\phi)(h_2)_{s_1} & \text{(pela def. 4.6,2)} \end{array}$$

Finalmente, o funtor $Mod$, que consiste essencialmente nos componentes semânticos da instituição *MSEqtl* é formalizado abaixo.

***Proposição 4.2 (Funtor** $Mod_{MSEqtl}$**)*** *Podemos definir $Mod_{MSEqtl}$ como um funtor $Mod_{MSEqtl} : \mathsf{Sign}^{op}_{MSEqtl} \to \mathsf{Cat}$, tal que,*

- *Para toda assinatura $\Sigma \in |\ \mathsf{Sign}\ |$, $Mod(\Sigma)$ é a categoria de todas as $\Sigma$-álgebras segundo a proposição 4.1 .*
- *Para todo morfismo entre assinaturas $\phi : \Sigma_1 \to \Sigma_2$ em $\mathsf{Sign}$, $Mod(\phi) : Mod(\Sigma_2) \to Mod(\Sigma_1)$ é o funtor "esquecimento" como definido em 4.6.*

*Prova.* Precisamos provar que o funtor $Mod_{MSEqtl}$ preserva o morfismo identidade e a composição.

1. Para verificar se $Mod_{MSEqtl}$ é compatível com a composição, sendo $\phi : \Sigma_1 \to \Sigma_2$ e $\psi : \Sigma_2 \to \Sigma_3$ dois morfismos em $\mathsf{Sign}$, temos que provar que os seguintes morfismos em $\mathsf{Cat}$ são iguais:

$$
\begin{array}{ccc}
Mod(\Sigma_3) & & Mod(\Sigma_3) \\
\Big\downarrow{\scriptstyle Mod(\psi\circ\phi)} & & \Big\downarrow{\scriptstyle Mod(\psi)} \\
 & & Mod(\Sigma_2) \\
 & & \Big\downarrow{\scriptstyle Mod(\phi)} \\
Mod(\Sigma_1) & & Mod(\Sigma_1)
\end{array}
$$

Para todo $s \in S$ e para toda $\Sigma_3$-álgebra $A_3$, temos que: $Mod(\psi \circ \phi)(A_3)_s =^{Def} A_{3,\psi(\phi(s))}$, então:

$$
\begin{array}{lll}
Mod(\psi \circ \phi)(A_3)_s & = A_{3,\psi(\phi(s))} & \text{(pela definição acima)} \\
 & = (Mod(\psi)(A_3))_{\phi(s)} & \text{(pela def. 4.6)} \\
 & = Mod(\phi)(Mod(\psi)(A_3))_s & \text{(pela def. 4.6)} \\
 & = (Mod(\phi) \circ Mod(\psi))(A_3)_s & \text{(pela comp. em Set)}
\end{array}
$$

2. Da mesma maneira, para verificar se $Mod_{MSEqtl}$ preserva as identidades, sendo $\phi : \Sigma_1 \to \Sigma_2$ e $id_1 : \Sigma_1 \to \Sigma_1$ morfismos em Sign, temos que provar que os seguintes morfismos em Cat são iguais:

$$
\begin{array}{ccc}
Mod(\Sigma_2) & & Mod(\Sigma_2) \\
\Big\downarrow{\scriptstyle Mod(\phi)} & & \Big\downarrow{\scriptstyle Mod(\phi)} \\
 & & Mod(\Sigma_1) \\
 & & \Big\downarrow{\scriptstyle Mod(id_1)} \\
Mod(\Sigma_1) & & Mod(\Sigma_1)
\end{array}
$$

Para todo $\Sigma_2$-modelo $A_2$ e $s \in S$, pela definição de composição acima, temos que $(Mod(id_1) \circ Mod(\phi))(A_2)_s = A_{2,\phi(id_1(s))} = A_{2,\phi(s)}$, então:

$$
\begin{array}{lll}
(Mod(id_1) \circ Mod(\phi))(A_2)_s & = A_{2,\phi(id_1(s))} & \text{(def. de comp. de } Mod) \\
 & = A_{2,\phi(s)} & \text{(def. de identidade)} \\
 & = Mod(\phi)(A_2)_s & \text{(def. 4.6)}
\end{array}
$$

O terceiro ingrediente da instituição é o funtor $Sen$, que atribui para cada assinatura em Sign um conjunto de sentenças.

### 4.1.3 O Funtor Sentença $Sen_{MSEqtl} : \mathsf{Sign}_{MSEqtl} \to \mathsf{Set}$

Nesta seção, apresentamos o funtor $Sen_{MSEqtl}$, que define termos e sentenças sobre uma assinatura, onde as sentenças são equações. Este funtor define morfismos (traduções) entre essas equações (em Set), que são induzidos pelos morfismos entre assinaturas (em Sign).

Um sistema de variáveis é uma família de conjuntos de variáveis indexada por sorts, com a restrição de que os nomes das variáveis não se repetem. Por exemplo,

seja $[x : nat, y : list, l : list]head(cons(x :: l)) = x : nat$ uma equação com seu sistema de variáveis. Assim, não pode ocorrer uma situação tal que $[x : nat, y : nat, y : list, l : list]head(cons(x :: l)) = x : nat$, pois o nome de variável $y$ aparece repetido, representando uma mesma variável com dois sorts.

**Definição 4.7 ($\Sigma$-Sistema de Variáveis)** Sejam $\Sigma = \langle S, F \rangle$ uma assinatura. Um **$\Sigma$-sistema de variáveis** $X$ é uma família de conjuntos indexada por $S$, i.e., $\langle X_s \mid s \in S \rangle$, com

$$s_1 \neq s_2 \Longrightarrow X_{s_1} \cap X_{s_2} = \varnothing \text{ para todo } s_1 \text{ e } s_2 \in S.$$

A Figura 4.1 pode ser vista como uma denotação de um sistema de variáveis, dado um conjunto $s$ de sorts.

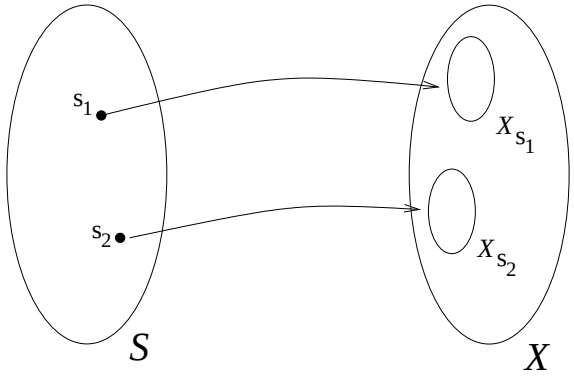



FIGURA 4.1 – Sistema de Variáveis

Os termos de uma assinatura são definidos para cada sort $s$ do conjunto de sorts.

**Definição 4.8 (Termos de uma Assinatura)** Sejam $\Sigma = \langle S, F \rangle$ uma assinatura e $X$ um $\Sigma$-sistema de variáveis.

$$T(\Sigma, X) = \langle T(\Sigma, X)_s \mid s \in S \rangle$$

é chamada família de **$\Sigma$-termos sobre** $X$ e é definida para cada $s \in S$ como conjunto $T(\Sigma, X)_s$, para o qual vale:

- $X_s \subseteq T(\Sigma, X)_s$,
- $f \in T(\Sigma, X)_s$ para todo $f : \lambda \to s \in F$,
- $f(t_1, \ldots, t_n) \in T(\Sigma, X)_s$ para todo $f : s_1 \ldots s_n \to s \in F$ e todo $t_1 \in T(\Sigma, X)_{s_1}$, $\ldots t_n \in T(\Sigma, X)_{s_n}$.

As sentenças da lógica *MSEqtl* são equações. A idéia de uma equação é sempre carregar o seu contexto, i.e., sistema de variáveis. Por exemplo, considere as seguintes sentenças $[x : int]\ plus(x, 0) = x : int$ e $[x, y : int, l : list]\ plus(x, 0) = x : int$. Elas são vistas como equações diferentes, já que seus sistemas de variáveis não são os mesmos. Neste sentido, podemos pensar que cada equação tem uma *tag*, i.e., seu contexto (*declaração de variáveis é local*). Desta forma, o conjunto de equações de uma assinatura são todas as equações com seus possíveis sistemas de variáveis. Essa idéia é colocada em detalhes na seguinte

**Definição 4.9 ($\Sigma$-Equações)** Sejam $\Sigma = \langle S, F\rangle$ uma assinatura e $X$ um $\Sigma$-sistema de variáveis. Uma $\Sigma$-equação é escrita $(X \vdash t = t')$, onde $t$ e $t' \in T(\Sigma, X)_s,\ s \in S$. Sendo que $\mathsf{Eq}(\Sigma, X)$ denota o conjunto de todas as $\Sigma$-equações sobre $X$ e $\mathsf{Eq}(\Sigma)$ é o conjunto de todas as equações para uma assinatura, isto é:

$$\mathsf{Eq}(\Sigma) =^{Def} \bigcup\{\mathsf{Eq}(\Sigma, X) \mid\ X \text{ é um } \Sigma\text{-sistema de variáveis.}\}$$

O funtor *Sen* é definido em setas atribuindo para cada morfismo entre assinaturas ($\phi$) um mapeamento entre os conjuntos de sentenças dessas assinaturas. Assim, para cada termo da assinatura origem de $\phi$ temos um respectivo na assinatura destino. A tradução destes termos segue a sua estrutura, i.e., é definida para variáveis, para constantes e para termos compostos.

**Definição 4.10 (Tradução de Termos)** Dadas duas assinaturas $\Sigma_1 = \langle S_1, F_1\rangle$, $\Sigma_2 = \langle S_2, F_2\rangle$, um morfismo entre assinaturas $\phi : \Sigma_1 \to \Sigma_2$ e $X_1$ um $\Sigma_1$-sistema de variáveis. Definimos $X_2$ como $\Sigma_2$-sistema de variáveis por

$$X_{2,s_2} =^{Def} \bigcup\{X_{1,s_1} \mid \phi(s_1) = s_2\}$$

para todo $s_1 \in S_1$. Ou seja,

$$X_{1,s_1} \xmapsto{\overline{\phi}} X_{1,\phi(s_1)}$$

Assim, a tradução $\overline{\phi} : T(\Sigma_1, X_1) \to T(\Sigma_2, X_2)$ de $\Sigma_1$-termos é dada por uma família de mapeamentos $\overline{\phi}_{s_1} : T(\Sigma_1, X_1)_{s_1} \to T(\Sigma_2, X_2)_{\phi(s_1)}$ indexada por $S_1$, com

- $\overline{\phi}_{s_1}(x_1) =^{Def} x_1 (\in X_{2,\phi(s1)})$ para todo $x_1 \in X_{1,s_1}$
- $\overline{\phi}_{s_1}(f_1) =^{Def} \phi(f_1)$ para todo $f_1 : \lambda \to s_1 \in F_1$
- $\overline{\phi}_{s_1}(f_1(t_1, \ldots, t_n)) =^{Def} \phi(f_1)(\overline{\phi}_{s_1}(t_1), \ldots, \overline{\phi}_{s_n}(t_n))$ para todo $f_1 : w_1 \to s_1 \in F_1$ e todo $t_1 \in T(\Sigma_1, X_1)_{s_1}, \ldots t_n \in T(\Sigma_1, X_1)_{s_n}$.

para todo $s_1 \in S_1$ e para todo $w_1 = s_1 \ldots s_n \in S_1^*$. A tradução de termos pode ser verificada nos seguintes diagramas.

$$x_1 \xmapsto{\overline{\phi}_{s_1}} x_1 \in X_{2,\phi(s_1)}$$

$$f_1 \xmapsto{\overline{\phi}_{s_1}} \phi(f_1) \in F_{2,\phi(s_1)}$$

$$f_1(t_1, \ldots, t_n) \xmapsto{\overline{\phi}_{s_1}} \phi(f_1)(\overline{\phi}_{s_1}(t_1), \ldots, \overline{\phi}_{s_n}(t_n)) \in F_{2,\phi^*(w_1),\phi(s_1)}$$

Em relação à tradução de variáveis, considere o seguinte exemplo do início da seção.

$$\overline{\phi}([x : nat, s : string], first(cons(x\ w)) = x : nat) =$$
$$= ([x : int, l : list], head(conc(x\ l)) = x : int)$$
$$\overline{\phi}([x : nat, s : string], rest(cons(x\ w)) = w : string) =$$
$$= ([x : int, l : list], tail(conc(x\ l)) = l : list)$$

Temos uma tradução de sentenças entre assinaturas *MSEqtl*. Nos sistemas de variáveis que acompanham as equações, a única mudança na tradução de variáveis é respectiva ao sort. Os *nomes* das variáveis permanecem os *mesmos*.

O funtor $Sen_{MSEqtl} : \mathsf{Sign}_{MSEqtl} \to \mathsf{Set}$ leva cada assinatura ao conjunto de todas as equações sobre esta assinatura e cada morfismo entre assinaturas para um morfismo que traduz equações das respectivas assinaturas.

**Definição 4.11 (Funtor Sentença $Sen_{MSEqtl}$)** O **funtor** $Sen_{MSEqtl} : \mathsf{Sign} \to \mathsf{Set}$ (daqui em diante simplesmente $Sen$) é definido:

1. Sobre assinaturas: $Sen(\Sigma) =^{def} \mathsf{Eq}(\Sigma)$
2. Sobre morfismos entre assinaturas: $Sen(\phi : \Sigma_1 \to \Sigma_2) = Sen(\phi) : \mathsf{Eq}(\Sigma_1) \to \mathsf{Eq}(\Sigma_2)$, onde $Sen(\phi)(X \vdash t = t') =^{Def} (\overline{\phi}(X_1) \vdash \overline{\phi}(t) = \overline{\phi}(t'))$, para $t, t' \in T(\Sigma_1, X_1)$

**Nota 4.4** Precisamos verificar que $Sen$ realmente é um funtor, isto é, preserva identidades e composição.

Para verificar se $Sen$ é compatível com a composição, sendo $\phi : \Sigma_1 \to \Sigma_2$ e $\psi : \Sigma_2 \to \Sigma_3$ dois morfismos em $\mathsf{Sign}$, temos que provar que os seguintes morfismos em $\mathsf{Set}$ são iguais:

$$\begin{array}{ccc} \mathsf{Eq}(\Sigma_1) & & \mathsf{Eq}(\Sigma_1) \\ \Big\downarrow{\scriptstyle Sen(\psi\circ\phi)} & & \Big\downarrow{\scriptstyle Sen(\phi)} \\ & & \mathsf{Eq}(\Sigma_2) \\ & & \Big\downarrow{\scriptstyle Sen(\psi)} \\ \mathsf{Eq}(\Sigma_3) & & \mathsf{Eq}(\Sigma_3) \end{array}$$

Para toda $(X_1 \vdash t = t')$ $\Sigma_1$-equação, temos que: $Sen(\psi \circ \phi)(X_1 \vdash t = t') =^{def} (\overline{\psi}(\overline{\phi}(X_1)) \vdash \overline{\psi}(\overline{\phi}(t)) = \overline{\psi}(\overline{\phi}(t')))$, então;

$$\begin{array}{ll} (Sen(\psi) \circ Sen(\phi))(X_1 \vdash t = t') & = Sen(\psi)(Sen(\phi)(X_1 \vdash t = t')) \\ & (\text{comp. em } \mathsf{Set}) \\ & = Sen(\psi)(\overline{\phi}(X_1) \vdash \overline{\phi}(t) = \overline{\phi}(t')) \\ & (\text{def. 4.11,2}) \\ & = (\overline{\psi}(\overline{\phi}(X_1)) \vdash \overline{\psi}(\overline{\phi}(t)) = \overline{\psi}(\overline{\phi}(t'))) \\ & (\text{def. 4.11,2}) \\ & = Sen(\psi \circ \phi)(X_1 \vdash t = t') \\ & (\text{def. acima}) \end{array}$$

Da mesma maneira, para verificar se $Sen$ preserva as identidades, sendo $\phi : \Sigma_1 \to \Sigma_2$ e $id_1 : \Sigma_1 \to \Sigma_1$ dois morfismos em $\mathsf{Sign}$, temos que provar que os seguintes morfismos em $\mathsf{Set}$ são iguais :

$$\begin{array}{ccc} \mathsf{Eq}(\Sigma_1) & & \mathsf{Eq}(\Sigma_1) \\ \Big\downarrow{\scriptstyle Sen(\phi)} & & \Big\downarrow{\scriptstyle Sen(id_1)} \\ & & \mathsf{Eq}(\Sigma_1) \\ & & \Big\downarrow{\scriptstyle Sen(\phi)} \\ \mathsf{Eq}(\Sigma_2) & & \mathsf{Eq}(\Sigma_2) \end{array}$$

Para toda $(X_1 \vdash t = t')$ $\Sigma_1$-equação, temos que $Sen(\phi \circ \ id_1)(X_1 \vdash t = t') =^{def} (\overline{\phi}(\overline{id}_1(X_1)) \vdash \overline{\phi}(\overline{id}_1(t)) = \overline{\phi}(\overline{id}_1(t')))$ que pode ser simplificada para $(\overline{\phi}(X_1) \vdash \overline{\phi}(t) = \overline{\phi}(t'))$, então:

$$\begin{array}{ll} (Sen(\phi) \circ \ Sen(id_1))(X_1 \vdash t = t') & = (\overline{\phi}(\overline{id}_1(X_1)) \vdash \overline{\phi}(\overline{id}_1(t)) = \overline{\phi}(\overline{id}_1(t'))) \\ & \text{(composição)} \\ & = (\overline{\phi}(X_1) \vdash \overline{\phi}(t) = \overline{\phi}(t')) \\ & \text{(definição acima)} \\ & = Sen(\phi)(X_1 \vdash t = t') \\ & \text{(definição 4.11)} \end{array}$$

### 4.1.4 Condição de Satisfação

Nesta seção apresentamos a relação de satisfação e as definições importantes para provarmos que os componentes sintáticos são compatíveis com os componentes semânticos.

Uma atribuição de variáveis é uma família de funções indexada por sorts, tal que para cada variável de sort $s$ é dado um valor na álgebra no respectivo conjunto base deste sort.

**Definição 4.12 (Atribuição de Variáveis)** Sejam $A$ uma $\Sigma$-álgebra e $X$ um $\Sigma$-sistema de variáveis. Uma atribuição $\alpha : X \to A$ é uma família de mapeamentos indexada por $S$

$$\alpha = (\alpha_s : X_s \to A_s \mid s \in S)$$

Na Figura 4.2 temos a família de atribuições (indexada por S).

A avaliação de termos é também uma família de mapeamentos indexada por sort, que interpreta cada termo da assinatura $\Sigma$ em uma $\Sigma$-álgebra. A definição de avaliação é dada de acordo com a estrutura de termos, i.e., para variáveis (ou seja, a própria atribuição), para constantes e para termos complexos.

**Definição 4.13 (Avaliação de $\Sigma$-Termos)** Para uma $\Sigma$-álgebra $A$, para um conjunto de termos $T(\Sigma, X)$ e para uma atribuição $\alpha : X \to A$, a avaliação de termos $\overline{\alpha} : T(\Sigma, X) \to A$ é uma família de mapeamentos indexada por $S$

$$\overline{\alpha} = \langle \overline{\alpha}_s : T(\Sigma, X)_s \to A_s \mid \ s \in S \rangle$$

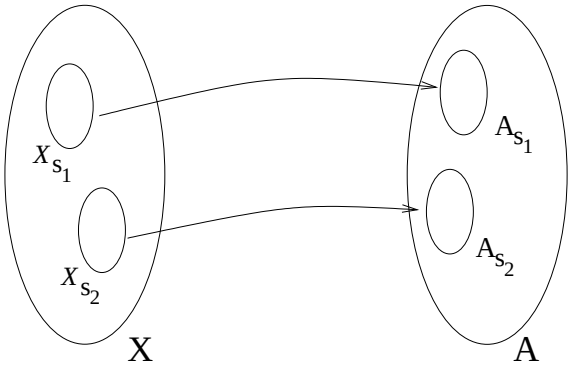



FIGURA 4.2 – Atribuição

definida para todo $s \in S$ como segue:

- $\overline{\alpha}_s(x) =^{Def} \alpha_s(x)$ para todo $x \in X_s$,
- $\overline{\alpha}_s(f) =^{Def} A_f$ para todo $f : \lambda \to s \in F$,
- $\overline{\alpha}_s(f(t_1, \ldots, t_n)) =^{Def} A_f(\overline{\alpha}_w(t_1, \ldots, t_n))$, onde para todo $f : w \to s \in F$ e todo $t_1 \in T(\Sigma, X)_{s_1}, \ldots, t_n \in T(\Sigma, X)_{s_n}$ sendo $w = s_1 \ldots s_n$.

Podemos analisar a avaliação de termos no diagrama abaixo.

$$x \xmapsto{\overline{\alpha}_s} \alpha_s(x)$$

$$f \xmapsto{\overline{\alpha}_s} A_f$$

$$f(t_1, \ldots, t_n) \xmapsto{\overline{\alpha}_s} A_f(\overline{\alpha}_w(t_1, \ldots, t_n))$$

Como mencionado no ínicio desta seção, dada uma assinatura $\Sigma$ e uma $\Sigma$-álgebra temos o conceito de satisfação de $\Sigma$-sentenças nesta álgebra.

**Definição 4.14 (Satisfação de uma equação em uma atribuição)** Sejam $\Sigma = \langle S, F \rangle$ uma assinatura, $\varphi = (X \vdash t = t')$ uma equação sobre $\Sigma$, uma $\Sigma$-álgebra $A = \langle A(S), A(F) \rangle$ e $\alpha : X \to A$ uma atribuição. Então $A$ satisfaz $\varphi$ em $\alpha : X \to A$, escrita $A, \alpha \models_\Sigma \varphi$, se e somente se $\overline{\alpha}(t) = \overline{\alpha}(t')$.

**Definição 4.15 (Satisfação de uma equação)** Sejam $\Sigma = \langle S, F \rangle$ uma assinatura, $\varphi = (X \vdash t = t')$ uma equação sobre $\Sigma$, uma $\Sigma$-álgebra $A = \langle A(S), A(F) \rangle$. Então $A$ satisfaz $\varphi$, escrita $A \models_\Sigma \varphi$, se e somente se $A, \alpha \models_\Sigma \varphi$, para todo $\alpha : X \to A$.

**Nota 4.5** Então, podemos dizer que uma álgebra satisfaz um conjunto de equações, se e somente se as avaliações dos lados direito e esquerdo da equação produzem o mesmo resultado em $A$, sendo $A$ uma $\Sigma$-álgebra, para toda atribuição $\alpha : X \to A$.

Considere $\phi$ um morfismo entre duas assinaturas. Para toda a atribuição de variáveis para valores da álgebra da assinatura origem temos uma semelhante na destino e vice-versa.

**Definição 4.16 (Atribuições Correspondentes)** Dado um morfismo entre assinaturas $\phi : \Sigma_1 \to \Sigma_2$, uma $\Sigma_2$-álgebra $A_2$, um $\Sigma_1$-sistema de variáveis $X_1$ e um $\Sigma_2$-sistema de variáveis $\overline{\phi}(X_1)$. Para toda atribuição $\alpha_{1,s_1} : X_{1,s_1} \to Mod(\phi)(A_2)_{s_1}$ existe uma atribuição $\alpha_{2,s_2} : \overline{\phi}(X_1)_{s_2} \to A_{2,s_2}$, definida por:

$$\alpha_{2,s_2}(x) =^{Def} \alpha_{1,s_1}(x)$$

para todo $s_2 \in S_2$, onde $\phi(s_1) = s_2$ e $x \in X_{1,s_1} \subseteq \overline{\phi}(X_1)_{s_2}$.

Da mesma maneira, para toda atribuição $\alpha_{2,s_2} : \overline{\phi}(X_1)_{s_2} \to A_{2,s_2}$ existe uma atribuição $\alpha_{1,s_1} : X_{1,s_1} \to Mod(\phi)(A_2)_{s_1}$, dada por:

$$\alpha_{1,s_1}(x) =^{Def} \alpha_{2,\phi(s_1)}(x)$$

para $s_1 \in S_1$ onde $x \in X_{1,s_1} \subseteq \overline{\phi}(X_1)_{\phi(s_1)}$. Neste caso, chamamos $(\alpha_1, \alpha_2)$ como **par de atribuições correspondentes**, que pode ser verificado no seguinte diagrama:

$$\begin{array}{ccc} Mod(\phi)(A_2)_{s_1} & = & A_{2,\phi(s_1)} \\ \uparrow{\scriptstyle \alpha_{1,s_1}} & & \uparrow{\scriptstyle \alpha_{2,\phi(s_1)}} \\ X_{1,s_1} & \subseteq & \overline{\phi}(X_1)_{\phi(s_1)} \end{array}$$

Note que a igualdade na parte superior do diagrama é garantida pela característica que $Mod(\phi)$ tem de não mudar os carregadores, e, porque $\overline{\phi}$ não muda os nomes das variáveis temos a relação $\subseteq$.

Na definição acima, observamos que temos atribuições correspondentes, assim é definida a mesma relação para avaliações. Mais especificamente, dada uma avaliação de um termo da assinatura origem (de $\phi$) temos uma correspondente na assinatura destino, para o termo traduzido.

***Proposição 4.3 (Avaliações Correspondentes)*** *Sejam $\phi : \Sigma_1 \to \Sigma_2$ um morfismo em* Sign*, uma $\Sigma_2$-álgebra $A_2$, e um $\Sigma_1$-sistema de variáveis $X_1$. Para toda atribuição $\alpha_1 : X_1 \to Mod(\phi)(A_2)$, temos a correspondente atribuição $\alpha_2 : \overline{\phi}(X_1) \to A_2$ e vice-versa, tal que:*

$$\overline{\alpha}_{1,s_1}(t) = \overline{\alpha}_{2,\phi(s_1)}(\overline{\phi}_{s_1}(t))$$

*para todo $s_1 \in S_1$, $t \in T(\Sigma_1, X_1)_{s_1}$.*

*Prova.* A prova de correspondência entre as funções de avaliação é feita por indução estrutural sobre a construção de termos. Para todo $s_1 \in S_1$ e todo $t \in T(\Sigma_1, X_1)_{s_1}$, temos:

1. Para todas as variáveis $x$, com $x \in X_{1,s_1}$:

$$\begin{array}{lll} \overline{\alpha}_{1,s_1}(x) & = \alpha_{1,s_1}(x) & (\text{def. de } \overline{\alpha}_1) \\ & = \alpha_{2,\phi(s_1)}(x) & (\text{def. de } \alpha_2) \\ & = \overline{\alpha}_{2,\phi(s_1)}(x) & (\text{def. de } \overline{\alpha}_2) \\ & = \overline{\alpha}_{2,\phi(s_1)}(\overline{\phi}_{s_1}(x)) & (\text{def. de } \overline{\phi}) \end{array}$$

2. Para constantes $f_1$, com $f_1 : \lambda \to s_1$:

$$\begin{array}{rll} \overline{\alpha}_{1,s_1}(f_1) & = Mod(\phi)(A_2)_{f_1} & \text{(def. de } \overline{\alpha}_1) \\ & = A_{2,\phi(f_1)} & \text{(def. de } Mod(\phi)) \\ & = \overline{\alpha}_{2,\phi(s_1)}(\phi(f_1)) & \text{(def. de } \overline{\alpha}_2) \\ & = \overline{\alpha}_{2,\phi(s_1)}(\overline{\phi}_{s_1}(f_1)) & \text{(def. de } \overline{\phi}) \end{array}$$

3. Para termos compostos $f_1(t_1, \ldots, t_n)$, com $f_1 : w \to s_1$ e $t_1 \in T(\Sigma_1, X_1)_{s1}, \ldots, t_n \in T(\Sigma_1, X_1)_{sn}$, onde $w = s_1 \ldots s_n$:

$$\begin{array}{ll} \overline{\alpha}_{1,s_1}(f_1(t_1, \ldots t_n)) & = Mod(\phi)(A_2)_{f_1}(\overline{\alpha}_{1,s_1}(t_1), \ldots \overline{\alpha}_{1,s_n}(t_n)) \\ & \text{(def. de } \overline{\alpha}_1) \\ & = Mod(\phi)(A_2)_{f_1}(\overline{\alpha}_{2,\phi(s_1)}(\overline{\phi}_{s_1}(t_1)), \ldots \overline{\alpha}_{2,\phi(s_n)}(\overline{\phi}_{s_n}(t_n))) \\ & \text{(indução)} \\ & = A_{2,\phi(f_1)}(\overline{\alpha}_{2,\phi(s_1)}(\overline{\phi}_{s_1}(t_1)), \ldots \overline{\alpha}_{2,\phi(s_n)}(\overline{\phi}_{s_n}(t_n))) \\ & \text{(def. de } Mod(\phi)) \\ & = \overline{\alpha}_{2,\phi(s_1)}(\phi(f_1)(\overline{\phi}_{s_1}(t_1), \ldots \overline{\phi}_{s_n}(t_n))) \\ & \text{(def. de } \overline{\alpha}_2) \\ & = \overline{\alpha}_{2,\phi(s_1)}(\overline{\phi}_{s_1}(f_1)(t_1, \ldots t_n))) \\ & \text{(def. de } \overline{\phi}) \end{array}$$

A condição de satisfação é tal que, quando mudamos de assinatura, através de um morfismo $\phi$, a satisfação de sentenças por modelos muda coerentemente. Então, os modelos da assinatura destino satisfazem as sentenças traduzidas *se e somente se* os modelos traduzidos satisfazem as sentenças originais.

***Teorema 4.1 (Condição de Satisfação)*** *Para todo morfismo entre assinaturas $\phi : \Sigma_1 \to \Sigma_2$, e toda $\Sigma_1$-equação $\varphi = (X \vdash t = t')$ e toda $\Sigma_2$-álgebra $A_2$, o seguinte vale:*

$$A_2 \models_{\Sigma_2} Sen(\phi)(\varphi) \Leftrightarrow Mod(\phi)(A_2) \models_{\Sigma_1} \varphi$$

*Prova.* Pela proposição 4.3 sabemos que a avaliação dos termos em ambas as álgebras produz o mesmo valor, então, para todo $\alpha_{2,\phi(s_1)} : \overline{\phi}(X_1)_{\phi(s_1)} \to (A_2)_{\phi(s_1)}$, $s_1 \in S_1$:

$$\begin{array}{ll} & A_2 \models_{\Sigma_2} Sen(\phi)(\varphi) \\ \Leftrightarrow & A_2 \models_{\Sigma_2} (\overline{\phi}(X) \vdash \overline{\phi}(t) = \overline{\phi}(t')) \qquad \text{(def. de } \varphi) \\ \Leftrightarrow & \overline{\alpha}_{2,\phi(s_1)}(\overline{\phi}_{s_1}(t)) = \overline{\alpha}_{2,\phi(s_1)}(\overline{\phi}_{s_1}(t')) \quad \text{(pela def. 4.15)} \\ \Leftrightarrow & \overline{\alpha}_{1,s_1}(t) = \overline{\alpha}_{1,s_1}(t') \qquad \text{(pela prop. 4.3)} \\ \Leftrightarrow & Mod(\phi)(A_2) \models_{\Sigma_1} (X \vdash t = t') \qquad \text{(pela def. 4.15)} \\ \Leftrightarrow & Mod(\phi)(A_2) \models_{\Sigma_1} \varphi \qquad \text{(def de } \varphi) \end{array}$$

## 4.2 Instituição da *lógica Equacional Ordenada por Sorts*

Primeiramente, introduzimos como motivação, uma discussão informal de especificações na lógica *OSEqtl*.

Assim, observe as seguintes teorias nesta lógica.

*ThTriv*, *OSEqtl*

Sorts
*Elem*.
Opns
$* :\rightarrow Elem$

*Th* *Lista*$[X :: TRIV]$, *OSEqtl*

Sorts
*Lista*, *NvLista*.
Subsorts
$Elem < NvLista < Lista$.
Opns
$:: : Lista, Lista \rightarrow Lista$.
$:: : NvLista, Lista \rightarrow Lista$.
$hd : NvLista \rightarrow Lista$.
$tl : NvLista \rightarrow Lista$.

$[x : Elem, l : Lista]\ hd(x :: l) = x : Lista$.
$[x : Elem, l : Lista]\ tl(x :: l) = l : Lista$.

A primeira, é uma especificação que requer um sort e um elemento deste sort (representado por$*$). A segunda é uma especificação de listas. Nesta teoria, a notação $Lista[X :: TRIV]$, pode ser entendida como uma *importação*, i.e., que a especificação *Triv* é *incluída* na *Lista*. Neste contexto, podemos pensar que entre essas duas especificações (*Triv* e *Lista*) existe uma **seta**, mais precisamente um morfismo que *inclui Triv* em *Lista*. Então, esta situação pode ser capturada por um morfismo entre assinaturas *OSEqtl* (ver figura 4.3).

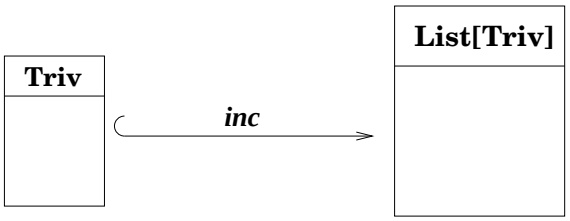



FIGURA 4.3 – Seta entre assinaturas *OSEqtl*

Outra questão que deve ser observada é que na especificação *Lista* temos a idéia de subsorts, i.e., uma ordem parcial no conjunto de sorts. Sendo que, para os modelos dessas especificações isto significa que, se temos a seguinte relação entre os sorts $s < s'$, então o conjunto carregador de $s$ é subconjunto do carregador de $s'$. Assim, em *Lista*, um elemento é também um lista.

A seguir, apresentamos a instituição da *OSEqtl*. Esta instituição pode ser pensada como uma "extensão" conservativa ou, melhor ainda, que *MSEqtl* é uma sublógica da *OSEqtl* (ver proposição 6.3). Esta idéia é discutida em detalhes no capítulo 6. Desta maneira, apresentaremos os principais conceitos que diferem da lógica equacional multi sortida. Todas as provas e demais definições podem se verificadas na seção 4.1.

### 4.2.1 Categoria $\mathsf{Sign}_{OSEqtl}$

A categoria $\mathsf{Sign}_{OSEqtl}$ tem assinaturas *OSEqtl* como objetos e morfismos entre assinaturas como setas.

Uma assinatura *OSEqtl* pode ser entendida como uma assinatura *MSEqtl* junto com uma ordem parcial no conjunto de sorts. Além disso, os símbolos de operações na *OSEqtl* são monotônicos, i.e., a ordem parcial na aridade (domínio) implica na preservação desta ordem nos sorts codomínio. Por exemplo, considere que temos um conjunto de sorts $\{nat, int\}$, tal que $nat < int$. Então, um símbolo de operação $plus \in F_{nat,nat,nat} \cap F_{int,int,int}$ e $nat, nat < int, int$ implica na restrição $nat < int$. Neste sentido, esta lógica, assim como a *MSEqtl*, permite *overloading* para símbolos de operações. Entretanto, aqui temos um *overloading* com uma noção de extensão, no sentido que o conceito de subsort transporta as funções que interpretam os símbolos de operações. Por exemplo, a função que interpreta $plus : nat, nat \to nat$ é uma restrição de função que interpreta $plus : int, int \to int$. Isso tudo, está em detalhes na seguinte

**Definição 4.17 (Assinatura *OSEqtl*)** Uma assinatura *OSEqtl* é uma tupla $\Sigma = \langle S, F, < \rangle$ onde:

- $\langle S, F \rangle$ é uma assinatura *MSEqtl*;
- $\langle S, < \rangle$ é uma ordem parcial, tal que os símbolos de operação $f \in F$ satisfazem a seguinte condição de motonicidade $f \in F_{w,s} \cap F_{w',s'}$ e $w < w' \Rightarrow s < s'$

Um morfismo entre assinaturas *OSEqtl* é similar a um morfismo entre assinaturas *MSEqtl*, tal que a ordem parcial no conjunto de sorts é preservada na tradução.

**Definição 4.18 (Morfismo entre Assinaturas *OSEqtl*)** Sejam $\Sigma_1 = \langle S_1, F_1, <_1 \rangle$ e $\Sigma_2 = \langle S_2, F_2, <_2 \rangle$ duas assinaturas *OSEqtl*. Um morfismo $\phi : \Sigma_1 \to \Sigma_2$ é definido como:

- um morfismo $\phi = \langle \phi^S, \phi^F \rangle$ como na assinatura *MSEqtl*, sendo que $\phi^S : S_1 \to S_2$ é tal que $s_1 <_{\Sigma_1} s_2 \Rightarrow \phi^S(s_1) <_{\Sigma_2} \phi^S(s_2)$.

Uma álgebra *OSEqtl* é uma álgebra *MSEqtl*, sendo que uma ordem parcial no conjunto de sorts implica na relação de subconjuntos entre os carregadores. Além disso, as funções respeitam a seguinte condição: considere o símbolo de operação $plus \in F_{nat,nat,nat} \cap F_{int,int,int}$ tal que $nat, nat < int, int$. Assim, $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ é a restrição $+ : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$.

**Definição 4.19 ($\Sigma$-Álgebra *OSEqtl*)** Dada uma assinatura *OSEqtl* $\Sigma = \langle S, F, < \rangle$, uma $\Sigma$-álgebra *OSEqtl* $A = \langle A(S), A(F), \leq \rangle$ é uma álgebra *MSEqtl* $A = \langle A(S), A(F) \rangle$, tal que a seguinte condição de motonicidade vale:

- $s \leq s'$ implica que $A_s \subseteq A_{s'}$ e
- $f \in F_{w,s} \cap F_{w',s'}$ e $w \leq w'$ implica que $A_f : A_w \to A_s$ é igual a $A_f : A_{w'} \to A_{s'}$.

De maneira análoga aos outros componentes, um homomorfismo entre $\Sigma$-álgebras *OSEqtl* é como um homomorfismo em $\Sigma$-álgebras *MSEqtl*, tal que a seguinte condição é respeitada: considere $nat < int$, a relação de ordem parcial entre os sorts vista acima, e $3 \in \mathbb{N}$, um valor no conjunto base de $nat$. Então, temos que $h_{nat}(3) = h_{int}(3)$.

**Definição 4.20 (Homomorfismo entre $\Sigma$-álgebras *OSEqtl*)** Seja $\Sigma = \langle S, F, <\rangle$ uma assinatura *OSEqtl* e $A$ e $B$ duas álgebras *OSEqtl*. Então um $\Sigma$-homomorfismo $h : A \to B$ é um $\langle S, F\rangle$-homomorfismo, como definido na instituição *MSEqtl*, satisfazendo a seguinte condição:

$$s \leq s' \text{ e } a \in A_s \text{ implica que } h_s(a) = h_{s'}(a).$$

Como já mencionado as demais definições e provas podem ser verificados na instituição *MSEqtl*.

# 5 Instituições II

Neste capítulo, temos uma apresentação detalhada da *lógica Horn sem sorts* (*Prolog* puro - conjunção e implicação). Nesta lógica, um ponto essencial a ser observado é a idéia que uma "lógica sem sorts" pode ser entendida como uma lógica com assinaturas que possuem um único sort. Desta forma este único sort pode ser pensado como uma representação do *universo* (domínio). Além disso, neste capítulo, os predicados são tratados com mais atenção, já que os outros conceitos são similares ao capítulo anterior.

## 5.1 Instituição da *Lógica Horn Sem Sorts*

Primeiramente, temos uma discussão informal da instituição para *Lógica Horn sem sorts* (daqui para frente, simplesmente lógica *Horn*).

Neste sentido, considere a seguinte especificação do tipo de dado *pilha* na lógica *Horn*.

*ThPilha, Horn*

Sort
$u$

Pred
$alfabeto, pilha : u$

Opns
$k_1, \ldots, k_n :\rightarrow u$
$vazia :\rightarrow u$
$cons : u, u \rightarrow u$
$topo : u \rightarrow u$
$pe : u \rightarrow u$

$[x, p : u]\ alfabeto(x), pilha(p) \Rightarrow pe(cons(x, p)) = p$
$[x, p : u]\ alfabeto(x), pilha(p) \Rightarrow topo(cons(x, p)) = x$
$[\ ]\ pe(vazia) = vazia$

*axiomas*
$[x, p : u]\ alfabeto(x), pilha(p) \Rightarrow pilha(cons(x, p))$
$[x, p : u]\ alfabeto(x), pilha(p) \Rightarrow pilha(pe(cons(x, p)))$
$[x, p : u]\ alfabeto(x), pilha(p) \Rightarrow alfabeto(topo(cons(x, p)))$

Note que especificamos neste esquema uma pilha com as operações *topo*, que produz como saída o elemento do topo da pilha, e *pe*, que retira o elemento do topo e dá o resto da pilha como resultado. Observe também, que a impossibilidade de declarar vários sorts nesta lógica requer, na especificação, predicados de tipos e axiomas adicionais.

Assim, uma estrutura de interpretação para esta apresentação de teoria pode ser da seguinte forma

$\{Pilha, Horn\}, Mod$

Conjuntos

$U = \mathbb{N} \uplus \mathbb{N}^* \uplus \{\bot\}$

Fun

$a_1, \ldots, a_n : \{*\} \to U$

$\lambda : \{*\} \to U$

$:: : U \times U \to U$

$top : U \to U$

$foot : U \to U$

---

$x :: w = (xw) \quad se\ x \in \mathbb{N}, w \in \mathbb{N}^*$

$\quad = \bot \quad$ caso contrário

$foot(xw) = w \quad se\ x \in \mathbb{N}, w \in \mathbb{N}^*$

$\quad = \bot \quad$ caso contrário

$foot(\lambda) = \lambda$

$top(xw) = x \quad se\ x \in \mathbb{N}, w \in \mathbb{N}^*$

$\quad = \bot \quad$ caso contrário

Observe que as funções são condicionais. Desta maneira, produzem um resultado desejado somente se aplicadas aos argumentos certos. O papel do elemento $\bot$ neste modelo é fazer com que as funções sejam totais. De maneira análoga as seções anteriores temos o conceito de satisfação de sentenças da especificação por esta estrutura de interpretação.

### 5.1.1 Categoria $\mathsf{Sign}_{Horn}$ de Assinaturas

Primeiramente, introduziremos a categoria de assinaturas *Horn*, que tem assinaturas como objetos e morfismos entre essas assinaturas como setas.

Uma assinatura *Horn* pode ser pensada como uma assinatura *MSEqtl* com um único sort junto com uma família de conjuntos de símbolos de predicados indexada por palavras de sort. Essa idéia é mostrada em detalhes na seguinte

**Definição 5.1 (Assinatura *Horn*)** Uma assinatura *Horn* $\Sigma = \langle \{u\}, F, P \rangle$ consiste de:

- um conjunto *unitário* $\{u\}$ ,
- uma família de conjuntos de nomes de operadores $F$, indexada por $\{u\}^* \times \{u\}$. Assim, $F = \langle F_{u^n,u} \mid n \in \mathbb{N} \rangle$. Um operador $f \in F_{u^n,u}$ com aridade $u^n$ e sort $u$ é escrito $f : u^n \to u$.
- uma família de conjuntos de símbolos de predicados $P$, indexada por $\{u\}^*$. Então $P = \langle P_{u^n} \mid n \in \mathbb{N} \rangle$.

tal que $F \cap P = \varnothing$

**Nota 5.1** Para representar o domínio e o codomínio dos símbolos de operações ($f \in F$) utilizaremos a seguinte notação: $f : u^n \to u$. Similarmente para representar a aridade dos símbolos de predicados $p \in P$, escreveremos $p : u^n$.

Note que na especificação *ThPilha* a notação utilizada para os sorts dos símbolos de operações é um pouco diferente, e.g., o símbolo $cons : u, u \to u$ na notação apresentada na definição é $cons : u^2 \to u$.

Uma assinatura para esta lógica normalmente poderia ser apresentada como $\langle F, P, ar_F : F \to \mathbb{N}, ar_P : P \to \mathbb{N} \rangle$. Assim, esta notação pode ser adaptada para a apresentada acima, já que temos um único símbolo para sort. Como $ar_F(u^n) = n$ ou $\mid u^n \mid = n$, $\mid u \mid = 1$ e o símbolo é sempre o mesmo, então o essencial em relação a aridade é o tamanho da palavra. Logo, $f : u^n \to u$ pode ser considerada $f : n \to 1$.

Um morfismo entre assinaturas *Horn* consiste de uma função que traduz o único sort, e de duas famílias de funções que mapeiam símbolos de operações e símbolos de predicados. Essas famílias de funções preservam o tamanho da aridade.

**Definição 5.2 (Morfismo entre Assinaturas *Horn*)** Sejam $\Sigma_1 = \langle \{u_1\}, F_1, P_1 \rangle$ e $\Sigma_2 = \langle \{u_2\}, F_2, P_2 \rangle$ duas assinaturas *Horn*. Um morfismo entre essas assinaturas $\phi : \Sigma_1 \to \Sigma_2$ é uma tripla $\langle \phi^u, \phi^F, \phi^P \rangle$ de funções, onde:

- $\phi^u : \{u_1\} \to \{u_2\}$,
- $\phi^F$ é uma família de mapeamentos de símbolos de operações indexada por $\{u_1\}^* \times \{u_1\}$, tal que $\phi^F = \langle \phi_{u^n,u} : F_{1,u^n,u} \to F_{2,\phi^*(u^n),\phi(u)} \rangle$, onde $\phi^*$ é a extensão de $\phi^u : \{u_1\} \to \{u_2\}$ para palavras de $\{u\}^*$, tal que $\phi^* : \{u_1\}^* \to \{u_2\}^*$,
- $\phi^P$ é uma família de mapeamentos de símbolos de predicados indexada por $\{u_1\}^*$, tal que $\phi^P = \langle \phi_{u^n} : P_{1,u^n} \to P_{2,\phi^*(u^n)} \rangle$, onde $\phi^*$ é a $\phi^u$ extendida definida acima.

**Nota 5.2** Para a notação ficar mais clara, escrevemos $\phi(u)$ no lugar de $\phi^u(u)$. A função $f : u^n \to u$ poderia ser escrita como $f : w \to u$ se $w = u^n \in \{u\}^*$ e $p_{u^n}$ como $p_w$ se $w = u^n \in \{u\}^*$.

Na sequência, temos em detahes a categoria de assinaturas que é o primeiro componente da instituição da lógica *Horn*.

**Definição 5.3 (Categoria $\mathsf{Sign}_{Horn}$)** A **categoria** $\mathsf{Sign}_{Horn}$ consiste de:

- assinaturas *Horn* $\langle \{u\}, F, P \rangle$ como objetos,
- morfismos entre assinaturas *Horn* como morfismos,
- a composição dos morfismos é a composição dos mapeamentos nas componentes $S$ e $F$ como definida na instituição da *MSEqtl* (ver def.4.3), adicionando a composição na componente $P$ (dos símbolos de predicados). Então, sejam dois morfismos

  $$\phi : \Sigma_1 \to \Sigma_2 = \langle \phi^u, \phi^F, \phi^P \rangle \text{ e } \psi : \Sigma_2 \to \Sigma_3 = \langle \psi^u, \psi^F, \psi^P \rangle$$

  em $\mathsf{Sign}$, onde $\phi^P : P_1 \to P_2$, definida como

  $$\phi^P = \langle \phi^P_{u^n} : P_{1,u^n} \to P_{2,\phi^*(u^n)} \rangle$$

para todo $u^n \in \{u_1\}^*$ e $\phi^* : \{u_1\}^* \to \{u_2\}^*$. Da mesma maneira para o morfismo $\psi$, temos que $\psi^P : P_2 \to P_3$, que é definido como

$$\psi^P = \langle \psi^P_{u^n} : P_{2,u^n} \to P_{3,\psi^*(u^n)} \rangle$$

para todo $u^n \in \{u_2\}^*$ e $\psi^* : \{u_2\}^* \to \{u_3\}^*$. Assim, a definição da composição para a componente $P$ é

$$(\psi \circ \phi)_{u^n} =^{Def} (\psi_{\phi^*(u^n)} \circ \phi_{u^n}).$$

Pela definição da composição nesta componente, temos que:

$$(\psi \circ \phi)_{u^n} : P_{1,u^n} \to P_{3,\psi^*(\phi^*(u^n))}.$$

Note a composição nos diagramas abaixo.

$$\begin{array}{ccc} \{u_1\} & \{u_1\}^* & P_{1,u_1^n} \\ {\scriptstyle \phi^u}\downarrow & \downarrow{\scriptstyle \phi^*} & \downarrow{\scriptstyle \phi_{u_1^n}} \\ \{u_2\} & \{u_2\}^* & P_{2,\phi^*(u_1^n)} \end{array}$$

para $\psi$

$$\begin{array}{ccc} \{u_2\} & \{u_2\}^* & P_{2,u_2^n} \\ {\scriptstyle \psi^u}\downarrow & \downarrow{\scriptstyle \psi^*} & \downarrow{\scriptstyle \psi_{u_2^n}} \\ \{u_3\} & \{u_3\}^* & P_{3,\psi^*(u_2^n)} \end{array}$$

finalmente, para composição $(\psi \circ \phi)$

$$\begin{array}{ccc} \{u_1\} & \{u_1\}^* & P_{1,u_1^n} \\ {\scriptstyle (\psi\circ\phi)}\downarrow & {\scriptstyle (\psi\circ\phi)^*}\downarrow & \downarrow{\scriptstyle \psi_{\phi^*(u_1^n)}\circ\phi_{u_1^n}} \\ \{u_3\} & \{u_3\}^* & P_{3,\psi^*(\phi^*(u_1^n))} \end{array}$$

- os morfismos identidade $id_\Sigma : \Sigma_1 \to \Sigma_1$ são triplas de mapeamentos identidades $id_\Sigma = (id^u, id^F, id^P)$. Assim, sejam

$$id_\Sigma : \Sigma_1 \to \Sigma_1 = \langle id^u, id^F, id^P \rangle \text{ e } \phi : \Sigma_1 \to \Sigma_2 = \langle \phi^u, \phi^F \phi^P \rangle$$

morfismos em $\mathsf{Sign}$, onde $\phi^P : P_1 \to P_2$, definida como:

$$\phi^P = \langle \phi^P_{u^n} : P_{1,u^n} \to P_{2,\phi^*(u^n)} \rangle$$

para todo $u^n \in \{u_1\}^*$ e $\phi^* : \{u_1\}^* \to \{u_2\}^*$. Igualmente para o morfismo identidade $id^P : P_1 \to P_1$, definida como

$$id^P = \langle id^P_{u^n} : P_{1,u^n} \to P_{1,id^*(u^n)} \rangle$$

para todo $u^n \in \{u_1\}^*$ e $id^* : \{u_1\}^* \to \{u_1\}^*$. Então, temos que a composição desses morfismos na componente $P$ é definida como

$$(\phi \circ \ id)_{u^n} =^{Def} (\phi^P_{id^*(u^n)} \circ \ id^P_{u^n}) = \phi^P_{u^n}.$$

Assim, pela definição acima, temos que

$$(\phi \circ id)_{u^n} : P_{1,u^n} \to P_{2,\phi^*(u^n)}.$$

Observe a seguir, os diagramas para identidade.

$$\begin{array}{ccc} \{u_1\} & \{u_1\}^* & P_{1,u_1^n} \\ {\scriptstyle id}\downarrow & \downarrow{\scriptstyle id^*} & \downarrow{\scriptstyle id_{u_1^n}} \\ \{u_1\} & \{u_1\}^* & P_{1,id^*(u_1^n)} \end{array}$$

O diagrama de $\phi$ pode ser observado no ítem acima. Logo, temos que a composição é

$$\begin{array}{ccc} \{u_1\} & \{u_1\}^* & P_{1,u^n} \\ {\scriptstyle (\phi \circ id)}\downarrow & \downarrow{\scriptstyle (\phi \circ id)^*} & \downarrow{\scriptstyle \phi_{id^*(u_1^n)} \circ id_{u_1^n}} \\ \{u_2\} & \{u_2\}^* & P_{2,\phi^*(id^*(u_1^n))} \end{array}$$

que pode ser simplificada como:

$$\begin{array}{ccc} \{u_1\} & \{u_1\}^* & P_{1,u^n} \\ {\scriptstyle (\phi \circ id)}\downarrow & \downarrow{\scriptstyle (\phi \circ id)^*} & \downarrow{\scriptstyle \phi_{u^n}} \\ \{u_2\} & \{u_2\}^* & P_{2,\phi^*(u_1^n)} \end{array}$$

O segundo ingrediente da instituição *Horn* é o funtor modelo ($Mod$).

### 5.1.2 Funtor Modelo $Mod_{Horn} : \mathsf{Sign}^{op}_{Horn} \to \mathsf{Cat}$

O Funtor modelo define uma categoria de modelos para cada assinatura *Horn* e homomorfismos entre os modelos *Horn* para cada seta entre assinaturas em Sign.

**Definição 5.4 (Modelo *Horn*)** Seja uma assinatura $\Sigma = \langle \{u\}, F, P \rangle \in |\ \mathsf{Sign}_{Horn}\ |$, definimos um $\Sigma$-modelo $A = \langle A(u), A(F), A(P) \rangle$ como segue:

- $A(u) = A_u$ é um conjunto, chamado **carregador** de $u$,
- $A(F) = \langle A(F)_{u^n,u} \mid n \in \mathbb{N} \rangle$ é uma família de funções indexada por $\{u\}^* \times \{u\}$, com

$$A_{f,u^n,u} : A_{u^n} \to A_u$$

- $A(P) = \langle A(P)_{u^n} \mid n \in \mathbb{N} \rangle$ é uma família de relações indexada por $\{u\}^*$ (subconjuntos do apropriado produto cartesiano $A_{u^n}$), com

$$A_{p,u^n} \subseteq A_{u^n}$$

**Nota 5.3 (Modelo *Horn*)** $A_{u^n}$ é a abreviação para o produto cartesiano $A_u \times \ldots \times A_u$ (n vezes). Como $u$ é um sort único, $A_{u^n}$ poderia ser escrita $A_u^n = A_u \times \ldots \times A_u$.

Uma função entre os conjuntos carregadores de duas $\Sigma$-estruturas de interpretação é chamado homomorfismo entre $\Sigma$-estruturas de interpretação, se a função é compatível com todas as constantes, funções e relações da estrutura. Essa idéia é capturada na seguinte

**Definição 5.5 (Homomorfismo entre Modelos *Horn*)** Sejam uma assinatura *Horn* $\Sigma \in |\, \mathsf{Sign}_{Horm} \,|$ e dois $\Sigma$-modelos *Horn* $A, B$. Uma função

$$h = \langle h_u : A_u \to B_u \rangle$$

é chamado **homomorfismo** $h : A \to B$, se para qualquer $f : u^n \to u \in F, p : u^n \in P$ as seguintes condições de compatibilidade valem:

1. $h_u \circ A_f = B_f \circ h_{u^n}$

$$\begin{array}{ccc} A_{u^n} & \xrightarrow{h_{u^n}} & B_{u^n} \\ {\scriptstyle A_{f,u^n,u}}\downarrow & & \downarrow{\scriptstyle B_{f,u^n,u}} \\ A_u & \xrightarrow[h_{u^n}]{} & B_u \end{array}$$

2. $h_{u^n} \circ A_p \subseteq B_p \ (\subseteq B_{u^n})$

$h_{u^n}$ denota a extensão da função em conjuntos para a função no produto cartesiano de conjuntos, isto é, a aplicação paralela.

O funtor $Mod : \mathsf{Sign}_{OSEqtl} \to \mathsf{Cat}$ é definido em objetos, atribuindo para cada assinatura $\Sigma$ *Horn* sua categoria de modelos.

***Proposição 5.1 (Categoria** $Mod(\Sigma)_{Horn}$**)*** *Seja $\Sigma \in |\, \mathsf{Sign}_{Horn} \,|$ uma assinatura então podemos definir a **categoria de $\Sigma$-modelos Horn** $Mod(\Sigma)_{Horn}$, que possui a classe de todos os $\Sigma$-modelos Horn como objetos e homomorfismos entre esses modelos como setas.*

*Prova.* Primeiramente, o homomorfismo identidade no conjunto carregador é realmente o homomorfismo identidade, pois esta prova vem de Set. Segundamente, a composição de homomorfismos deve ser um homomorfismo: Dados três $\Sigma$-modelos *Horn* $A, B, C$, e dois homomorfismos $h : A \to B$, $g : B \to C$, definimos $g \circ h : A \to C$ como composição de mapeamentos em Set. Assim, para funções, temos que o diagrana abaixo comuta:

$$\begin{array}{ccccc} A_{u^n} & \xrightarrow{h_{u^n}} & B_{u^n} & \xrightarrow{g_{u^n}} & C_{u^n} \\ \downarrow{\scriptstyle A_{f,u^n,u}} & & \downarrow{\scriptstyle B_{f,u^n,u}} & & \downarrow{\scriptstyle C_{f,u^n,u}} \\ A_u & \xrightarrow[h_u]{} & B_u & \xrightarrow[g_u]{} & C_u \end{array}$$

i.e,

$$\begin{array}{lll} (i) \quad (g \circ h)_u \circ A_f & = g_u \circ (h_u \circ A_f) & \text{(pela definição de composição)} \\ & = g_u \circ (B_u \circ h_{u^n}) & (h \text{ é um homomorfismo}) \\ & = C_f \circ g_{u^n} \circ h_{u^n} & (g \text{ é um homomorfismo}) \\ & = C_f \circ (g \circ h)_{u^n} & \text{(pela definição de composição)} \\ \\ (ii) \quad (g \circ h)_{u^n} \circ A_p & \subseteq g_{u^n} \circ (h_{u^n} \circ A_p) & (g \text{ é um homomorfismo}) \\ & \subseteq g_{u^n} \circ B_p & (h \text{ é um homomorfismo}) \\ & \subseteq C_p & (g \text{ é um homomorfismo}) \end{array}$$

para qualquer $f : u^n \to u \in F$ e $p : u^n \in P$

A definição do funtor *Mod* em setas atribui para cada morfismo entre assinaturas em Sign um funtor entre categorias de modelos das assinaturas.

**Definição 5.6 (Funtor Esquecimento $Mod(\phi)_{Horn}$)** Seja $\phi : \Sigma_1 \to \Sigma_2$ um morfismo entre assinaturas *Horn* em Sign, definimos o **funtor esquecimento** $Mod(\phi)_{Horn} : Mod(\Sigma_2)_{Horn} \to Mod(\Sigma_1)_{Horn}$ (daqui em diante, simplesmente $Mod(\phi)$) entre as categorias de modelos das correspondentes assinaturas *Horn* (em direção oposta) como segue:

1. em objetos (isto é, em modelos *Horn*): seja $A_2$ um $\Sigma_2$-modelo *Horn*, definimos um $\Sigma_1$-modelo *Horn* $Mod(\phi)(A_2)_{u_1} = A_{2,\phi(u_1)}$ constituído dos seguintes elementos: um conjunto carregador, operações e relações, i.e.,
   (a) para o conjunto unitário $\{u_1\}$ de $\Sigma_1$ : $\quad Mod(\phi)(A_2)_{u_1} =^{Def} A_{2,\phi(u_1)}$
   (b) para todo símbolo de operação $f_1 \in F_1$ : $\quad Mod(\phi)(A_2)_{f_1} =^{Def} A_{2,\phi(f_1)}$
   (c) para todo símbolo de predicado $p_1 \in P_1$ : $\quad Mod(\phi)(A_2)_{p_1} =^{Def} A_{2,\phi(p_1)}$
2. em morfismos (isto é, em homomorfismos entre modelos *Horn*): dado um homomorfismo $h_2 : A_2 \to B_2$ entre dois $\Sigma_2$ modelos *Horn*, definimos um $\Sigma_1$ homomorfismo como segue:

   $$Mod(\phi)(h_2) : Mod(\phi)(A_2) \to Mod(\phi)(B_2)$$

   sendo que:

   $$Mod(\phi)(h_2)_{u_1} =^{Def} h_{2,\phi(u_1)}$$

Nos diagramas abaixo temos a representação da definição do funtor esquecimento em objetos e setas.

$$(A_2)_{u_1} \xleftarrow{Mod(\phi)} A_{2,\phi(u_1)}$$

$$(A_2)_{f_1} \xleftarrow{Mod(\phi)} A_{2,\phi(f_1)}$$

$$(A_2)_{p_1} \xleftarrow{Mod(\phi)} A_{2,\phi(p_1)}$$

$$(h_2)_{u_1} \xleftarrow{Mod(\phi)} h_{2,\phi(u_1)}$$

**Nota 5.4 (Funtor Esquecimento)** Para simplificar a leitura, muitas vezes estamos utilizando simplesmente $\phi$ no lugar $\phi^u$, $\phi^F$ e $\phi^P$. Esta consideração é importante, pois $\phi$ é dependente do tipo argumentos.
Precisamos verificar se o funtor preserva as identidades e composição. Em geral, preservar as identidades quer dizer: $F(id_A) = id_{F(A)}$. Dado um $\Sigma_2$-modelo *Horn* $A_2$ e seus correspondentes homomorfismos identidades $id_{A_2} : A_2 \to A_2$, podemos concluir que:

$$\begin{array}{lll} Mod(\phi)(id_{A_2})_{u_1} & = id_{A_2,\phi(u_1)} & \text{(pela definição acima)} \\ & = id_{Mod(\phi)(A_2)_{u_1}} & \text{(pela definição acima)} \end{array}$$

Finalmente, o funtor deve ser compatível com as composições: $F(g \circ h) = F(g) \circ F(h)$. Dados dois $\Sigma_2$-homomorfismos $h_2 : A_2 \to B_2$, $g_2 : B_2 \to C_2$ induzindo a composição $g_2 \circ h_2 : A_2 \to C_2$, podemos concluir que:

$$\begin{array}{lll} Mod(\phi)(g_2 \circ h_2)_{u_1} & = (g_2 \circ h_2)_{\phi(u_1)} & \text{(como definido acima)} \\ & = g_{2,\phi(u_1)} \circ h_{2,\phi^u(u_1)} & \text{( composição em Set)} \\ & = Mod(\phi)(g_2)_{u_1} \circ Mod(\phi)(h_2)_{u_1} & \text{(como definido acima)} \end{array}$$

Assim, o funtor *Mod* é definido em objetos e setas na seguinte

***Proposição 5.2 (Funtor Modelo*** $Mod_{Horn}$***)*** *Definimos o funtor modelo* $Mod_{Horn} : \mathsf{Sign}^{op}_{Horn} \to \mathsf{Cat}$ *como um funtor mapeando cada:*

- $\Sigma$ *assinatura Horn para* $Mod(\Sigma)_{Horn}$*, a categoria de modelos Horn de acordo com a proposição 5.1.*
- *morfismo entre assinaturas Horn* $\phi : \Sigma_1 \to \Sigma_2$ *para* $Mod(\phi)_{Horn}$*, o funtor esquecimento de acordo com a definição 5.6.*

*Prova.* A prova que $Mod_{Horn}$ é realmente um funtor é a mesma do funtor $Mod_{MSEqtl}$ apresentada na instituição da *MSEqtl* (ver prova da definição 4.2 ). Esta prova deixa claro a contravariância do funtor modelo. Assim, quando trabalhamos a nível de modelos, que estão relacionados pelo morfismo entre assinaturas em Sign, os morfismos são tratados no sentido contrário, utilizando apenas as estruturas necessárias, do modelo da assinatura destino (no morfismo entre assinaturas), para satisfazer as estruturas da assinatura origem.

### 5.1.3 O Funtor Sentença $Sen_{Horn} : \mathsf{Sign}_{Horn} \to \mathsf{Set}$

Nesta seção, introduzimos o terceiro elemento da instituição, i.e., o funtor *sentença* (*Sen*). O funtor *Sen* "relaciona" cada assinatura em Sign com seu conjunto de sentenças em Set e cada morfismo entre assinaturas com um morfismo entre os conjuntos de sentenças das respectivas assinaturas.

Inicialmente, consideramos algumas definições importantes que incluem, termos de uma assinatura, fórmulas atômicas e fórmula *Horn* universal, a seguir apresentamos a tradução destas definições.

Um sistema de variáveis é um conjunto de variáveis, tal que nenhum nome de variáveis é igual a um símbolo de operação ou símbolo de predicado.

**Definição 5.7 (Variáveis e Sistema de Variáveis)** Os símbolos de variáveis estão num conjunto contável representado pelo símbolo $\Gamma$, que para cada assinatura *Horn* $\Sigma = \langle \{u\}, F, P \rangle$, admite escolher um subconjunto $X_u \subseteq \Gamma$. Para simplificar, daqui para frente usaremos simplesmente $X$. Chamamos o conjunto $X$ de $\Sigma$-sistema de variáveis, tal que $X \cap (F \cup P) = \varnothing$.

Novamente, para a lógica *Horn* o conjunto de termos é formado de maneira estrutural incluindo variáveis, símbolos de constantes e símbolos de operações (termos compostos).

**Definição 5.8 (Termos de uma assinatura)** Sejam $\Sigma = \langle \{u\}, F, P \rangle$ uma assinatura Horn e $X$ um $\Sigma$-sistema de variáveis. Chamamos

$$T(\Sigma, X) =^{def} T(\Sigma, X)_u$$

de conjuntos de termos sobre $X$, definido da seguinte maneira:

- $X_u \subseteq T(\Sigma, X)_u$ *(toda variável é um termo)*
- Para cada $f : \lambda \to u \in F$ temos que $f \in T(\Sigma, X)_u$ *(toda constante é um termo)*
- Para todo $f : u^n \to u \in F$ e para todo termo $t_1 \in T(\Sigma, X)_u, \ldots, t_n \in T(\Sigma, X)_u$ temos que $f(t_1, \ldots, t_n) \in T(\Sigma, X)_u$ *(termos compostos)*

As fórmulas atômicas desta lógica são constituídas de equações e predicados. Em detalhes, considere a seguinte

**Definição 5.9 (Fórmulas Atômicas)** Sejam $\Sigma$ uma assinatura *Horn* e $X$ um $\Sigma$-sistema de variáveis. Definimos duas formas de $\Sigma$-fórmulas atômicas sobre $X$:

1. Assumimos que o símbolo equacional ' $=$ ' não está incluído em $P$. Sejam $t, t' \in T(\Sigma, X)_u$, dois $\Sigma$-termos. Então:

   $$t = t'$$

   é uma $\Sigma$-fórmula atômica sobre $X$ chamada equação.

2. Dado um símbolo de predicado $p_{u^n} \in P$ e termos $t_1 \in T(\Sigma, X)_u, \ldots, t_n \in T(\Sigma, X)_u$. Então,

$$p(t_1, \ldots, t_n)$$

é uma $\Sigma$-fórmula atômica sobre $X$.

$\Sigma$-fórmulas atômicas podem ser chamadas de átomos.

**Nota 5.5 (Fórmulas Atômicas)** Para indicar que uma fórmula atômica é baseada em um sistema de variáveis $X$ escreveremos $(X \vdash \varphi)$.

Um fórmula *Horn* universal é uma implicação da forma capturada na próxima

**Definição 5.10 (Fórmula *Horn* Universal)** Sejam $\Sigma$ uma assinatura *Horn*, $X$ um $\Sigma$-sistema de variáveis e $\varphi_1, \ldots, \varphi_n, \varphi$ $\Sigma$-fórmulas atômicas sobre $X$ de acordo com a definição 5.9. Chamamos

$$\eta = \langle \varphi_1, \ldots, \varphi_n \Rightarrow \varphi \rangle$$

de **fórmula *Horn* universal** sobre $\Sigma$ e $X$.

**Nota 5.6** A definição acima requer que qualquer símbolo de variável usado em algum $\varphi_i, \varphi$ esteja incluído em $X$. Novamente, enfatizando o tratamento de variáveis (ver nota 5.5), se queremos expressar uma fórmula atômica como uma fórmula bem formada, devemos usá-la da maneira especificada acima, isto é, temos que referenciar o sistema de variáveis.

Dado um morfismo entre assinaturas em Sign. O funtor *Sen* é definido em setas como uma tradução entre as sentenças dessas assinaturas. Assim, para todo sistema de variáveis na assinatura origem temos um respectivo na assinatura destino.

**Definição 5.11 (Sistema de Variáveis Induzido)** Sejam $\Sigma_1, \Sigma_2$ assinaturas *Horn*, $\phi : \Sigma_1 \to \Sigma_2$ um morfismo entre assinaturas *Horn* e $X_1$ um $\Sigma_1$-sistema de variáveis. Chamamos

$$\overline{\phi}(X_1) = (\overline{\phi}(X_1)_{u_2})$$

de $\Sigma_2$-sistema de variáveis induzido por $\phi$, tal que:

$$\overline{\phi}(X_1)_{u_2} = \langle X_{1,u_1} \mid \phi(u_1) = u_2 \rangle$$

Isto é,

$$X_{1,u_1} \xmapsto{\quad\overline{\phi}\quad} X_{1,\phi(u_1)}$$

Desta forma, para cada termo da assinatura origem de $\phi$ temos um respectivo na assinatura destino.

**Definição 5.12 (Tradução de Termos)** Sejam $\Sigma_1, \Sigma_2$ duas assinaturas, $\phi : \Sigma_1 \to \Sigma_2$ um morfismo entre essas assinaturas e um $\Sigma_1$-termo $t_1 \in T(\Sigma_1, X_1)_{u_1}$. Definimos a tradução de $t_1$, escrita

$$\overline{\phi}_{u_1}(t_1) \in T(\Sigma_2, \overline{\phi}(X_1))_{\phi(u_1)}$$

recursivamente sobre a definição de termos, da seguinte maneira:

$(i)$ $\overline{\phi}_{u_1}(x_1) = x_1(\in \overline{\phi}(X_1))$ para todo $x_1 \in X_{1,u_1}$ ( sobre variáveis )
$(ii)$ $\overline{\phi}_{u_1}(f_1) = \phi(f_1)$ para todo $f_1 : \lambda \to u_1 \in F_1$ (sobre constantes)
$(iii)$ $\overline{\phi}_{u_1}(f_1(t_1, \ldots, t_n)) = \phi(f_1)(\overline{\phi}_{u^n}(t_1, \ldots, t_n))$ para todo $f_1(t_1, \ldots, t_n)$ com $f_1 : w \to u_1 \in F_1$ e $t1 \in T(\Sigma_1, X_1)_u, \ldots, t_n \in T(\Sigma_1, X_1)_u$.

**Nota 5.7 (Tradução de Termos)** A função extendida $\overline{\phi}$ que faz a tradução de termos é da seguinte forma: $\overline{\phi}_{u_1} : T(\Sigma_1, X_1)_{u_1} \to T(\Sigma_2, \overline{\phi}(X_1))_{\phi(u_1)}$. É importante verificar que os nomes das variáveis permanecem os mesmo, o que muda é o sort.

O funtor *Sen* traduz fórmulas atômicas, i.e., equações e predicados da assinatura origem em $\phi$ para a destino, para todo $\phi$ em Sign.

**Definição 5.13 (Tradução de Fórmulas Atômicas)** Sejam $\Sigma_1$, $\Sigma_2$ assinaturas *Horn*, $\phi : \Sigma_1 \to \Sigma_2$ um morfismos entre essas assinaturas e uma $\Sigma_1$-fórmula atômica $\varphi$. Definimos a tradução de $\overline{\phi}(\varphi)$, de acordo com a estrutura da definição de fórmulas atômicas (definição 5.9 ), como segue:

$(i)$ para $\varphi \equiv (X \vdash t = t')$ como $\overline{\phi}(\varphi) =^{def} (\overline{\phi}(X) \vdash \overline{\phi}_{u_1}(t) = \overline{\phi}_{u_1}(t'))$, com $t, t' \in T(\Sigma_1, X_1)_{u_1}$
$(ii)$ para $\varphi \equiv (X \vdash p(t_1, \ldots, t_n))$, como $\overline{\phi}(\varphi) =^{def} (\overline{\phi}(X) \vdash \phi(p)(\overline{\phi}_{u^n}(t_1) \ldots, \overline{\phi}_{u^n}(t_n)))$, com $t1 \in T(\Sigma_1, X_1)_u, \ldots, t_n \in T(\Sigma_1, X_1)_u$ para todo $p_{u^n} \in P_1$.

utilizando a tradução de termos definida acima (definição 5.12). A definição de tradução de fórmulas atômicas é representada pelos diagramas que seguem.

$$(X \vdash t = t') \longmapsto (\overline{\phi}(X) \vdash \overline{\phi}_{u_1}(t) = \overline{\phi}_{u_1}(t'))$$

$$(X \vdash p(t_1, \ldots, t_n)) \longmapsto (\overline{\phi}(X) \vdash \phi(p)(\overline{\phi}_{u^n}(t_1), \ldots, \overline{\phi}_{u^n}(t_n)))$$

Assim, podemos definir a tradução de fórmulas *Horn* Universais.

**Definição 5.14 (Tradução de Fórmulas *Horn* Universais)** Sejam $\Sigma_1$, $\Sigma_2$ assinaturas *Horn*, $\phi : \Sigma_1 \to \Sigma_2$ um morfismo entre assinaturas, $X_1$ um $\Sigma_1$-sistema de variáveis e uma fórmula *Horn* universal $\eta = (X_1 \vdash \varphi_1, \ldots, \varphi_n \Rightarrow \varphi)$ sobre $\Sigma_1$ e $X_1$, definimos a tradução de $\eta$ como:

$$\overline{\phi}(\eta) = (\overline{\phi}(X_1) \vdash \overline{\phi}(\varphi_1), \ldots, \overline{\phi}(\varphi_n) \Rightarrow \overline{\phi}(\varphi))$$

Utilizando a definição de tradução do sistema de variáveis (definição 5.11) e a definição acima de tradução de fórmula atômicas. Desta maneira, temos:

$$(X_1 \vdash \varphi_1, \ldots, \varphi_n \Rightarrow \varphi) \longmapsto (\overline{\phi}(X_1) \vdash \overline{\phi}(\varphi_1), \ldots, \overline{\phi}(\varphi_n) \Rightarrow \overline{\phi}(\varphi))$$

Apresentamos a seguir o terceiro ingrediente da instituição *Horn*, i.e., o funtor *Sen*.

**Definição 5.15 (O Funtor Sentença $Sen_{Horn} : \mathsf{Sign}_{Horn} \to \mathsf{Set}$)** Definimos o funtor sentença para *lógica Horn*, escrito $Sen_{Horn} : \mathsf{Sign}_{Horn} \to \mathsf{Set}$ (daqui em diante simplesmente $Sen$) por:

- $Sen(\Sigma)$ que é o conjunto de todas as fórmulas *Horn* universais sobre $\Sigma$.
- $Sen(\phi) = \overline{\phi} : Sen(\Sigma_1) \to Sen(\Sigma_2)$ que é a tradução de fórmulas Horn universais para qualquer dado $\phi : \Sigma_1 \to \Sigma_2$.

**Nota 5.8** A verificação de que $Sen_{Horn} : \mathsf{Sign}_{Horn} \to \mathsf{Set}$ é realmente um funtor segue o mesmo princípio da nota 4.4 mostrada na instituição da lógica equacional multi sortida para o funtor $Sen_{MSEqtl}$.

### 5.1.4 Condição de Satisfação

Nesta seção, apresentamos a condição de satisfação de fórmulas por modelos, como último ítem desta lógica.

Uma atribuição de variáveis é uma função, tal que para cada variável no sistema de variáveis é dado um valor no conjunto carregador da estrutura de interpretação.

**Definição 5.16 (Atribuição de Variáveis)** Dada uma assinatura *Horn* $\Sigma \in \mathsf{Sign}_{Horn}$, um $\Sigma$-modelo *Horn* $A \in\mid Mod(\Sigma)_{Horn} \mid$ e um $\Sigma$-sistema de variáveis $X$. Uma **atribuição** de um símbolo de variável em $X$ para valores em $A$:

$$\alpha : X \to A = \langle \alpha_u : X_u \to A_u \rangle$$

é um conjunto de funções, que atribui para cada símbolo de variável $x$ um valor $\alpha(x)$ no modelo (veja Figura 5.1)

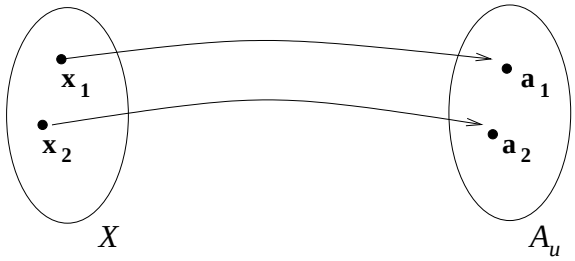



FIGURA 5.1 – Atribuição *Horn*

A definição de avaliação de termos é dada de acordo com a estrutura de termos, i.e., para variáveis (ou seja, a própria atribuição), para constantes e para termos complexos.

**Definição 5.17 (Avaliação de Termos)** Sejam $\Sigma \in\mid \mathsf{Sign}_{Horn} \mid$ uma assinatura, $A \in Mod(\Sigma)$ um $\Sigma$-modelo e $X$ um $\Sigma$-sistema de variáveis, uma atribuição $\alpha : X \to A$ e um $\Sigma$-termo $t \in T(\Sigma, X)_u$. A **avaliação** de $t$ sob $\alpha$:

$$\overline{\alpha} : T(\Sigma, X) \to A; \;\; \overline{\alpha} = \overline{\alpha}_u$$

é definida recursivamente, de acordo com a definição de termos (definição 5.8 ):

$(i)$ $\overline{\alpha}_u(x) =^{def} \alpha_u(x)$ para todo $x \in X_u$
$(ii)$ $\overline{\alpha}_u(f) =^{def} A_f$ para todo $f : \lambda \to u \in F$
$(iii)$ $\overline{\alpha}_u(f(t_1, \ldots, t_n)) =^{def} A_f(\overline{\alpha}_{u^n}(t_1), \ldots, \overline{\alpha}_{u^n}(t_n)))$ para todo $f : u^n \to u \in F$ e $t_1 \in T(\Sigma, X)_u, \ldots, t_n \in T(\Sigma, X)_u$.

**Nota 5.9 (Avaliação de Termos)** Quando utilizamos $t \in T(\Sigma, X)$, significa que $X$ é o conjunto máximo de símbolos que podem ser usados para construir $t$. $\overline{\alpha}_{u^n}$ é a extensão de $\overline{\alpha}_u$ para palavras de $\{u\}^*$.

Da mesma maneira que para termos, temos também a avaliação de fórmulas atômicas, que é capturada pela seguinte

**Definição 5.18 (Avaliação de fórmulas atômicas)** Sejam $\Sigma = \langle \{u\}, F, P \rangle$ uma assinatura *Horn*, $A = \langle A(u), A(F), A(P) \rangle$ um modelo, $X$ um sistema de variáveis, $\alpha : X \to A$ uma atribuição e $\varphi$ uma **fórmula atômica** *Horn*, segundo a definição 5.9. A avaliação de fórmulas $\varphi$ é definida como segue:

- $\overline{\alpha}_u(t) = \overline{\alpha}_u(t')$, i.e., a avaliação dos termos $t, t'$ produz o mesmo resultado em $A$;
- $\overline{\alpha}_u(p(t_1, \ldots, t_n)) =^{def} (\overline{\alpha}_{u_1}(t_1), \ldots, \overline{\alpha}_{u_n}(t_n)) \in A(P)$, i.e., a avaliação da termo tupla $t_1 \in T(\Sigma, X)_u, \ldots, t_n \in T(\Sigma, X)_u$ em $A$ produz um elemento da relação $A(P)$.

Como mencionado no ínicio desta seção, dada uma assinatura $\Sigma$ e uma $\Sigma$-estrutura de interpretação temos o conceito de satisfação de $\Sigma$-sentenças nesta estrutura.

**Definição 5.19 (Satisfação de uma fórmula atômica em uma atribuição)**

- Caso 1: Sejam $\varphi = (X \vdash t, t')$ uma equação sobre $\Sigma$, $A = \langle A(u), A(F), A(P) \rangle$ um modelo *Horn* e $\alpha : X \to A$ uma atribuição. Então $A$ satisfaz $\varphi$ em $\alpha : X \to A$, escrita $A, \alpha \models_\Sigma \varphi$, se e somente se $\overline{\alpha}(t) = \overline{\alpha}(t')$.
- Caso 2: Sejam $\varphi = (X \vdash p(t))$ um predicado sobre $\Sigma$, $A = \langle A(u), A(F), A(P) \rangle$ um modelo *Horn* e $\alpha : X \to A$ uma atribuição. Então $A$ satisfaz $\varphi$ em $\alpha : X \to A$, escrita $A, \alpha \models_\Sigma \varphi$, se e somente se $\overline{\alpha}(t) \in A(P)$.

O conceito de satisfação de uma sentença em um modelo vale se, e somente se, a satisfação vale para toda atribuição.

**Definição 5.20 (Satisfação de uma fórmula atômica em um modelo)**

- Caso 1: Sejam $\varphi = (X \vdash t = t')$ uma equação sobre $\Sigma$ e $A = \langle A(u), A(F), A(P) \rangle$ um modelo *Horn*. Então, $A$ satisfaz $\varphi$, escrita, $A \models_\Sigma \varphi$ se e, somente se, $A, \alpha \models_\Sigma \varphi$ para toda atribuição $\alpha : X \to A$.
- Caso 2: Sejam $\varphi = (X \vdash p(t_1, \ldots, t_n))$ um predicado sobre $\Sigma$ e $A = \langle A(u), A(F), A(P) \rangle$ um modelo *Horn*. Então, $A$ satisfaz $\varphi$, escrita, $A \models_\Sigma \varphi$ se e, somente se, $A, \alpha \models_\Sigma \varphi$ para toda atribuição $\alpha : X \to A$.

Da mesma maneira que para fórmulas atômicas, segue a satisfação de fórmulas *Horn* universais em uma atribuição.

**Definição 5.21 (Satisfação de uma fórmula *Horn* em uma atribuição)** Seja $\eta = \langle X \vdash \varphi_1, \ldots, \varphi_n \Rightarrow \varphi \rangle$ uma fórmula *Horn* sobre $\Sigma$, $A = \langle A(u), A(F), A(P) \rangle$ um $\Sigma$-modelo e $\alpha : X \to A$ uma atribuição. Então $A$ satisfaz $\eta$ em $\alpha$, escrita, $A, \alpha \models_\Sigma \eta$, se e somente se, sempre que $A, \alpha \models_\Sigma \varphi_1, \ldots, A, \alpha \models_\Sigma \varphi_n$ temos que $A, \alpha \models_\Sigma \varphi$.

**Definição 5.22 (Satisfação de uma fórmula *Horn* em um modelo)** Seja $\eta = \langle X \vdash \varphi_1, \ldots, \varphi_n \Rightarrow \varphi \rangle$ uma fórmula *Horn* sobre $\Sigma$ e $A = \langle A(u), A(F), A(P) \rangle$ um $\Sigma$-modelo. Então $A$ satisfaz $\eta$, escrita $A \models_\Sigma \eta$, se e somente se, sempre que $A, \alpha \models_\Sigma \varphi_1, \ldots, A, \alpha \models_\Sigma \varphi_n$ temos que $A, \alpha \models_\Sigma \varphi$ para toda atribuição $\alpha : X \to A$.

A condição de satisfação em uma instituição é tal que, quando mapeamos uma assinatura para outra, então a satisfação de fórmulas pelos modelos (ver definição 5.22), muda coerentemente. Neste contexto, os nomes não são importantes, no entanto as **estruturas** devem ser preservadas.

Sabemos que, na tradução de termos, os nomes das variáveis não mudam. Assim, toda a atribuição de variáveis para valores da álgebra da assinatura origem temos uma semelhante na destino e vice-versa.

**Definição 5.23 (Atribuições Correspondentes)** Sejam $\phi : \Sigma_1 \to \Sigma_2$ um morfismo entre assinaturas *Horn*, $X_1$ um $\Sigma_1$-sistema de variáveis e $A_2$ um $\Sigma_2$ modelo *Horn*.

1. Dada uma atribuição $\alpha_2 : \overline{\phi}(X_1) \to A_2$, com $\overline{\phi}(X_1)$ um $\Sigma_2$ sistema de variáveis induzido de acordo com a definição 5.11, chamamos $\alpha_1 : X \to Mod(\phi)(A_2)$ de atribuição correspondente, definida como:

$$\alpha_{1,u_1}(x) =^{Def} \alpha_{2,\phi(u_1)}(x)$$

   para todo $x_1 \in X_{1,u1}$

2. Dada uma atribuição $\alpha_1 : X_1 \to Mod(\phi)(A_2)$, chamamos $\alpha_2 : \overline{\phi}(X_1) \to A_2$ de atribuição correspondente, definida como:

$$\alpha_{2,u_2}(x) =^{Def} \alpha_{1,u_1}(x)$$

   para todo $x \in \overline{\phi}(X_1)$, tal que $u_2 = \phi(u_1)$.

Agora, precisamos definir atribuições estendidas para termos, átomos e fórmulas, isto é, dado um par de atribuições correspondentes, podemos extender ambas produzindo a noção de **avaliações correspondentes**.

***Proposição 5.3 (Avaliações Correspondentes)*** *Seja*

- *Horn* $\phi : \Sigma_1 \to \Sigma_2$ *um morfismo entre assinaturas Horn;*
- $X_1$ *um* $\Sigma_1$*- sistema de variáveis;*

- $A_2 \in |\ Mod(\Sigma_2)\ |$ *um* $\Sigma_2$*-modelo Horn e*
- *as atribuições correspondentes, como definidas em 5.23.*

*Então, temos as seguintes* ***avaliações correspondentes****:*

$$\overline{\alpha}_1 : T(\Sigma_1, X_1) \to Mod(\phi)(A_2)$$
$$\overline{\alpha}_2 : T(\Sigma_2, \overline{\phi}(X_1)) \to A_2$$

*Tal que, para todo* $t \in T(\Sigma_1, X_1)_{u_1}$*, a seguinte propriedade vale:*

$$\overline{\alpha}_{1,u_1}(t) = \overline{\alpha}_{2,\phi(u_1)}(\overline{\phi}_{u_1}(t))$$

*Prova.* Prova análoga a proposição 4.3.

***Lema 5.1 (Condição de Satisfação para Átomos)*** *Sejam* $\phi : \Sigma_1 \to \Sigma_2$ *um morfismo entre assinaturas,* $A_2$ *um* $\Sigma_2$*-modelo ,* $X_1$ *um* $\Sigma_1$*-sistema de variáveis e uma* $\varphi$ $\Sigma_1$*-fórmula atômica sobre* $X_1$*, segundo a definição 5.9. Para toda atribuição* $\alpha_2 : \overline{\phi}(X_1) \to A$ *temos que:*

$$A_2 \models_{\Sigma_2} Sen(\phi)(\varphi) \Leftrightarrow Mod(\phi)(A_2) \models_{\Sigma_1} \varphi$$

*Prova.* O lema contém uma proposição sobre $\Sigma_1$ átomos $\varphi$ sobre $X_1$. Isto é, temos que prová-lo de acordo com a estrutura de átomos, seguindo a definição 5.9. A prova para equações é análogo a prova apresentada no teorema 4.1. Segue abaixo a prova para predicados. Seja $\varphi = \langle X \vdash p_1(t_1, \ldots, t_n)\rangle$ para $t_1 \in T(\Sigma_1, X_1)_u, \ldots, t_n \in T(\Sigma_1, X_1)_u$ e $p_{1,w} \in P_1$

$$\begin{array}{lr}
A_2 \models_{\Sigma_2} Sen(\phi)(\varphi) & \\
\Leftrightarrow A_2 \models_{\Sigma_2} (\overline{\phi}_u(X_1) \vdash \overline{\phi}_u(p_1)(\overline{\phi}_u(t_1), \ldots, \overline{\phi}_u(t_n)) & \text{(def. de } \varphi) \\
\Leftrightarrow A_2 \models_{\Sigma_2} \phi_u(p_1)(\overline{\phi}_u(t_1) \ldots, \overline{\phi}_u(t_n)) & \text{(def. de 5.13)} \\
\Leftrightarrow \overline{\alpha}_{2,\phi(u)}(\overline{\phi}_u(t_1)), \ldots, \overline{\alpha}_{2,\phi(u)}(\overline{\phi}_u(t_n)) \in A_{2,\phi(p)} & \text{(def. 5.20)} \\
\Leftrightarrow \overline{\alpha}_{1,u}(t_1, \ldots, t_n) \in A_{2,\phi(p)} & \text{(prop. 5.3)} \\
\Leftrightarrow \overline{\alpha}_{1,u}(t_1, \ldots, t_n) \in Mod(\phi)(A_2)_p & \text{(def. 5.6)} \\
\Leftrightarrow Mod(\phi)(A_2) \models_{\Sigma_1} p_1(t_1, \ldots, t_n) & \text{(def. 5.20)} \\
\Leftrightarrow Mod(\phi)(A_2) \models_{\Sigma_1} \varphi & \text{(def. } \varphi)
\end{array}$$

***Corolário 5.1 (Condição de Satisfação)*** *Dada um morfismo entre assinaturas Horn* $\phi : \Sigma_1 \to \Sigma_2$*, um* $\Sigma_2$ *modelo Horn* $A_2$*, um* $\Sigma_1$*-sistema de variáveis* $X_1$ *e uma fórmula Horn universal* $\eta = \langle X_1 \vdash \varphi_1, \ldots, \varphi_n \Rightarrow \varphi\rangle$*, a seguinte* condição de satisfação *vale:*

$$A_2 \models_{\Sigma_2} Sen(\phi)(\eta) \Leftrightarrow Mod(\phi)(A_2) \models_{\Sigma_1} \eta$$

*Prova.* Este corolário tem uma proposição sobre fórmulas *Horn* universais, então temos que provar de acordo com sua estrutura (ver definição 5.10). Seja $\alpha_2 : \overline{\phi}(X) \to A_2$ uma atribuição. Temos para toda fórmula $\eta = \langle X_1 \vdash \varphi_1, \ldots, \varphi_n \Rightarrow \varphi\rangle$ a seguinte prova. Por hipótese temos que $A_2 \models_{\Sigma_2} Sen(\phi)(\eta)$. Agora, temos que $A_2 \models_{\Sigma_2} Sen(\phi)(\eta)$ é equivalente a $A_2 \models_{\Sigma_2} (\overline{\phi}(X) \vdash \overline{\phi}(\varphi_1), \ldots, \overline{\phi}(\varphi_n) \Rightarrow \overline{\phi}(\varphi))$ pela definição de $\eta$. Mas temos que isto é equivalente a $A_2, \alpha_2 \models_{\Sigma_2} \overline{\phi}(\varphi_1), \ldots, A_2, \alpha_2 \models_{\Sigma_2} \overline{\phi}(\varphi_n) \Rightarrow A_2, \alpha_2 \models_{\Sigma_2} \overline{\phi}(\varphi)$ pela definição 5.22. Assim, segue $Mod(\phi)(A_2), \alpha_1 \models_{\Sigma_1} \varphi_1, \ldots, Mod(\phi)(A_2), \alpha_1 \models_{\Sigma_1} \varphi_n \Rightarrow Mod(\phi)(A_2), \alpha_1 \models_{\Sigma_1} \varphi$ pela definição 5.23. Novamente, isto é equivalente a $Mod(\phi)(A_2) \models_{\Sigma_1} (X_1 \vdash \varphi_1, \ldots, \varphi_n \Rightarrow \varphi)$, pela definição 5.22. Finalmente, segue $Mod(\phi)(A_2) \models_{\Sigma_1} \eta$ pela definição de $\eta$, que era o que tinha que ser provado.

# 6 Mapeamentos

Neste capítulo, apresentamos dois tipos de mapeamentos entre instituições, mais precisamente, *mapeamento Puro* e *mapeamento Simples*, que foram introduzidos em [MES 89]. Um mapeamento Puro entre duas lógicas é essencialmente uma estrutura que traduz assinaturas e sentenças de $\mathcal{L}$ para $\mathcal{L}'$ e que mapeia modelos de $\mathcal{L}'$ para $\mathcal{L}$. Como mapeamento puro temos uma seta que relaciona (*representa*) uma instituição $\mathcal{L}$ em outra $\mathcal{L}'$, tal que as assinaturas de $\mathcal{L}$ são mapeadas para assinaturas de $\mathcal{L}'$. Já um mapeamento simples traduz assinaturas de $\mathcal{L}$ para *teorias* de $\mathcal{L}'$. Isto acontece pois a tradução da informação na assinatura da lógica origem pode requerer axiomas na lógica destino. Como exemplos, mostramos detalhadamente um mapeamento puro entre a *MSEqtl* e a *OSEqtl* e um mapeamento simples entre a *MSEqtl* e a lógica *Horn*. Também, atenção especial é dada para as propriedades lógicas desses mapeamentos.

## 6.1 Mapeamento Puro

De forma a motivar a definição de *mapeamento Puro* considere as duas especificações de listas nas lógicas *MSEqtl* e *OSEqtl* apresentadas abaixo.

*ThList*, *MSEqtl*

Sorts
$nat, list$
Opns
$empty :\rightarrow list$
$head : list \rightarrow nat$
$tail : list \rightarrow list$
$cons : nat, list \rightarrow list$

---

$[x : nat, l : list] head(cons(x, l)) = x : nat$
$[x : nat, l : list] tail(cons(x, l)) = l : list$
$[\ ] tail(empty) = empty$

*ThList*, *OSEqtl*

Sorts
$nat, list$
Subsorts
$nat < nat, list < list$
Opns
$empty :\rightarrow list$
$head : list \rightarrow nat$
$tail : list \rightarrow list$
$cons : nat, list \rightarrow list$

---

$[x : nat, l : list] head(cons(x, l)) = x : nat$
$[x : nat, l : list] tail(cons(x, l)) = l : list$
$[\ ] tail(empty) = empty$

O ponto essencial a ser observado nessas especificações é a maneira como a especificação de listas em *MSEqtl* pode ser "representada" em *OSEqtl*. Mais precisamente, note que isto foi possível através da inclusão da relação de ordem parcial identidade no conjunto de sorts, representada por $<$. Desta maneira, o que está acontecendo é que uma assinatura *MSEqtl* pode ser codificada em uma assinatura *OSEqtl* com a inclusão da relação $<$ no conjunto de sorts. Entretanto, podemos verificar que as sentenças permanecem iguais nas duas especificações.

Como cada uma dessas especificações é denotada por uma classe de modelos (álgebras) então, elas também estão relacionadas semanticamente. Contudo, os modelos que interpretam as estruturas sintáticas dessas dessas duas lógicas são essencialmente diferentes, i.e., na *OSEqtl*, as estruturas de interpretação requerem uma relação de ordem parcial nos conjuntos, que não existe para as estruturas que interpretam *MSEqtl*. No entanto, podemos pensar em álgebras *MSEqtl* como álgebras *OSEqtl* com uma relação de ordem parcial identidade nos conjuntos carregadores. Desta forma, temos uma maneira de relacionar os modelos das duas especificações apresentadas acima, i.e., como esta relação é a identidade podemos "traduzir" a álgebra que interpreta listas em *OSEqtl* para uma álgebra que denota listas em *MSEqtl*.

Esta idéia de *representar* uma lógica $\mathcal{L}$ em uma outra lógica $\mathcal{L}'$ está formalizada na seguinte

**Definição 6.1 (Mapeamento puro de instituições)** Sejam duas instituições $\mathcal{L} = (\mathsf{Sign}, Sen, Mod, \models)$ e $\mathcal{L}' = (\mathsf{Sign}', Sen', Mod', \models')$. Um *mapeamento puro* $(\Phi, \alpha, \beta) : \mathcal{L} \to \mathcal{L}'$ consiste de

- um funtor $\Phi : \mathsf{Sign} \to \mathsf{Sign}'$;
- uma transformação natural $\alpha : Sen \Rightarrow \Phi;\ Sen' : \mathsf{Sign} \to \mathsf{Set}$ e
- uma transformação natural $\beta : \Phi^{op};\ Mod' \Rightarrow Mod : \mathsf{Sign}^{op} \to \mathsf{Cat}$

tal que para cada $\Sigma \in |\ \mathsf{Sign}\ |$, $\varphi \in Sen(\Sigma)$ e $M' \in |Mod'(\Phi(\Sigma))|$ a seguinte propriedade vale:

$$\beta(\Sigma)(M') \models_\Sigma \varphi \Leftrightarrow M' \models'_{\Phi(\Sigma)} \alpha(\Sigma)(\varphi) \quad (\textit{Condição Pura})$$

$$\begin{array}{ccc} \mathsf{Sign} & & Mod(\Sigma) \xleftarrow{\models_\Sigma} Sen(\Sigma) \\ {\scriptstyle\Phi}\downarrow & & {\scriptstyle\beta(\Sigma)}\uparrow \qquad\qquad \downarrow{\scriptstyle\alpha(\Sigma)} \\ \mathsf{Sign}' & & Mod'(\Phi(\Sigma)) \xleftarrow[\models'_{\Phi(\Sigma)}]{} Sen'(\Phi(\Sigma)) \end{array}$$

Considere agora a Figura 6.1, Onde os dois circulos maiores representam duas lógicas (instituições). Note que, a seta pontilhada no topo é o funtor $\Phi : \mathsf{Sign} \to \mathsf{Sign}'$ e a seta sólida inferior é a transformação natural $\beta$. Já tradução pontilhada inferior denota a transformação $\alpha$. Podemos observar, que $\beta$ relaciona os modelos das lógicas no sentido contrário de $\Phi$ (em $\mathsf{Cat}$), e que $\alpha$ ralaciona as fórmulas das lógicas no mesmo sentido de $\phi$ (em $\mathsf{Set}$). Finalmente, o símbolo $\models$ no centro do desenho aparece como a *condição pura*. Essa condição diz que os modelos da lógica destino (de $\Phi$) satisfazem os axiomas traduzidos por $\alpha$ se e somente se os modelos traduzidos por $\beta$ satisfazem os axiomas da lógica origem.

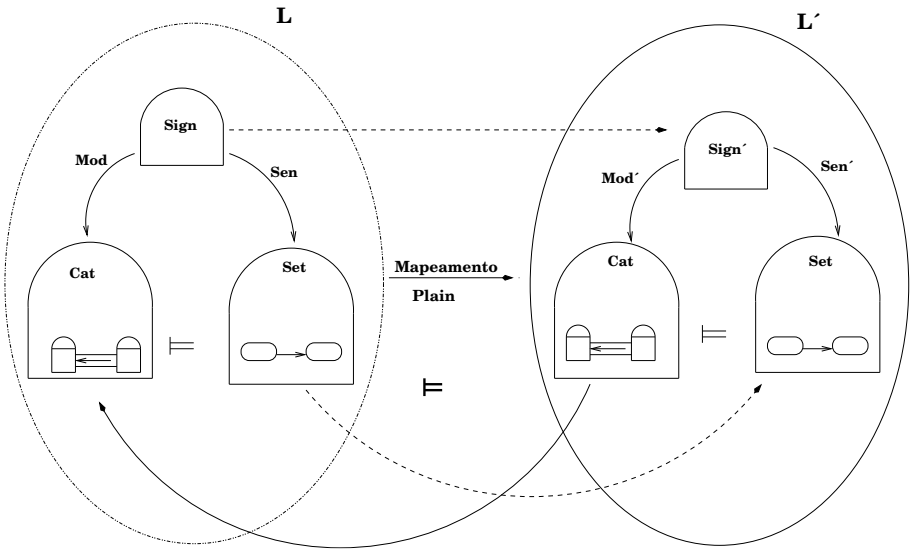



FIGURA 6.1 – Mapeamento Puro

### 6.1.1 Exemplo

Nesta seção, discutimos mais detalhadamente o *mapeamento puro* já introduzido no início desta seção, i.e., entre a lógica *MSEqtl* e a lógica *OSEqtl*.

**Exemplo 6.1 (Mapeamento Puro *MSEqtl* $\rightarrow$ *OSEqtl*)** Um mapeamento puro $(\Phi, \alpha, \beta) : \mathcal{L}_{MSEqtl} \rightarrow \mathcal{L}_{OSEqtl}$ é definido como segue:

- para cada assinatura $\Sigma = \langle S, F \rangle \in |\ \mathsf{Sign}_{MSEqtl}\ |$ temos uma $\Sigma = \langle S, F, < \rangle \in \mathsf{Sign}_{OSEqtl}$, ou seja, $\Phi\langle S, F \rangle =^{def} \langle S, F, < \rangle$, onde $<$ é a ordem parcial identidade no conjunto de sorts $S$.

- para cada $\Sigma = \langle S, F \rangle \in |\ \mathsf{Sign}_{MSEqtl}\ |$ e para cada equação $\langle X \vdash t = t' \rangle \in Sen_{MSEqtl}\langle S, F \rangle$ temos uma equação ordenada por sorts $\langle X \vdash t = t' \rangle \in Sen_{OSEqtl}(\Phi(\langle S, F \rangle))$.

- para cada $\Sigma = \langle S, F \rangle \in \mathsf{Sign}_{MSEqtl}$ e para cada modelo ordenado por sorts $M' = \langle A_s \mid s \in S, A_f \mid f \in F, = \rangle \in |\ Mod'_{OSEqtl}(\Phi(\langle S, F \rangle))\ |$ temos uma álgebra multi sortida $\beta(\Sigma)(M') = \langle A_s \mid s \in S, A_f \mid f \in F \rangle \in |\ Mod_{MSEqtl}(\langle S, F \rangle)\ |$, onde a ordem parcial nos carregadores é esquecida.

Além disso, a partir dessas informações temos, para todo $\varphi \in Sen_{MSEqtl}(\Sigma)$ e para todo $M' \in |\ Mod'_{OSEqtl}(\Phi(\langle S, F \rangle))\ |$ que:

$$\beta(\Sigma)(M') \models_{\Sigma_{MSEqtl}} \varphi \Leftrightarrow M' \models_{\Sigma_{OSEqtl}} \alpha(\Sigma)(\varphi)$$

que é a condição pura do mapeamento.

Na sequência mostramos que as definições dadas acima satisfazem os requisitos para serem um mapeamento puro, i.e., que globalmente elas definem respectivamente funtores entre assinaturas e transformações naturais entre sentenças e modelos.

***Proposição 6.1*** *As definições apresentadas no exemplo 6.1 definem um mapeamento Puro* $(\Phi, \alpha, \beta) : \mathcal{L}_{MSEqtl} \rightarrow \mathcal{L}_{OSEqtl}$.

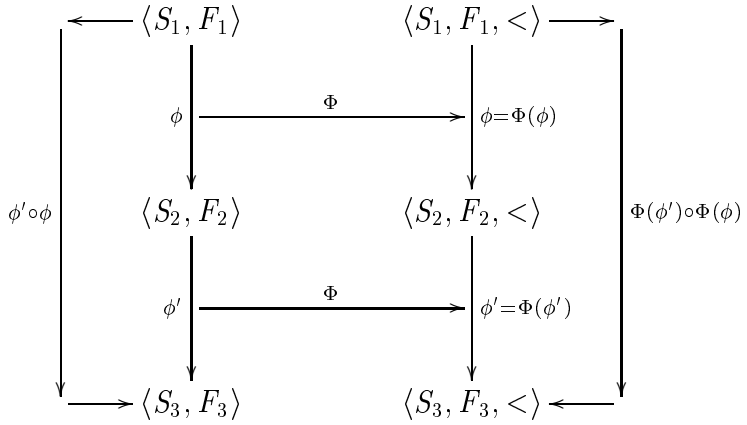
*Prova.* A proposição acima requer que verifiquemos uma diversidade de itens.

1. Começamos com o funtor $\Phi : \mathsf{Sign}_{MSEqtl} \to \mathsf{Sign}_{OSEqtl}$. Pela definição temos que para todo $\phi : \Sigma_1 \to \Sigma_2 \in \mathsf{Sign}_{MSEqtl}$ existe um $\Phi(\phi) : \Phi(\Sigma_1) \to \Phi(\Sigma_2) \in \mathsf{Sign_OSEqtl}$ tal que $\Phi(\phi) = \phi$, i. e.,

$$\begin{array}{ccc} \langle S_1, F_1 \rangle & \xrightarrow{\Phi} & \langle S_1, F_1, < \rangle \\ {\scriptstyle \phi}\downarrow & & \downarrow{\scriptstyle \Phi(\phi)=\phi} \\ \langle S_2, F_2 \rangle & \xrightarrow[\Phi]{} & \langle S_2, F_2, < \rangle \end{array}$$

   Temos que investigar se o funtor $\Phi$ é compatível com a composição e identidade.

   - Composição, i.e., sejam $\phi : \Sigma_1 \to \Sigma_2$, $\phi' : \Sigma_2 \to \Sigma_3$ morfismos em $\mathsf{Sign}_{MSEqtl}$ temos que constatar se $\Phi(\phi' \circ \phi) = \Phi(\phi') \circ \Phi(\phi)$. Esta prova pode ser resumida no diagrama abaixo:

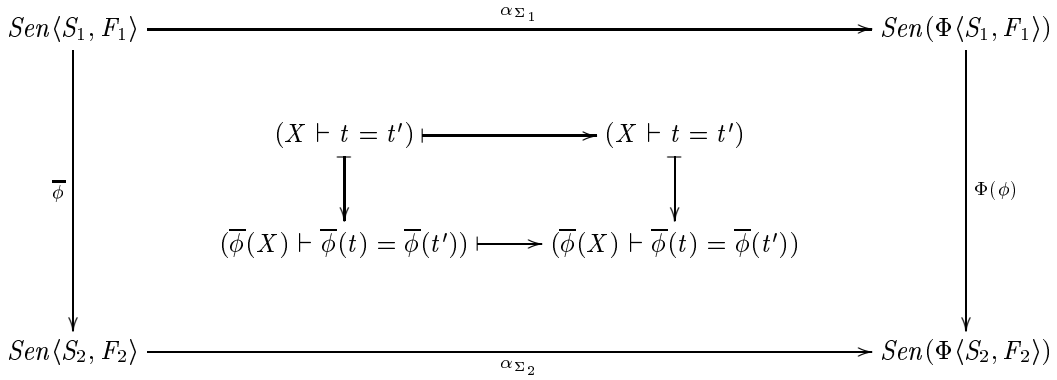



$$\begin{aligned} \Phi(\phi' \circ \phi) &= \phi' \circ \phi && \text{(definição de } \Phi) \\ &= \Phi(\phi') \circ \Phi(\phi) && \text{(definição de } \Phi) \end{aligned}$$

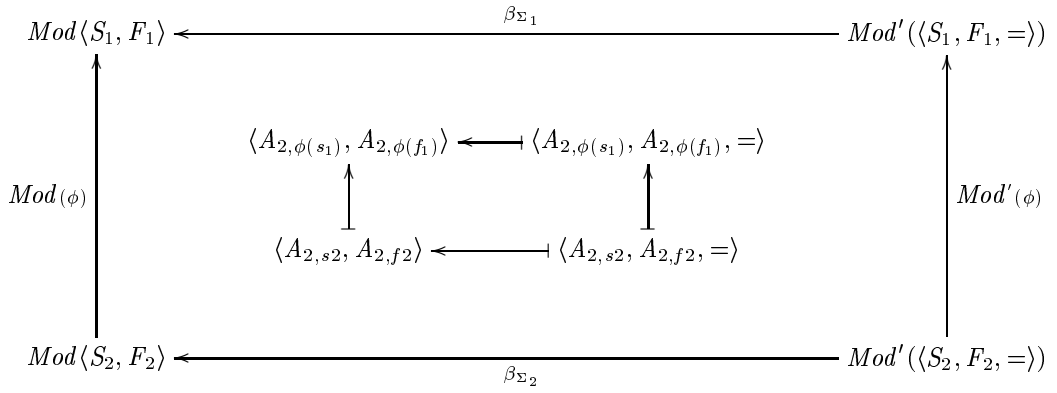
   - Identidade, ou seja, $\Phi(id_\Sigma) = id_{\Phi(\Sigma)}$, então:

$$\begin{aligned} \Phi(id_{\langle S,F \rangle}) &= \Phi(\langle S, F \rangle) && \text{(definição de } id) \\ &= \langle S, F, < \rangle && \text{(definição de } \Phi) \\ &= id_{\langle S,F,< \rangle} && \text{(definição de } id) \\ &= id_{\Phi(\langle S,F \rangle)} && \text{(definição de } \Phi) \end{aligned}$$

2. Para a transformação natural $\alpha : Sen_{MSEqtl} \Rightarrow \Phi;\ Sen_{OSEqtl} : \mathsf{Sign} \to \mathsf{Set}$, temos que, para qualquer $\phi : \langle S_1, F_1 \rangle \to \langle S_2, F_2 \rangle \in \mathsf{Sign}_{MSEqtl}$, o seguinte diagrama comuta:




3. Finalmente, para a transformação natural $\beta : \Phi^{op};\ Mod'_{OSEqtl} \Rightarrow Mod_{MSEqtl} : \mathsf{Sign}^{op}_{MSEqtl} \rightarrow \mathsf{Cat}$ definida como $\beta(\langle A(S), A(F), =\rangle) = \langle A(S), A(F)\rangle$, temos que, para qualquer $\phi : \Sigma_1 \rightarrow \Sigma_2 \in \mathsf{Sign}_{MSEqtl}$, o seguinte diagrama comuta:




A prova da condição pura é como a prova da satisfação em instituição. Contudo, as atribuições correspondentes são entre duas instituições diferentes.

Além disso, a transformação natural $\beta$ é uma coleção de setas em $\mathsf{Cat}$. Assim, temos que $\beta$ é um funtor.

**Definição 6.2 (Funtor $\beta$)** O funtor $\beta : \Phi^{op};\ Mod_{OSEqtl} \rightarrow Mod_{MSEqtl}$ é definido da seguinte maneira:

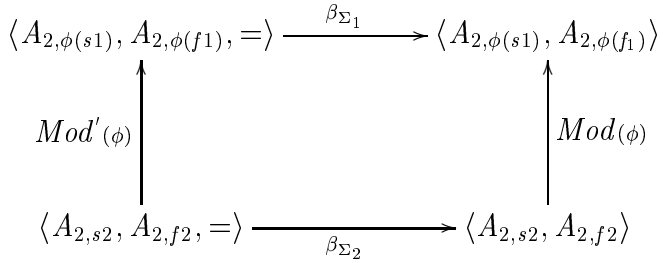
- em objetos: sejam $\Sigma = \langle S, F\rangle$ uma assinatura *MSEqtl*, $\Phi : \mathsf{Sign}_{MSEqtl} \rightarrow \mathsf{Sign}_{OSEqtl}$ um funtor, tal que $\Phi(\langle S, F\rangle) = \langle S, F, <\rangle$ e $Mod' : \mathsf{Sign}^{op}_{OSEqtl} \rightarrow \mathsf{Cat}$, então $\beta$ é definido em objetos , para todo $\langle A_{\phi(s)}, A_{\phi(f)}, =\rangle \in \mid Mod'(\langle S, F, < \rangle) \mid$, como:

$$\beta(\langle A_{\phi(s)}, A_{\phi(f)}, =\rangle) =^{def} \langle A_{\phi(s)}, A_{\phi(f)}\rangle$$

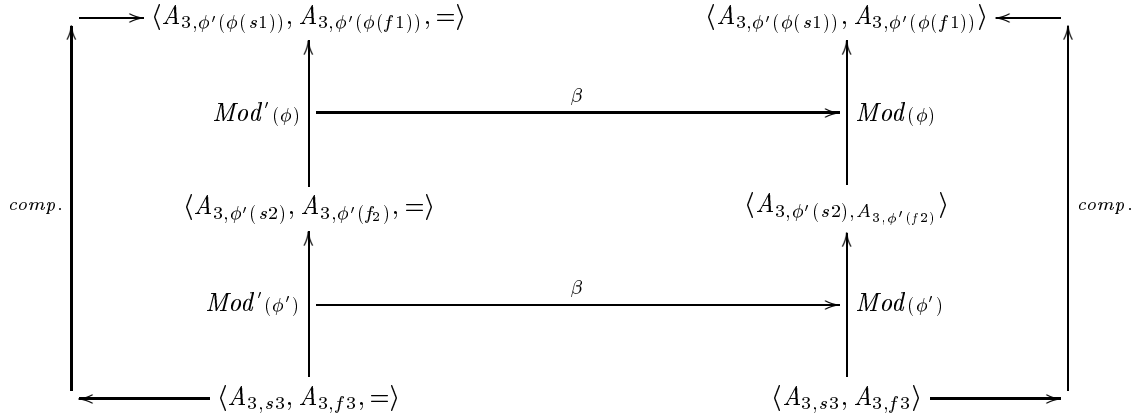
  esquecendo a relação de ordem parcial identidade nos carregadores.

- em setas: para todo $Mod'(\Phi(\phi)) : Mod'(\Phi(\Sigma_2)) \rightarrow Mod'(\Phi(\Sigma_1)) \in Mod_{OSEqtl}$ temos que $\beta(Mod'(\phi)) : \beta(Mod'(S_2, F_2, =)) \rightarrow \beta(Mod'(S_1, F_1, =)) \in Mod_{MSEqtl}$, definido como:

$$\beta(Mod'(\phi)) =^{def} Mod(\phi)$$




**Nota 6.1** Temos que verificar se o funtor preserva composição e identidade. Primeiramente para a composição, sejam $\phi : \Sigma_1 \to \Sigma_2$ e $\phi' : \Sigma_2 \to \Sigma_3$ dois morfismos em $\mathsf{Sign}_{MSEqtl}$, $Mod'(\phi) : Mod'(\Phi(\langle S_2, F_2 \rangle)) \to Mod'(\Phi(\langle S_1, F_1 \rangle))$ e $Mod'(\phi') : Mod'(\Phi(\langle S_3, F_3 \rangle)) \to Mod'(\Phi(\langle S_2, F_2 \rangle))$ temos que provar, para todo $\langle A_{3,s3}, A_{3,f3}, = \rangle \in \mid Mod_{OSEqtl}(\langle S_3, F_3, < \rangle) \mid$, $\beta(Mod'(\phi) \circ Mod'(\phi')) = \beta(Mod'(\phi)) \circ \beta(Mod'(\phi'))$. Assim, isso é fato mostrado no seguinte diagrama:




$$\begin{array}{lll} \beta(Mod'(\phi) \circ Mod'(\phi')) & = Mod(\phi) \circ Mod(\phi') & \text{(definição de } \beta) \\ & = \beta(Mod'(\phi)) \circ \beta(Mod'(\phi')) & \text{(definição de } \beta) \end{array}$$

Agora para identidade, seja $id_{Mod'_{(\Phi(\Sigma))}} : Mod'(\Phi(\Sigma)) \to Mod'(\Phi(\Sigma))$ um morfismo em $Mod'_{OSEqtl}$ . Então

$$\begin{array}{lll} \beta(id_{Mod'_{(\Phi(\Sigma))}}) & = id_{Mod_{(\Sigma)}} & \text{(definição de } \beta) \\ & = id_{\beta(Mod'_{(\Phi(\Sigma))})} & \text{(definição de } \beta) \end{array}$$

### 6.1.2 Propriedades Lógicas

Seguem nesta seção as propriedades lógicas preservadas pelo mapeamento puro.

Em aplicações dos mapeamentos, normalmente trabalhamos com noção de dedução (relação de consequência) das lógicas relacionadas. Com relação a essa idéia, o *mapeamento puro* preserva consequência semântica.

***Proposição 6.2 (Preservação de Consequência Semântica)*** *O mapeamento Puro* $(\Phi, \alpha, \beta) : \mathcal{L} \to \mathcal{L}'$ *preserva consequência semântica, i.e., para todo* $\Sigma \in |\ \mathsf{Sign}\ |$, *para todo* $\Gamma \in Sen(\Sigma)$, *e para todo* $\varphi \in Sen(\Sigma)$, *temos que*

$$\Gamma \models_\Sigma \varphi \Rightarrow \alpha(\Sigma)(\Gamma) \models'_{\Phi(\Sigma)} \alpha(\Sigma)(\varphi).$$

*Prova.* Seja $M' \in |\ Mod'_{OSEqtl}(\Phi(\Sigma))\ |$, então temos:

$$\begin{array}{lr}
M' \models'_{\Phi(\Sigma)} \alpha(\Sigma)(\Gamma) & \\
\Leftrightarrow \forall\, \varphi \in \Gamma : M' \models'_{\Phi(\Sigma)} \alpha(\Sigma)(\varphi) & \text{(def. de } \models) \\
\Leftrightarrow \forall\, \varphi \in \Gamma : \beta(\Sigma)(M') \models_\Sigma \varphi & \text{(condição pura)} \\
\Rightarrow \beta(\Sigma)(M') \models_\Sigma \varphi & \text{(assumpção)} \\
\Leftrightarrow M' \models_{\Phi(\Sigma)} \alpha(\Sigma)(\varphi) & \text{(condição pura)}
\end{array}$$

***Corolário 6.1*** *O mapeamento Puro* $(\Phi, \alpha, \beta) : \mathcal{L} \to \mathcal{L}'$ *com* $\beta$ *sobrejetor preserva e reflete consequêcia semântica, i.e., para qualquer* $\Gamma \in Sen(\Sigma)$, *e para qualquer* $\varphi \in Sen(\Sigma)$, *temos que*

$$\Gamma \models_\Sigma \varphi \Leftrightarrow \alpha(\Sigma)(\Gamma) \models'_{\Phi(\Sigma)} \alpha(\Sigma)(\varphi).$$

*Prova.* A preservação foi provada acima. Assim segue para reflexão: seja $M \in |\ Mod(\Sigma)\ |$. Pela sobrejetividade de $\beta$ existe um $M' \in |\ Mod'(\Phi(\Sigma))\ |$, tal que $\beta(\Sigma)(M') = M$. Então temos

$$\begin{array}{lr}
M \models_\Sigma \Gamma \Rightarrow \beta(\Sigma)(M') \models_\Sigma \Gamma & (\beta \text{ sobrejetor}) \\
\Leftrightarrow \forall\, \gamma \in \Gamma : \beta(\Sigma)(M') \models_\Sigma \gamma & \text{(def. de } \models) \\
\Leftrightarrow \forall\, \gamma \in \Gamma : M' \models'_{\Phi(\Sigma)} \alpha(\Sigma)(\gamma) & \text{(condição pura)} \\
\Leftrightarrow M' \models'_{\Phi(\Sigma)} \alpha(\Sigma)(\Gamma) & \text{(def. de } \models) \\
\Rightarrow M' \models'_{\Phi(\Sigma)} \alpha(\Sigma)(\varphi) & \text{(hipótese)} \\
\Leftrightarrow \beta(\Sigma)(M') \models_\Sigma \varphi & \text{(condição pura)} \\
\Rightarrow M \models_\Sigma \varphi & \text{(definição de } \beta)
\end{array}$$

Como esse trabalho considera uma lógica partindo do conceito de Instituições, para relatar a noção de sublógica temos a idéia de *subinstituição*.

**Definição 6.3 (Subinstituição)** Uma *subinstituição* é um mapeamento puro $(\Phi, \alpha, \beta) : \mathcal{L} \to \mathcal{L}'$, com $\Phi$ injetivo em objetos e $\beta$ um isomorfismo natural.

***Proposição 6.3*** *A instituição MSEqtl é uma subinstituição da OSEqtl.*

*Prova.* Precisamos verificar que o funtor $\Phi : \mathsf{Sign}_{MSEqtl} \to \mathsf{Sign}_{OSEqtl}$ é injetivo e que a transformação natural $\beta : \Phi^{op};\ Mod'_{OSEqtl} \Rightarrow Mod_{MSEqtl} : \mathsf{Sign}^{op}_{MSEqtl} \to \mathsf{Cat}$ é um isomorfismo natural.

Primeiramente, para o funtor $\Phi$ temos que verificar que ele é injetivo. Sejam $\Sigma = \langle S, F\rangle$ e $\Sigma' = \langle S', F'\rangle$ assinaturas em $|\ \mathsf{Sign}_{MSEqtl}\ |$. Como o funtor $\Phi$ não altera o objeto, a menos da ordem parcial nos sorts, se temos que $\Sigma \neq \Sigma'$ então $\Phi(\Sigma) \neq \Phi(\Sigma')$.

Seguindo, para verificar se $\beta$ é um isomorfismo, temos que provar que $\beta^{-1}_\Sigma \circ \beta_\Sigma = id_{Mod'(\Phi(\Sigma))}$ e $\beta_\Sigma \circ \beta^{-1}_\Sigma = id_{Mod(\Sigma)}$.

Seja $\beta^{-1}_\Sigma$ um funtor $\beta^{-1} : Mod_{MSEqtl} \to Mod_{OSEqtl}$ definido da seguinte maneira:

- em objetos: seja $M = \langle A_s \mid s \in S,\ A_f \mid f \in F \rangle \in \mid Mod_{MSEqtl}(\Sigma) \mid$

$$\beta^{-1}(\langle A_s \mid s \in S,\ A_f \mid f \in F \rangle) = \langle A_s \mid s \in S,\ A_f \mid f \in F, = \rangle$$

onde $=$ é a relação de ordem parcial identidade nos carregadores.

- em setas: para todo $Mod(\phi) : Mod(\Sigma_2) \to Mod(\Sigma_1) \in\ Mod_{MSEqtl}$ temos que $\beta^{-1}(Mod(\phi)) : \beta^{-1}(Mod(\Sigma_2)) \to \beta^{-1}(Mod(\Sigma_1))$ é definido como :

$$\beta^{-1}(Mod(\phi)) =^{def} Mod'(\phi)$$

Assim, segue a prova. Sejam $M' = \langle A_s \mid s \in S,\ A_f \mid f \in F, = \rangle \in \mid Mod'(\Phi(\Sigma)) \mid$ e $M = \langle A_s \mid s \in S,\ A_f \mid f \in F \rangle \in \mid Mod(\Sigma) \mid$. Primeiramente, temos:

$$\begin{array}{lll} (\beta_\Sigma^{-1} \circ \beta_\Sigma)(M') & = \beta_\Sigma^{-1}(\beta_\Sigma(M')) & \text{(composição)} \\ & = \beta_\Sigma^{-1}(M) & \text{(def. de } \beta_\Sigma) \\ & = M' & \text{(def. de } \beta_\Sigma^{-1}) \\ & = id_{Mod'(\Phi(\Sigma))}(M') & \text{(def. de } id) \end{array}$$

Segundamente, temos:

$$\begin{array}{lll} (\beta_\Sigma \circ \beta_\Sigma^{-1})(M) & = \beta_\Sigma(\beta_\Sigma^{-1}(M)) & \text{(composição)} \\ & = \beta_\Sigma(M') & \text{(def. de } \beta_\Sigma^{-1}) \\ & = M & \text{(def. de } \beta_\Sigma) \\ & = id_{Mod(\Sigma)}(M) & \text{(def. de } id_{Mod(\Sigma)}) \end{array}$$

## 6.2 Mapeamento Simples

Nesta seção apresentamos outro tipo de seta entre *instituições*, mais especificamente, *mapeamento Simples*. Este mapeamento traduz assinaturas de $\mathcal{L}$ (lógica origem ) para *teorias* de $\mathcal{L}'$ (lógica destino). Isto acontece, quanto a tradução da informação na assinatura da lógica origem requer axiomas na lógica destino, ou seja, somente a assinatura de $\mathcal{L}'$ não é suficiente para expressar toda a informação que *vem* de $\mathcal{L}$.

Como motivação para este conceito considere as especificações de listas nos esquemas que seguem.

*ThList, MSEqtl*

Sorts
$nat, list$
Opns
$empty :\to list$
$head : list \to nat$
$tail : list \to list$
$cons : nat, list \to list$

$[x : nat, l : list] head(cons(x, l)) = x : nat$
$[x : nat, l : list] tail(cons(x, l)) = l : list$
$[\ ] tail(empty) = empty : list$

*ThList*, *Horn*

Sort
  $u$
Pred
  $nat, list : u$
Opns
  $empty :\rightarrow u$
  $head : u \rightarrow u$
  $tail : u \rightarrow u$
  $cons : u, u \rightarrow u$

$[x, l : u] nat(x), list(l) \Rightarrow head(cons(x, l)) = x$
$[x, l : u] nat(x), list(l) \Rightarrow tail(cons(x, l)) = l$
$[\ ] tail(empty) = empty$

Axiomas
  $empty : u \Rightarrow list(empty)$
  $[x, l : u] nat(x), list(y) \Rightarrow list(cons(x, l))$
  $[x, l : u] nat(x), list(l) \Rightarrow nat(head(cons(x, l)))$
  $[x, l : u] nat(x), list(l) \Rightarrow list(tail(cons(x, l)))$

Observe que temos duas especificações de listas, uma na *MSEqtl* (ver seção 4.1) e a outra na lógica *Horn* (ver capítulo 5). Destacamos que na segunda especificação existem alguns axiomas adicionais. Essas sentenças estão presentes para ser possível representar toda a informação sobre listas que é representada na especificação *MSEqtl*. Assim, em um *mapeamento* entre essas duas especificações, a assinatura de listas em *MSEqtl* é relacionada com uma *teoria* (assinaturas mais axiomas) na lógica *Horn*. Isto acontece, pois na *MSEqtl* temos sorts e operações com seus respectivos sorts e esta idéia precisa de alguma maneira ser representada na lógica *Horn sem sorts*. Portanto, uma seta entre essas duas especificações traduz os símbolos de sorts para predicados na lógica *Horn* e os axiomas adicionais são utilizados para garantir o "tipo" das operações e de seus argumentos.

Semanticamente, esses predicados de tipos podem ser vistos como "filtros" que restrigem os modelos da lógica *Horn* a álgebras *MSEqtl* totais. Novamente, a tradução de modelos é dada no sentido contrário (*Horn* $\rightarrow$ *MSEqtl*), i.e., são traduzidas as álgebras que são extraídas (pelos predicados) das estruturas de interpretação *Horn*.

Um *mapeamento simples* é formalizado na seguinte

**Definição 6.4 (Mapeamento simples de instituições)** Sejam duas instituições $\mathcal{L} = (\mathsf{Sign}, Sen, Mod, \models)$ e $\mathcal{L}' = (\mathsf{Sign}', Sen', Mod', \models')$. Um *mapeamento simples* $(\Phi, \Psi, \alpha, \beta) : \mathcal{L} \rightarrow \mathcal{L}'$ consiste de:

- um funtor $\Phi : \mathsf{Sign} \rightarrow \mathsf{Sign}'$;
- um funtor $\Psi : \mathsf{Sign} \rightarrow \mathsf{Th}'$, tal que $\Psi(\Sigma) = \langle \Phi(\Sigma), \varnothing'_{\Sigma} \rangle$;
- uma transformação natural $\alpha : Sen \Rightarrow \Phi$; $Sen' : \mathsf{Sign} \rightarrow \mathsf{Set}$ e de
- uma transformação natural $\beta : \Psi^{op}$; $Mod'_{\models} \Rightarrow Mod : \mathsf{Sign}^{op} \rightarrow \mathsf{Cat}$,

tal que para cada $\Sigma \in \mathsf{Sign}$, $\varphi \in Sen(\Sigma)$ e $M' \in |Mod'_{\models}(\Psi(\Sigma))|$ a seguinte propriedade vale:

$$\beta(\Sigma)(M') \models_{\Sigma} \varphi \Leftrightarrow M' \models'_{\Phi(\Sigma)} \alpha(\Sigma)(\varphi) \quad (\textit{Condição Simples}) \qquad \square$$

$$\begin{array}{ccc} \mathsf{Sign} & & Mod(\Sigma) \xleftarrow{\models_{\Sigma}} Sen(\Sigma) \\ \Psi \downarrow & & \beta(\Sigma) \uparrow \qquad\qquad \downarrow \alpha(\Sigma) \\ \mathsf{Th}' & & Mod'_{\models}(\Psi(\Sigma)) \xleftarrow[\models'_{\Phi(\Sigma)}]{} Sen'(\Phi(\Sigma)) \end{array}$$

Na Figura 6.2 este mapeamento pode ser analisado mais informalmente. A explicação é similiar a discutida na Figura 6.1. O detalhe adicional é que agora temos a seta tracejada mais espessa representa o funtor $\Psi : \mathsf{Sign} \to \mathsf{Th}'$. Podemos observar que os modelos que são traduzidos de $\mathcal{L}'$ para $\mathcal{L}$ são os modelos das teorias que contém as informações da assinatura $\mathcal{L}$. A *condição simples* representada por $\models$ no centro do desenho significa que os modelos das teorias de $L'$ (dadas por $\psi$) satisfazem os axiomas traduzidos por $\alpha$ (em $\Sigma'$) se, e somente se, esses modelos traduzidos por $\beta$ satisfazem os axiomas da lógica origem, i.e., em $\Sigma$.

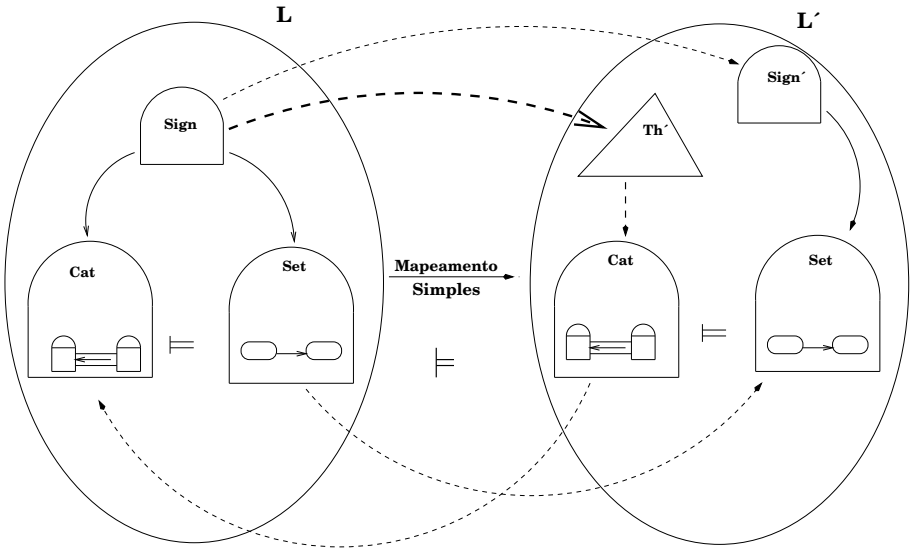



FIGURA 6.2 – Mapeamento Simples

### 6.2.1 Exemplo

Novamente, apresentaremos um mapeamento conforme a discussão apresentada no início desta seção, i.e., um mapeamento entre a *MSEqtl* e a lógica *Horn*.

**Exemplo 6.2 (Mapeamento Simples *MSEqtl* $\to$ *Horn*)** Um mapeamento simples entre essas lógicas $(\Phi, \Psi, \alpha, \beta)$ é definido da seguinte forma:

- para cada assinatura $\Sigma = \langle S, F \rangle \in |\ \mathsf{Sign}_{MSEqtl}\ |$ temos uma assinatura em $\mathsf{Sign}_{Horn}$. Tal que $\Phi\langle S, F\rangle =^{def} \langle \{u\}, F_{Horn}, P\rangle$. Note que para todo $f : s_1, \ldots, s_n \to s \in F$ associa-se um $f : u^n \to u \in F_{Horn}$, sendo que a aridade de $f \in F$ é preservada e $P = \langle p_s \mid s \in S\rangle$;

- para cada assinatura $\Sigma = \langle S, F\rangle \in |\ \mathsf{Sign}_{Horn}\ |$ temos uma teoria em $\mathsf{Th}_{\mathsf{Horn}}$, ou seja, $\Psi(\langle S, F\rangle) = \langle \Phi(\langle S, F\rangle), \varnothing'_\Sigma\rangle$, tal que

$$\varnothing'_\Sigma = \langle \forall\, x_1, \ldots, x_n : p_{s_1}(x_1), \ldots, p_{s_n}(x_n) \Rightarrow p_s(f(x_1, \ldots, x_n)) \mid f \in F\rangle$$

  Observe que os símbolos de predicados nas sentenças *Horn* garantem o tipo e a aridade dos símbolos de operações.

- para cada $\Sigma = \langle S, F\rangle \in |\ \mathsf{Sign}_{MSEqtl}\ |$ e para qualquer equação $\varphi = \langle X \vdash t = t'\rangle \in\ Sen_{MSEqtl}(\Sigma)$, onde $X$ é uma família de conjuntos de variáveis indexada por $S$, temos uma equação equivalente sem sorts $\langle \forall\, x_1, \ldots, x_n : p_{s_1}(x_1), \ldots, p_{s_n}(x_n) \Rightarrow t = t'\rangle \in\ Sen_{Horn}(\langle \{u\}, F, P\rangle)$;

- para cada $\Sigma = \langle S, F\rangle \in |\ \mathsf{Sign}_{MSEqtl}\ |$ e para qualquer estrutura de interpretação *Horn* $M = \langle M(\{u\}), M(F), M(P)\rangle$ temos uma $\Sigma$-álgebra

$$A = \langle A(S), A(F)\rangle =^{def} \langle A_s \mid s \in\ S, A_f \mid f \in\ F\rangle$$

  onde $A_s =^{def} \{m \mid M_{p_s} = m\}$ e $A_f : A_{s_1} \times \ldots \times A_{s_n} \to A_s$ é a restrição de $M_f : M \times \ldots \times M \to M$. Em geral, as $A_f$ obtidas através deste procedimento são operações parciais, pois temos que restringir à tradução aquelas $\langle \{u\}, F, P\rangle$-estruturas de interpretação $M$ que satisfazem os axiomas adicionais vindos do funtor $\Psi$, i. e.,

$$\varnothing_\Sigma = \langle \forall\, x_1, \ldots, x_n : p_{s_1}(x_1) \ldots p_{s_n}(x_n) \Rightarrow p_s(f(x_1, \ldots, x_n)) \mid f \in F\rangle$$

  Assim, temos que as álgebras $A$ são aquelas estruturas $M$ dadas pelo *funtor modelo generalizado* $Mod_{Horn,\models}(\Phi(\Sigma), \varnothing'_\Sigma)$ (ver definição 3.1).

Finalmente, a partir dessas informações temos, para todo $\varphi \in Sen_{MSEqtl}(\Sigma)$ e para todo $M' \in |\ Mod_{Horn,\models}(\Psi(\Sigma))\ |$ que:

$$\beta(\Sigma)(M') \models_{\Sigma_{MSEqtl}} \varphi \Leftrightarrow M' \models_{\Sigma_{Horn}} \alpha(\Sigma)(\varphi)$$

que é a condição simples do mapeamento.

***Proposição 6.4*** *As definições apresentadas no exemplo 6.2 definem um mapeamento Simples* $(\Phi, \Psi\alpha, \beta) : \mathcal{L}_{MSEqtl} \to \mathcal{L}_{Horn}$.

*Prova.* Esta prova é similar a da proposição 6.1.

### 6.2.2 Propriedades Lógicas

A preservação de consequência semântica para o mapemanto simples é baseada na noção de teorias. Para o mapeamento puro tínhamos a seguinte quantificação na instituição destino $\models'_{\Phi(\Sigma)}$, contudo para o simples temos $\models'_{\Psi(\Sigma)}$. Assim, precisamos introduzir uma noção de consequência semântica mais flexivel, baseada não somente em assinaturas, mas também em teorias.

**Definição 6.5 (Consequência Semântica Extendida)** Seja $\Sigma \in |\ \mathsf{Sign}\ |$, $\varphi \in Sen(\Sigma), \Delta, \Gamma \subseteq Sen(\Sigma)$. Então, $\Delta \models_{\langle \Sigma, \Gamma\rangle} \varphi =^{def} \Delta \cup \Gamma \models_\Sigma \varphi$.

***Proposição 6.5*** *Mapeamentos Simples* $(\Phi, \Psi, \alpha, \beta) : \mathcal{I} \to \mathcal{I}'$ *preservam consequência semântica, i.e., para todo* $\Sigma \in |\ \mathsf{Sign}\ |$, *para todo* $\varphi \in Sen(\Sigma)$ *e para todo* $\Gamma \subseteq Sen(\Sigma)$, *temos que:*

$$\Gamma \models_{\Sigma} \varphi \Rightarrow \alpha(\Sigma)(\Gamma) \models'_{\Psi(\Sigma)} \alpha(\Sigma)(\varphi).$$

*Prova.* Seja $\Sigma \in |\ \mathsf{Sign}\ |$, $\Psi(\Sigma) =^{def} \langle \Phi(\Sigma), \varnothing'_{\Sigma} \rangle$ e $M' \in |\ Mod'_{\models}(\Psi(\Sigma))\ |$. Pela definição temos que $\alpha(\Sigma)(\Gamma) \models'_{\Psi(\Sigma)} \alpha(\Sigma)(\Gamma) \cup \varnothing'_{\Sigma} \models'_{\Phi(\Sigma)} \alpha(\Sigma)(\varphi)$. Então, podemos concluir que:

$$\begin{array}{ll}
M' \models_{\Phi(\Sigma)} \varnothing'_{\Sigma} \cup \alpha(\Sigma)(\Gamma) \Rightarrow M' \models'_{\Phi(\Sigma)} \alpha(\Sigma)(\Gamma) & \text{(assumpção em } M') \\
\Leftrightarrow \forall\, \gamma \in \Gamma : M' \models'_{\Phi(\Sigma)} \alpha(\Sigma)(\gamma) & \text{(por definição)} \\
\Leftrightarrow \forall\, \gamma \in \Gamma : \beta(\Sigma)(M') \models_{\Sigma} \gamma & \text{(condição simples)} \\
\Leftrightarrow \beta(\Sigma)(M') \models_{\Sigma} \Gamma & \text{(por definição)} \\
\Rightarrow \beta(\Sigma)(M') \models_{\Sigma} \varphi & \text{(pela assumpção)} \\
\Leftrightarrow M' \models'_{\Phi(\Sigma)} \alpha(\Sigma)(\varphi) & \text{(condição simples)} \\
\Rightarrow M' \models'_{\Phi(\Sigma)} \varnothing'_{\Sigma} \cup \alpha(\Sigma)(\varphi) & \text{(hipótese em M')} \\
\Leftrightarrow M' \models'_{\Psi(\Sigma)} \alpha(\Sigma)(\varphi) & \text{(por definição)}
\end{array}$$

## 6.3 Notas Bibliográficas

Os mapeamentos entre instituições foram primeiramente introduzidos em [GOG 92]. Após, outros tipos de mapeamentos, mais poderosos foram apresentados no clássico *General Logics* [MES 89].

Na literatura, existem interessantes trabalhos que utilizam mapeamentos entre lógicas como fundamento matemático para implementar situações onde a integração de sistemas lógicos faz-se necessária. Cerioli & Meseguer [CER 97] buscam a conexão entre lógicas com o objetivo de utilizar (reutilizar) componentes de uma lógica em outra, por exemplo: modelos, cálculos, sistemas de provas. Outro aspecto, tratado por Martini & Wolter [MAR 99, WOL 97, MAR 97, MAR 99a], é a descrição e análise do relacionamento entre lógicas via mapeamentos. Tarlecki [TAR 96] explora esta questão, no sentido de utilizar diferentes formalismos em diferentes visões do mesmo problema. Também, Martini, Wolter & Haeusler [MAR 2001], tratam a situação de conectar módulos de especificações (com semânticas possivelmente heterogêneas) através da definição de *pontes*.

# 7 Estudo de Caso - Reutilizando um Provador de Teoremas através de Mapeamentos entre Instituições

Neste capítulo apresentamos um estudo de caso: *reutilizando um provador de teoremas através de mapeamentos entre instituições*. Esta parte da dissertação tem como objetivo utilizar mapeamentos entre instituições como abordagem para tratar um simples aspecto da *interoperabilidade lógica*, ou seja, uma situação problema em que nos deparamos com a necessidade de *reutilizar* o provador de teoremas OBJ [GOG 92a]. OBJ é uma linguagem funcional de primeira ordem que é rigorosamente baseada em *OSEqtl* e programação parametrizada, suportando um estilo declarativo que facilita verificação e possibilita o OBJ ser utilizado como um provador de teoremas.

Primeiramente introduzimos o *problema*, i.e., ao trabalharmos com especificações em *MSEqtl* surgiu a necessidade de executarmos provas (testes) sob essas especificações. Seguindo, discutimos a idéia de utilizar o provador de teoremas OBJ para executar tais provas, já que conhecemos esse sistema e o temos instalado em nossas máquinas. Apresentamos algumas especificações em *MSEqtl* junto com a execução de uma série de provas sobre essas especificações, no OBJ.

Finalmente, mostramos como um mapeamento Puro entre *MSEqtl* e *OSEqtl* é uma estrutura matemática adequada para modelar tal situação. A execução das provas na ferramenta *OSEqtl* é possível no sentido de que todos e, somente todos, os teoremas prováveis nas especificações *MSEqtl* são provados em *OSEqtl*. Temos essa garantia, pois esse mapeamento (*MSEqtl* $\rightarrow$ *OSEqtl*) possui a propriedade lógica de ser uma extensão conservativa. Também é importante salientar que para o reuso acontecer é necessário, além do mapeamento entre instituições, a existência de relações internas das lógicas (completude e coerência).

## 7.1 Especificação do Problema

Em um momento de desenvolvimento desta dissertação estávamos construindo uma série de especificações algébricas na lógica *MSEqtl*. A idéia era sobrescrever algumas propriedades funcionais (propriedades que vem da definição) em especificações e testar os efeitos dessa ação, ou seja, verificar como a reescrita de termos agia sobre certas propriedades atribuídas às especificações de operações.

Contudo, para a execução desses testes precisávamos de um provador de teoremas que implementasse um sistema de reescrita. O sistema OBJ (uma implementação de *OSEqtl* via reescrita de termos) estava a disposição, instalado em nossas máquinas, pois já o havíamos utilizado anteriormente. Então surgiu a questão: este *software* pode ser *reutilizado* com a condição de que todos e, *somente todos*, os teoremas prováveis nas especificações *MSEqtl* serão provados no OBJ (*OSEqtl*)?

O objetivo deste capítulo é demonstrar como mapeamentos entre instituições são uma estrutura matemática adequada para o tratamento dessa situação. Desta maneira, é suficiente conseguirmos um mapeamento entre as instituições *MSEqtl* e *OSEqtl* que *preserve* e *reflita* a noção de conseqüência semântica. Esta situação é ilustrada pela Figura 7.1.

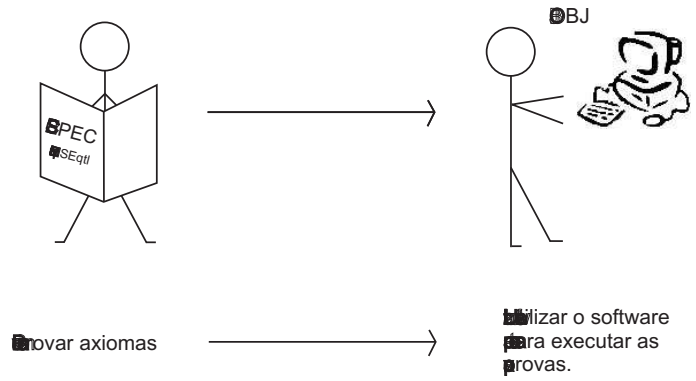



FIGURA 7.1 – Situação em questão

Como já mencionado, o provador de teoremas OBJ é uma implementação do cálculo equacional (*OSEqtl*) via *reescrita de termos*. Portanto, discutimos no texto que segue uma noção geral de como pode ser visto o *trabalho* de um sistema de reescrita. Para isso, é necessário introduzir as seguintes convenções, considerando $T(\Sigma, X)$ o conjunto de termos para uma assinatura *MSEqtl* segundo a definição 4.8. Seja a reescrita baseada na substituição de igual para igual. Então, se começamos com um termo $t_0$ (sobre uma dada assinatura $\Sigma$) e a partir dele aplicamos equações, cada passo produzirá novos termos $(t_1, t_2, \ldots, t_n)$, sendo que cada um deles é igual a todos os termos anteriores. Por exemplo, se as equações aplicadas são: $e_1, \ldots, e_n$, temos:

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

Além disso, para enterdermos a idéia de reescrita precisamos dos seguintes conceitos.

**Definição 7.1 (Regra de reescrita)** Uma equação da forma $[\,]t = t'$ (com $t, t' \in T(\Sigma, X)_s$, para algum sort $s$) é chamada regra de reescrita se e somente se, o conjunto de variáveis que acontece em $t'$ é subconjunto daquelas que ocorrem em $t$. Um conjunto finito de regras de reescrita sobre uma assinatura $\Sigma$ é chamado $\Sigma$-sistema de reescrita de termos.

**Definição 7.2 (Reescrita)** Dada uma regra de reescrita da forma $t = t'$ com $t, t' \in T(\Sigma, X)_s$ e um termo $t_0 \in T(\Sigma)_s$, sendo $z$ uma variável livre, i.e., um símbolo tal que $z \notin \Sigma \cup X$. Então, dizemos que um termo $t'_0 \in T(\Sigma)$ é um **subtermo** de $t_0$ se e somente se $t_0 = c(z \leftarrow t'_0)$ para algum termo $c \in T(\Sigma, z)$ chamado de **contexto**, e dizemos que o *match* de $e$ para $t_0$ consiste de um subtermo $t'_0$ de $t_0$, o qual é *match direto* de $e$, i.e., tal que $t'_0 = \overline{\theta}(t)$ para alguma substituição $\overline{\theta} : X \to T(\Sigma)$. Neste caso, dizemos que o termo $t_1 = c(z \leftarrow \overline{\theta}(t'))$ é o resultado de aplicar $e$ à $t_0$ no subtermo $t'_0$ utilizando a substituição $\theta$, assim podemos escrever $t_0 \Rightarrow t_1$, que é um passo de reescrita.

Por exemplo, considere a seguinte especificação dos números naturais em *MSEqtl*.

*ThNAT*, *MSEqtl*

Sorts

$nat$

Opns

$0 :\rightarrow nat \quad 1 :\rightarrow nat \quad 2 :\rightarrow nat \quad s : nat, nat \rightarrow nat$

$+ : nat, nat \rightarrow nat$

$* : nat, nat \rightarrow nat$

---

$[\ ]1 = s0 : nat \ \ (C_1) \quad [\ ]2 = s1 : nat \ \ (C_2) \quad [\ ]3 = s2 : nat \ \ (C_3)$

$[m, n : nat]m + sn = s(m + n) : nat \ \ (C_4)$

$[m : nat]m * 0 = 0 : nat \ \ (C_5)$

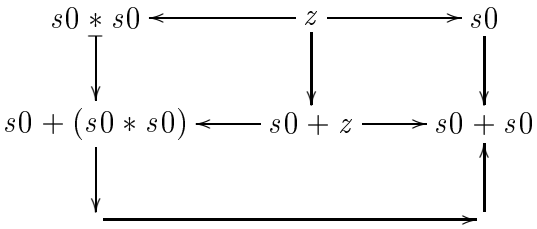
$[m : nat]0 + m = m : nat \ \ (C_6)$

$[m : nat]s0 * m = m \ \ (C_7)$

$[m, n : nat]n * sm = n * m + n : nat \ \ (C_8)$

$[m, n : nat]sm * n = n * m + n : nat \ \ (C_8')$

Seja $c$ o termo $s0 + z$, seja o $\Sigma$-sistema de variáveis $X = \{m\}$ e $\overline{\theta}(m) = s0$. Então $s0 + (s0 * s0)$ reescreve para $s0 + s0$ como resultado de aplicar a regra $[m : nat]s0 * m = m \ (C_7)$ ao subtermo $(s0 * s0)$ com a substituição $\overline{\theta}$. Isto tudo, pode ser simplificado no seguinte diagrama:

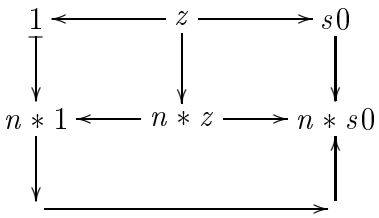



Agora, provando a equação: $n * 1 = n$ na teoria *ThNat*, através de aplicações de regras de reescrita, temos:

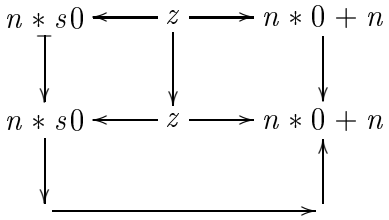
$$\begin{array}{lll} n * 1 & = n * s0 & (\text{por } C_1) \\ & = n * 0 + n & (\text{por } C_8) \\ & = 0 + n & (\text{por } C_5) \\ & = n & (\text{por } C_6) \end{array}$$

Mais detalhadamente, cada passo de reescrita é mostrado a seguir.

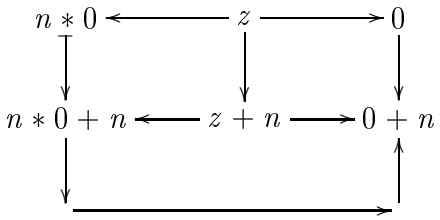
1. seja $c$ o termo $n * z$, $[\ ]1 = s0 : nat \ (C_1)$ a regra $e$, então o termo $(t_0)$ $n * 1$ reescreve para $n * s0$ como resultado de aplicar a regra ao subtermo $1 : nat$. Assim, temos o seguinte diagrama:



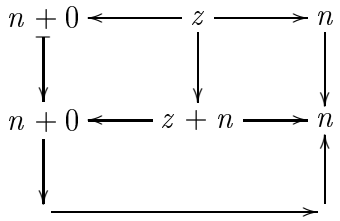
2. seja $c$ o termo $z, [n, m : nat]n * sm = n * m + n : nat$ $(C_8)$ a regra $e$ e $\overline{\theta}(m) = 0$ a atribuição, então o termo $n * s0$ reescreve para $n * 0 + n$ como resultado de aplicar a regra ao termo (subtermo) $n * s0$. Novamente, temos o diagrama a seguir:




3. seja $c$ o termo $z + n$, $[n : nat]n * 0 = 0 : nat$ $(C_5)$ a regra $e$. Então, o termo $n * 0 + n$ reescreve para $0 + n$ como resultado de aplicar a regra ao subtermo $n * 0$, i.e.,




4. seja $c$ o termo $z, [n : nat]n + 0 = n : nat$ $(C_6)$ a regra $e$. Então, o termo $n + 0$ reescreve para $n$ como resultado de aplicar a regra ao termo $n + 0$, i.e.,

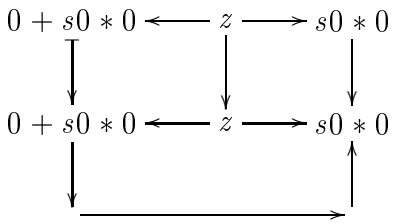



Na sequência, provamos, também com reescrita de termos, a equação $0 + (s0 * 0) = 0$ em *ThNat*. Assim,

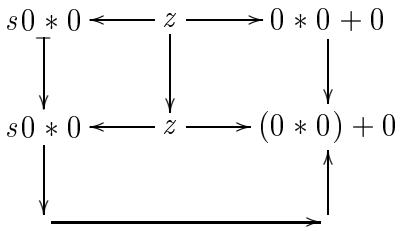
$$\begin{array}{lll} 0 + (s0 * 0) & = s0 * 0 & (\text{por } C_6) \\ & = (0 + 0) + 0 & (\text{por } C_8') \\ & = 0 + 0 & (\text{por } C_5) \\ & = 0 & (\text{por } C_6) \end{array}$$

Detalhadamente, segue cada passo de reescrita.

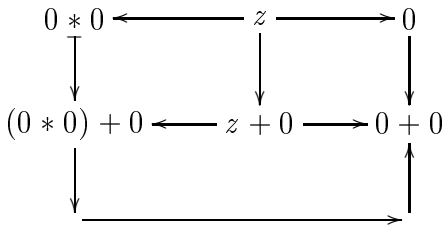
1. Seja $c$ o termo $z$, $[m : nat]0 + m = m : nat$ $(C_6)$ a regra $e$ e a substituição $\overline{\theta}(m) = s0 * 0$. Então, o termo $0 + (s0 * 0)$ reescreve para $s0 * 0$ como resultado de aplicar a regra $e$ ao termo $0 + (s0 * 0)$, i.e.,



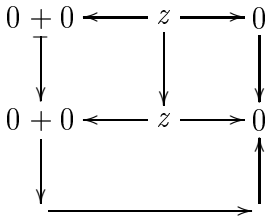
2. seja $c$ o termo $z$, $[m, n : nat]sm * n = n * m + n : nat$ $(C_8')$ a regra $e$ e as atribuições $\overline{\theta}(m) = 0$ e $\overline{\theta}(n) = 0$. Então o termo $s0 * 0$ reescreve para $(0 * 0) + 0$ como resultado de aplicar a regra $e$ ao termo $s0 * 0$, i.e.,




3. seja $c$ o termo $z + 0$, $[m : nat]m * 0 = 0 : nat$ $(C_5)$ e a atribuição $\overline{\theta}(m) = 0$. Então, o termo $(0 * 0) + 0$ reescreve para $0 + 0$ como resultado de aplicar a regra $e$ ao subtermo $(0 * 0)$, i.e.,




4. seja $c$ o termo $z$, $[m : nat]0 + m = m : nat$ $(C_6)$ a regra $e$ e a substituição $\overline{\theta}(m) = 0$. Então, o termo $0 + 0$ reescreve para $0$ como resultado de aplicar a regra $e$ ao termo $0 + 0$, i.e.,




Visto essa idéia sobre reescrita de termos podemos então discutir sobre as especificações de operações com certas propriedades atribuídas e sobre os testes que foram executados no OBJ. Para isto considere a seguinte teoria em *MSEqtl*.

*ThDATA*1, *MSEqtl*

Sorts
  *data*
Opns
  $0 :\rightarrow data$
  $*\_ : data \rightarrow data$
  $\_fun\_ : data, data \rightarrow data$

$[m, n : data]m\ fun\ n = *m : data$

Note, que temos uma especificação de um tipo de dado (*data*) com o símbolo de constante 0, com um símbolo de operação $*$, que pode ser pensado com um construtor, e com um símbolo de função *fun* que especifica uma operação que, pelo único axioma, não tem a propriedade associativa, i.e., $(0\ fun\ 0)\ fun\ 0 \neq 0\ fun\ (0\ fun\ 0)$.

Agora, considere que desejamos atribuir a propriedade associativa na especificação do símbolo *fun* acima. Assim, temos o seguinte esquema.

*ThDATA2, MSEqtl*

Sorts
*data*

Opns
$0 :\rightarrow data$
$*\_ : data \rightarrow data$
$\_fun\_ : data, data \rightarrow data$

$[m, n : data] m\ fun\ n = *m : data$
$[m, n, r : data](m\ fun\ n) fun\ r = m\ fun(n\ fun\ r)$

Onde, a associatividade é dada pelo último axioma.

Dadas essas duas teorias (*ThDATA1* e *ThDATA2*), a idéia então era testar o efeito do acréscimo do axioma da associatividade a um símbolo de operação, que pelo primeiro axioma (definição) não é associativo. Então utilizamos o provador de teoremas OBJ para executar os testes mostrados abaixo.

```
OBJ> red in DATA2 : 0 fun 0 fun 0 == 0 fun 0 fun 0 .
reduce in DATA2 : 0 fun 0 fun 0 == 0 fun 0 fun 0
********** rule
eq M fun N = * M
N:Data --> 0 fun 0
M:Data --> 0
0 fun 0 fun 0 ---> * 0
********** rule
eq M fun N = * M
N:Data --> 0 fun 0
M:Data --> 0
0 fun 0 fun 0 ---> * 0
********** rule
beq XU == YU = (obj_bool$coerce_to_bool
(term$equational_equal xu yu))
YU:Universal --> * 0
XU:Universal --> * 0
* 0 == * 0 ---> true
rewrites: 3
result Bool: true
OBJ>
```

Inicialmente, introduzimos uma redução no *prompt* do OBJ, respectiva a especificação *ThDATA2*. Na primeira linha, temos a verificação se a fórmula (0 *fun* 0 *fun* 0)

é equivalente (==) a ela mesma. Como pode ser observado, no rastro de execução, foi dado um único parser para avaliação da fórmula e ela resultou em uma verdade.

Agora, para testar a diferença de ter ou não o axioma da associatividade, executamos a mesma redução só que em *ThDATA*1, onde *fun* não possui o atributo de associatividade. Na sequência temos o teste.

```
OBJ> red in DATA1 : 0 fun 0 fun 0 == 0 fun 0 fun 0 .
Ambiguous term, two parses are:
0 fun (0 fun 0) == (0 fun 0) fun 0 -versus-
0 fun (0 fun 0) == 0 fun (0 fun 0)
differences are:
_fun_: Data Data->Data.DATA -versus- 0:->Data.DATA
in 0 fun 0 -versus-0
0:->Data.DATA -versus- _fun_: Data Data->Data.DATA
in 0 -versus-0 fun 0
Arbitrarily taking the first as correct.
reduce in DATA1 : 0 fun (0 fun 0) == (0 fun 0) fun 0
********** rule
eq M fun N = * M
N:Data --> 0 fun 0
M:Data --> 0
0 fun (0 fun 0) ---> * 0
********** rule
eq M fun N = * M
N:Data --> 0
M:Data --> 0 fun 0
(0 fun 0) fun 0 ---> * (0 fun 0)
********** rule
eq M fun N = * M
N:Data --> 0
M:Data --> 0
0 fun 0 ---> * 0
********** rule
beq XU == YU = (obj_bool$coerce_to_bool
(term$equational_equal xu yu))
YU:Universal --> * (* 0)
XU:Universal --> * 0
* 0 == * (* 0) ---> false
rewrites: 4
result Bool: false
OBJ>
```

Note, na última linha, que o resultado agora é falso. O que aconteceu foi que o sistema acusou ter dois parsers para avaliação (ambigüidade). Isto indica que para

reduzirmos uma equação em *DATA*1 é necessário colocarmos parênteses. Assim, podemos concluir que na verdade a inclusão do axioma associativo $(m\ fun\ n)fun\ r = m\ fun(n\ fun\ r)$ sintaticamente dispensa a colocação de parênteses para reduzir a equação $0\ fun\ 0\ fun\ 0$. Mais especificamente, com o atributo associativo temos que tanto faz escrevermos $0\ fun\ 0\ fun\ 0$ ou $0\ fun(0\ fun\ 0)$. Por exemplo, veja as duas seguintes reduções.

```
OBJ> red in DATA2 : 0 fun 0 fun 0 == 0 fun (0 fun 0) .
reduce in DATA2 : 0 fun 0 fun 0 == 0 fun 0 fun 0
********** rule
eq M fun N = * M
N:Data --> 0 fun 0
M:Data --> 0
0 fun 0 fun 0 ---> * 0
********** rule
eq M fun N = * M
N:Data --> 0 fun 0
M:Data --> 0
0 fun 0 fun 0 ---> * 0
********** rule
beq XU == YU = (obj_bool$coerce_to_bool
(term$equational_equal xu yu))
YU:Universal --> * 0
XU:Universal --> * 0
* 0 == * 0 ---> true
rewrites: 3
result Bool: true
OBJ>
```

Esta redução referencia o objeto *DATA*2 no qual é definda a propriedade associativa para *fun*. Observe na última linha que o resultado da redução é *true*. Entrentanto, note a diferença do mesmo teste mas agora no módulo não associativo (*ThDATA*1).

```
OBJ> red in DATA1 : 0 fun 0 fun 0 == 0 fun (0 fun 0) .
Ambiguous term, two parses are:
0 fun (0 fun 0) == 0 fun (0 fun 0) -versus-
(0 fun 0) fun 0 == 0 fun (0 fun 0)
differences are:
```

```
0:->Data.DATA1 -versus- _fun_: Data Data->Data.DATA1 in
0 -versus- 0 fun 0
_fun_: Data Data->Data.DATA1 -versus- 0:->Data.DATA1 in
0 fun 0 -versus- 0
Arbitrarily taking the first as correct.
reduce in DATA1 : 0 fun (0 fun 0) == 0 fun (0 fun 0)
********** rule
eq M fun N = * M
N:Data --> 0 fun 0
M:Data --> 0
0 fun (0 fun 0) ---> * 0
********** rule
eq M fun N = * M
N:Data --> 0 fun 0
M:Data --> 0
0 fun (0 fun 0) ---> * 0
********** rule
beq XU == YU = (obj_bool$coerce_to_bool
(term$equational_equal xu yu))
YU:Universal --> * 0
XU:Universal --> * 0
* 0 == * 0 ---> true
rewrites: 3
result Bool: true
OBJ>
```

Novamente, temos uma ambigüidade no parser de *ThDATA*1. Esta ambigüidade indica que o parser requer parênteses para definição da ordem de avaliação. Desta forma, essas duas últimas reduções temos que realmente o axioma associativo *dispensa* os parênteses para 0 *fun* 0 *fun* 0.

## 7.2 Mapeamentos & Situação Problema

Na seção anterior mostramos o uso do OBJ (*OSEqtl*) para fazer provas sob as especificações em *MSEqtl* (ver *ThDATA*1 e *ThDATA*2). No entanto, agora resta-nos saber se provamos somente o desejado e nada mais que o desejado para as teorias *MSEqtl*.

Para termos essa garantia precisamos de um *mapeamento Puro* entre as lógicas *MSEqtl* e *OSEqtl*, que pode ser verificado em detalhes na seção 6.1, que tenha a propriedade lógica de ser uma *extensão conservativa*.

Entretanto, além desse mapeamento, a situação requer uma atenção mais especial. Os mapeamentos entre instituições *conectam* duas lógicas semanticamente considerando a relação de satisfação, mas as provas (deduções) são no nível sintático considerando a relação de consequência. Por esse motivo, para o reuso do prova-

dor de teoremas OBJ acontecer precisamos de um mapeamento Puro entre *MSEqtl* e *OSEqtl* (conservativo) junto com as relações internas (completude e coerência) dessas duas lógicas.

Com esse mapeamento Puro a lógica *MSEqtl* é *codificada* na *OSEqtl*, i.e., uma maneira de representar especificações em *MSEqtl* na lógica *OSEqtl*. É interessante salientar que este mapeamento além de ser uma extensão conservativa "mostra" que a lógica das especificações é uma sublógica da lógica do OBJ (ver proposição 6.3).

Como mostrado no exemplo 6.1, um mapeamento Puro traduz, através do funtor $\alpha$, as estruturas sintáticas da lógica *MSEqtl* para a lógica *OSEqtl*. Por exemplo, considere a especificação *ThDATA*1 traduzida para *OSEqtl*.

*ThDATA*1, *MSEqtl*

Sorts
  $data$
Subsorts
  $data < data$
Opns
  $0 :\rightarrow data$
  $*\_ : data \rightarrow data$
  $\_fun\_ : data, data \rightarrow data$

$[m, n : data]m\ fun\ n = *m : data$

Note que a única diferença sintática é a introdução do símbolo $<$ representado a ordem parcial identidade no conjunto de sorts.

Como já mencionado, todos e, *somente* todos, os axiomas provados nas especificações originais devem ser possíveis de ser provados após a tradução, i.e., o mapeamento necessariamente deve ser uma *extensão conservativa*.

Garantindo que $\beta$ indexado por $\Sigma$ é um funtor e que para cada $\beta_\Sigma$ temos um $\beta_\Sigma^{-1} : Mod_{MSEqtl} \rightarrow Mod_{OSEqtl}$ tal que $\beta_\Sigma^{-1} \circ \beta_\Sigma = id_{Mod_{OSEqtl}(\Phi(\Sigma))}$ e $\beta_\Sigma \circ \beta_\Sigma^{-1} = id_{Mod_{MSEqtl}(\Sigma)}$, temos o seguinte

***Lema 7.1*** *A transformação natural $\beta$ é um isomorfismo natural.*

*Prova.* Este lema segue da prova 6.1.2.

O lema 7.1 garante que ao utilizarmos o sistema de provas OBJ provamos *todos e somente todos* os axiomas prováveis nas especificações.

Desta maneira, o mapeamento $(\Phi, \alpha, \beta) : MSEqtl \rightarrow OSEqtl$ é uma extensão conservativa, no seguinte sentido:

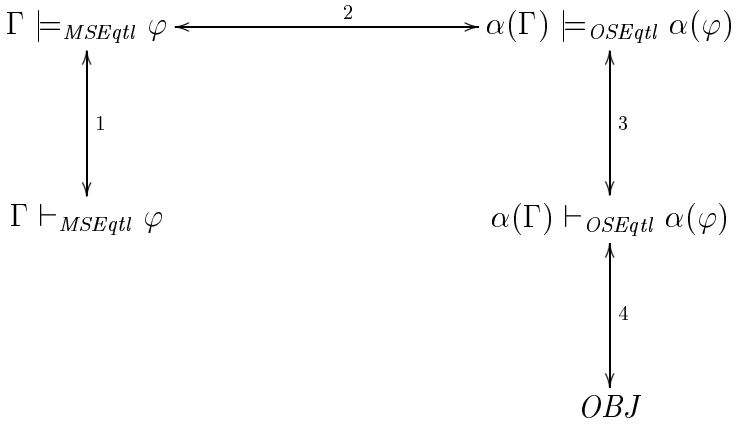
$$\Gamma \models_{\Sigma_{MSEqtl}} \varphi \Leftrightarrow \alpha(\Gamma) \models'_{\Phi(\Sigma)} \alpha(\varphi)$$

A verificação da afirmação acima, i.e, que o mapeamento puro tem a *propriedade lógica* de ser uma *extensão conservativa* pode ser analisada na prova 6.1.2.

O seguinte diagrama demostra o caminho percorrido, partindo de especificações em *MSEqtl* até as provas executadas no OBJ.

Note que temos uma situação em que a relação de consequência é *transportada* através de mapeamentos entre instituições (seta 2). Ou seja, temos um reuso de provas acontecendo através de um mapeamento junto com as relações internas das lógicas (equivalências 1 e 3). O reuso discutido aqui é a nível de provas ($\Gamma \vdash \varphi$, i.e., a

fórmula $\varphi$ é derivável de um conjunto de fórmulas $\Gamma$) não considerando a "estrutura" da prova (aplicações das regras do cálculo para construir a prova). No sentido de que diferentes cálculos podem ser utilizados, mas a relação de consequência gerada é sempre a mesma.




Mais especificamente, as extremidades $\Gamma \vdash_{MSEqtl} \varphi$ e $\alpha(\Gamma) \vdash_{OSEqtl} \alpha(\varphi)$ representam a relação de consequência (derivada de uma instituição, ver proposição 2.1) entre um conjunto de sentenças e uma sentença derivada deste conjunto (ver capítulo 1). Temos a seta número 1, pela definição de lógica *MSEqtl* ($\Gamma \vdash_\Sigma \varphi \Leftrightarrow \Gamma \models_\Sigma \varphi$, ver definição 2.7). A número 2 é a extensão conservativa descrita acima. Novamente, a partir definição de lógica *OSEqtl* temos a equivalência 3. Finalmente, a número 4 representa a relação de consequência gerada via reescrita (OBJ), que no caso é equivalente a qualquer outra gerada por qualquer outro cálculo.

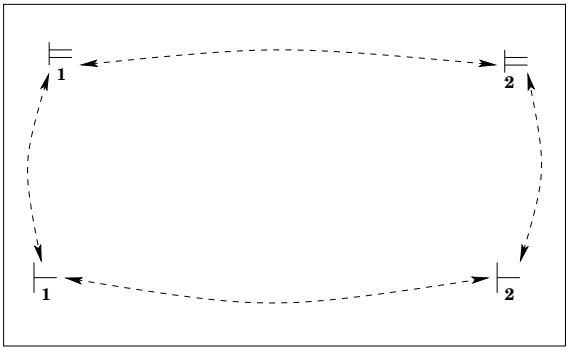



FIGURA 7.2 – Reuso de satisfação e consequência

Neste sentido, a reutilização de sistemas lógicos na componente relação de consequência, mas através de um mapeamento na componente semântica (*instituição*) Isto pode ser analisado na Figura 7.2. Podemos dizer que o mapeamento horizontal abaixo (tradução de consequência) é derivado do mapeamento horizontal acima (tradução entre instituições).

# 8 Conclusões e Trabalhos Futuros

Na área de especificação formal, devido a necessidade de capturar vários aspectos dos sistemas, precisamos trabalhar com diversos formalismos lógicos para especificação. Desta forma, o objetivo principal deste trabalho foi o de investigar fundamentos matemáticos e metodologias para tratar da integração consistente de lógicas para especificação.

Primeiramente discutimos noções rigorosas da idéia de lógica e de lógica para especificação. Neste sentido, apresentamos um esclarecimento sobre a estruturação das definições de lógica sob essas duas perspectivas. Atenção especial foi dada para utilização desses conceitos na área de especificação formal, que com a necessidade de compor sistemas requer assinaturas e morfismos entre assinaturas nas definições de lógica.

De maneira mais específica, trabalhamos com uma noção bastante geral do conceito de lógica, i.e., que engloba aspectos semânticos, dedutivos e de provabilidade. Entretanto a integração entre lógicas foi realizada através de noções de mapeamentos os quais consideram apenas os aspectos semânticos dessas mesmas lógicas.

Mostramos também, no nosso estudo de caso, reutilização de provas, que um problema prático aparentemente simples e ingênuo requer um tratamento matemático cuidadoso e ao mesmo tempo um conjunto de ferramentas (matemáticas) sofisticadas para ser solucionado.

Como trabalhos futuros propomos investigar os conceitos discutidos nesta dissertação para tratar da integração de métodos e linguagens para sistemas concorrentes e reativos. Neste contexto, temos a intenção de tratar da integração de linguagens de processos (como CCS, CSP), que são usuais para tratar aspectos de concorrência, com lógicas necessárias para especificar tipos de dados, como por exemplo lógica de primeira ordem ou lógicas utilizadas em especificação algébrica.

Outro aspecto interessante é abordar essas idéias para conexão de métodos na área de verificação formal.

É importante observar que não tratamos de uma série de outros tipos de mapeamentos entre lógicas, os quais apesar da sua importância, estão fora do escopo deste trabalho.

Finalmente, esperamos que essa dissertação sirva de motivação para que outros pesquisadores possam se interessar pelo fascinante mundo dos sistemas lógicos, de seus relacionamentos e de sua aplicação na área de ciência da computação.

# Bibliografia


[ARR 96] ARRAIS, M.; FIADEIRO, J. L. Unifying theories in different institutions. In: WORKSHOP ON SPECIFICATION OS ABSTRACT DATA TYPES, RECENT TRENDS IN TYPE SPECIFICATION, 11., 1996, Oslo, Noruega. **Proceedings...** Berlin: Springer-Verlag, 1996. p.81–101. (Lecture Notes in Computer Science, v.1130).

[AVR 89] AVRON, A.; HONSELL, F.; MASON, I. An overview of the edinburgh logical framework. In: CURRENT TRENDS IN HARDWARE VERIFICATION AND AUTOMATED THEOREM PROVING, 1989. **Proceedings...** New York: Springer-Verlag, 1989. p.323–340.

[BUR 80] BURSTALL, R. M.; GOGUEN, J. A. The semantics of Clear, a specification language. In: ABSTRACT SOFTWARE SPECIFICATION, 1980. **Proceedings...** Copenhagen: Springer-Verlag, 1980. p.292–332. (Lecture Notes in Computer Science, v.86).

[CER 97] CERIOLI, M.; MESEGUER, J. May I borrow your logic? (transporting logical structures along maps). **Theoretical Computer Science**, Amsterdam, v.173, n.2, p.311–347, 1997.

[CLA 96] CLARKE, E.; WING, J. M. Formal methods: state of the art and future directions. **ACM Computing Surveys**, New York, v.28, p.626–643, 1996.

[COU 97] COUTINHO, M. J. **Estudo categorial do relacionamento entre lógicas**. 1997. Dissertação (Mestrado em Ciência da Computação) — IST, Universidade Técnica de Lisboa, Lisboa.

[DIA 98] DIACONESCU, R. Extra-theory morphisms in institutions: logical semantics for multi-paradigm languages. **Journal Applied Categorical Structures**, [S.l.], v.6, n.4, p.427–453, 1998.

[DIA 96] DIACONESCU, R.; FUTATSUGI, K. **Logical semantics for CafeOBJ**. [S.l.]: Japan Advanced Institute of Information Science, 1996. (IS-RR-96-0024S).

[EIJ 89] EIJK, P. van; VISSERS, C.; DIAZ, M. **The formal description technique lotos**. [S.l.]: Elsevier Science Publishers B.V., 1989.

[FIA 87] FIADEIRO, J.; SERNADAS, A. Structuring theories on consequence. In: SANNELLA, D.; TARLECKI, A. (Eds.). **Recent trend in data type specification**. [S.l.]: Springer-Verlag, 1987. p.221–235. (Lecture Notes in Computer Science, v.332).

[FIS 97] FISCHER, C. Csp-oz: a combination of object-z and csp. In: IFIP TC6 WGC.1 INTERNATIONAL WORKSHOP ON FORMAL

METHODS FOR OPEN OBJECT-BASED DISTRIBUTED SYSTEMS, FMOODS, 2., 1997. **Proceedings...** London: Chapman & Hall, 1997. p.423–438.

[GOG 2001] GOGUEN, J. A. **Institutions morphisms**. [S.l.]: NASA Ames Research Center, RIACS, 2001.

[GOG 84] GOGUEN, J. A.; BURSTALL, R. M. Introducing institutions. In: LOGICS OF PROGRAMMING WORKSHOP, 1984. **Proceedings...** Carnegie Mellon: Springer-Verlag, 1984. p.221 – 256. (Lecture Notes in Computer Science, v.164).

[GOG 92] GOGUEN, J. A.; BURSTALL, R. M. Institutions: Abstract Model Theory for Specification and Programming. **Journal of the ACM**, [S.l.], v.39, n.1, p.95–146, January 1992.

[GOG 92a] GOGUEN, J. et al. **Introducing OBJ**. [S.l.]: SRI, 1992. (SRI-CSL-92-03).

[MAR 99] MARTINI, A. **Relating arrows between institutions in a categorical framework**. 1999. Tese (Doutorado em Ciência da Computação) — Tecnhical University of Berlin, Berlin.

[MAR 2001] MARTINI, A. et al. Linking logics for multiparadigm specifications. In: WORLD MULTICONFERENCE ON SYSTEMICS, CYBERNETICS AND INFORMATICS, 5., 2001. **Proceedings...** Orlando: International Institute of Informatics and Systemics, 2001. v.14, p.161–166.

[MAR 97] MARTINI, A.; WOLTER, U. A Systematic Study of Mappings Between Institutions. In INTERNATIONAL WORKSHOP ON ALGEBRAIC DEVELOPMENT TECHNIQUES, WADT, 12., 1997, Tarquinia, It. **Recent trends in algebraic development techniques**: selected papers. Berlin, Springer-Verlag, 1998. p.300-315. (Lecture Notes in Computer Science, v. 1376).

[MAR 99a] MARTINI, A.; WOLTER, U. A single perspective on arrows between institutions. In International Conference on Algebraic Methodology and Software Technology, 7., 1999, Amazônia. **Algebraic methodology and software technology**: proceedings. Berlin: Springer-Verlag, 1999. p. 486-501. (Lecture Notes in Computer Science, v. 1548).

[MES 89] MESEGUER, J. General logics. In: AL, H.-D. E. et. (Ed.). **Logic colloquium '87**. [S.l.]: Elsevier Science, 1989. p.275–329.

[MES 90] MESEGUER, J. **Rewriting logic as unified model of concurrency**. [S.l.]: SRI International, Computer Science Laboratory, 1990. (SRI–CSL–90-02).

[MIL 90] MILNER, R. **The proof theory for the raise specification language**. [S.l.]: STC Technology, 1990. (RAISE Report REM/12).

[MIL 80] MILNER, R. (Ed.). **A calculus for communicationg systems**. [S.l.]: Springer-Verlag, 1980. (Lectures Notes in Computer Science, v.92).

[PAU 2001] PAULSON, L. C. **The isabelle reference manual**. [S.l.]: University of Cambridge, 2001.

[POI 89] POIGNÉ, A. Foundations are rich institutions, but institutions are poor foundations. In: CATEGORICAL METHODS IN COMPUTER SCIENCE, 1989. **Proceedings...** [S.l.: s.n.], 1989. p.82–101.

[TAR 96] TARLECKI, A. Moving between logical systems. In: RECENT TRENDS IN DATA TYPE SPECIFICATION, 1996. **Proceedings...** Berlin: Springer-Verlag, 1996. p.478–502. (Lecture Notes in Computer Science, v.1130).

[WOL 94] WOLTER, U. et al. **Four Institutions – A Unified Presentation of Logical Systems for Specification**. Berlin: TU Berlin, Fachbereich Informatik, 1994. (Bericht-Nr. 94-24).

[WOL 97] WOLTER, U.; MARTINI, A. Shedding new light in the world of logical systems. In: INTERNATIONAL CONFERENCE ON CATEGORY THEORY AND COMPUTER SCIENCE, 7., Santa Margherita Ligure, It. **Category theory and computer science**: proccedings. Berlin, Springer-Verlag, 1997. p.159-176. (Lecture Notes in Computer Science, v. 1290).