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

RODRIGO MACHADO

Semântica Formal para TVQL

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Álvaro Freitas Moreira
Orientador

Profa. Dra. Renata de Matos Galante
Co-orientadora

Porto Alegre, setembro de 2005

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Rodrigo Machado,

Semântica Formal para TVQL /

Rodrigo Machado. – Porto Alegre: PPGC da UFRGS, 2005.

105 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2005. Orientador: Álvaro Freitas Moreira; Co-orientadora: Renata de Matos Galante.

1. Semântica Operacional. 2. Bancos de Dados Orientados a Objetos. 3. Bancos de Dados Temporais. 4. Sistemas de Tipos. 5. Linguagens de Consulta. I. Moreira, Álvaro Freitas. II. Galante, Renata de Matos. III. Título.

AGRADECIMENTOS

Gostaria de agradecer principalmente ao meu orientador, Professor Álvaro, e à minha co-orientadora, Professora Renata, pelo grande apoio, dedicação, paciência e compreensão. Além de terem sido ótimos nas questões técnicas, souberam lidar de uma forma muito legal com todos os problemas que eu acabei inventando durante este projeto. Sem eles eu definitivamente não teria conseguido nem "sair do chão". Agradeço ao Professor Tiarajú, meu orientador no início do mestrado, por todo apoio dado e por ter me ajudado a encontrar uma área de pesquisa na qual eu pretendo "fixar raízes" e construir um trabalho progressivo. Aos demais professores e funcionários do PPGC e do Instituto de Informática, que tornam este centro de pesquisa excelente, e ao CNPq, pelo suporte financeiro ao longo deste projeto.

Agradeço a Sanger, Clarissa, Diego, Everton, Ricardo, Daniel, Rúbia, Kassick, Caciano, Priscilla, Martinotto, Ennes, Fioreze e demais amigos do Instituto de Informática. Sem a amizade e a parceria que encontrei, a experiência de passar pela pós-graduação não seria tão recompensadora quanto foi. Aos "Teóricos do Mal" Kaqui e Marnes, além da amizade, também por terem me apresentado Teoria das Categorias e me acompanhado em inúmeras viagens matemáticas na sala 202. Aos amigos Gabriel, Angélica, Álvaro, Douglas, Marcelo, Toledo, Rodrigo, Edinise, por não me fazer esquecer que também existe vida fora da faculdade. Finalmente aos meus pais por todo amor e suporte, especialmente nos incontáveis momentos que eu duvidei que conseguiria concluir esta etapa.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

SGBD	Sistema Gerenciador de Banco de Dados
ODMG	*Object Database Management Group*
ODL	*Object Definition Language*
OQL	*Object Query Language*
IOQL	*Idealized Object Query Language*
TVM	*Temporal Versions Model*
TVQL	*Temporal Versioned Query Language*
SQL	*Structured Query Language*
XML	*Extensible Markup Language*
UML	*Unified Modelling Language*
S.O.	Semântica Operacional
S.T.	Sistema de Tipos
cOQL	*core Object Query Language*
cODM	*core Object Data Model*
cVOQL	*core Versioned Object Query Language*
cVODM	*core Versioned Object Data Model*
cTVQL	*core Temporal Versioned Query Language*
cTVM	*core Temporal Versions Model*

LISTA DE SÍMBOLOS

$A \times B$	Produto cartesiano de A e B
$\mathcal{P}(A)$	Conjunto das partes de A
$A \rightarrow B$	Função total entre A e B
$A \rightharpoonup B$	Função parcial entre A e B
$A \overset{fin}{\rightharpoonup} B$	Função parcial finita
$Dom(f)$	Domínio de definição da função f
$x \mapsto y$	x leva a y
$\perp$	Valor indefinido
$x \in A$	x é elemento do conjunto A
$A \subset B$	A é subconjunto próprio de B
$A \subseteq B$	A é subconjunto de B
$\vec{v}$	$v_1, \ldots, v_n \quad n \geq 0$
$\neg\phi$	ϕ negado
$\phi \wedge \psi$	ϕ e ψ
$\phi \vee \psi$	ϕ ou ψ
$\phi \longleftrightarrow \psi$	ϕ se e somente se ψ
$\phi \longrightarrow \psi$	Se ϕ então ψ

LISTA DE FIGURAS

RESUMO

Modelos de bancos de dados têm sido progressivamente estendidos a fim de melhor capturar necessidades específicas de aplicações. Bancos de dados versionados, por exemplo, provêm suporte a versões alternativas de objetos. Bancos de dados temporais, por sua vez, permitem armazenar todos os estados de uma aplicação, registrando sua evolução com o passar do tempo. Tais extensões sobre os modelos de dados se refletem nas respectivas linguagens de consulta, normalmente sob a forma de extensões a linguagens conhecidas, tais como SQL ou OQL. O modelo de banco de dados TVM (*Temporal Versions Model*), definido sobre o modelo de banco de dados orientado a objetos, suporta simultaneamente versões alternativas e o registro de alterações de objetos ao longo do tempo. A linguagem de consulta TVQL (*Temporal Versioned Query Language*), definida a partir da linguagem de consulta SQL, permite recuperar informações do modelo de dados TVM. As construções introduzidas em TVQL têm como objetivo tornar simples a consulta do banco de dados em diversos pontos da linha temporal.

Apesar das vantagens da utilização da linguagem TVQL para resgatar dados temporais do modelo TVM, existem algumas limitações importantes para seu aprimoramento. Uma delas é a alta complexidade do modelo TVM, proveniente da integração de conceitos variados como estados alternativos e rótulos temporais. Outro ponto é que, até o presente momento, não existe um interpretador para TVQL, impedindo uma experiência prática de programação de consultas.

O objetivo principal deste trabalho é o desenvolvimento de uma especificação formal para a linguagem TVQL, tornando possível um estudo consistente de suas construções. Adicionalmente, uma especificação formal serve como documentação para futuras implementações de interpretadores. Neste trabalho foi desenvolvido um protótipo de avaliador de consultas e verificador de tipos para um núcleo funcional da linguagem TVQL, possibilitando também uma experimentação prática sobre os modelos propostos.

Palavras-chave: Semântica Operacional, Bancos de Dados Orientados a Objetos, Bancos de Dados Temporais, Sistemas de Tipos, Linguagens de Consulta.

ABSTRACT

Database management systems have been constantly extended in order to achieve a better modeling of application data. For instance, versioned database models are useful to applications that need to maintain different views of a single entity, while temporal database models can automatically keep track of changes in the state of objects. These extensions often affect the respective query languages, which are usually defined over other well-known query languages such as SQL or OQL. The Temporal Versions Model (TVM) is an object-oriented database model which integrates simultaneously versioned and temporal features. The Temporal Versions Query Language (TVQL), presented as an extension of SQL, provides ways to consult the database considering temporal constraints. Despite the advantages of the use of TVQL, there are still some problems concerning the language development. Amongst them are the high complexity of the data model and the lack of a formal specification of the temporal statements.

The main purpose of this work is to provide a formal specification for the temporal statements of the TVQL language. It starts by providing an operational semantics and a type system for a subset of TVQL with support for versions. After proving that the type system is safe with respect to the semantics, the subset is extended with temporal features and an operational semantics and a type system are given.

Keywords: Operational Semantics, Object-oriented Database Management Systems, Temporal Database Management Systems, Type Systems, Query Languages.

1 INTRODUÇÃO

1.1 Motivação

Na área de banco de dados existem várias propostas que tratam de extensões dos modelos tradicionais a fim de suprir suporte a características específicas requisitadas por certas aplicações. Bancos de dados versionados, por exemplo, suportam o armazenamento e a manipulação de diferentes estados da base de dados. Bancos de dados temporais, por sua vez, têm por objetivo registrar a evolução histórica dos dados armazenados. O Modelo Temporal de Versões (TVM) (MORO, 2001) é um modelo de banco de dados orientado a objetos que possibilita representar simultaneamente estados alternativos (versões) e o registro das alterações dos dados ao longo do tempo. O Modelo Temporal de Versões surgiu dentro do Grupo de Pesquisa em Versões e Tempo em Banco de Dados do Instituto de Informática - UFRGS, onde há o interesse em estendê-lo e aprimorá-lo. Foi definida a linguagem TVQL (*Temporal and Versioned Query Language*) (MORO et al., 2002), que estabelece como são realizadas consultas sobre o modelo TVM. Entretanto, atualmente não existe nenhuma implementação ou especificação formal de TVQL, impedindo que se experimente a programação de consultas e que se investigue suas propriedades.

Este trabalho propõe a utilização de semântica operacional e de um sistema de tipos para especificar a linguagem de consulta TVQL. O uso de métodos formais reduz ambiguidades de especificações textuais e nos supre de meios para investigar e provar propriedades sobre os objetos especificados (NIELSON; NIELSON, 1992). Construções complexas de linguagens como polimorfismo de operadores, mecanismos de exceção ou mesmo cláusulas temporais são melhor compreendidas e discutidas se puderem se apoiar em um modelo matemático que descreva seu comportamento.

1.2 Objetivos

O objetivo principal deste trabalho é dar semântica formal para a linguagem TVQL. Essa especificação segue um *framework* básico, que consiste da especificação formal de um modelo de banco de dados baseado em TVM e de uma linguagem de consulta simples que capture a essência das construções de TVQL. Sobre a linguagem, objetiva-se definir um sistema de tipos e uma semântica formal. Almeja-se que o sistema de tipos seja *seguro* em relação à semântica operacional, sendo tal propriedade essencial para a linguagem e formalmente provada. Como objetivo secundário tem-se o desenvolvimento de um protótipo de um avaliador de consultas e verificador de tipos com base nas especificações desenvolvidas.

1.3 Trabalhos relacionados

A adoção de métodos formais para o estudo de linguagens de consultas de banco de dados orientados a objeto é uma abordagem relativamente recente. Em (BIERMAN, 2003), são definidos a semântica operacional e o sistema de tipos de um fragmento de OQL (denominada IOQL) com o propósito de se analisar construções que causam modificações no estado do banco de dados. São provadas propriedades essenciais da linguagem e é proposto um mecanismo de *rastreamento de efeitos* sob a forma de uma extensão do sistema de tipos.

Em (ZAMULIN, 2003), é definida semântica formal para a linguagem OQL através de uma *álgebra de objetos*. A álgebra em questão é proposta em um trabalho anterior do mesmo autor (ZAMULIN, 2002).

O trabalho apresentado em (COLAZZO et al., 2002) propõe um sistema de tipos para uma linguagem de consulta baseada em XML (W3C XML WORK GROUP, 2005), utilizando esse sistema de tipos para verificar a coerência das consultas com a estrutura definida para o esquema. Em (SIMÉON; WADLER, 2003) é apresentada uma semântica formal para XML Schema. Outro exemplo de formalização é a definição, pelo W3C, de uma semântica operacional e de um sistema de tipos para as linguagens XQuery e XPath (DRAPER et al., 2005).

No Grupo de Versões e Tempos do Instituto de informática há uma seqüência de trabalhos sobre a utilização de registro de informação temporal e gerência de versões alternativas. O Modelo de Versões (GOLENDZINER, 1995) é uma extensão ao modelo de banco de dados orientado a objetos que suporta versionamento no estado de objetos. O Modelo TF-ORM (EDELWEISS, 1994) apresenta o conceito de rótulos temporais para elementos de bancos de dados. O Modelo Temporal de Versões (MORO, 2001) propõe um modelo de dados que incorpora conceitos apresentados no modelo de versões e no modelo TF-ORM, lidando simultaneamente com versões alternativas e registro de modificações de objetos ao longo do tempo. A linguagem TVQL, apresentada em (MORO et al., 2002), propõe uma extensão de SQL para lidar com as estruturas do modelo TVM.

Os modelos que lidam com versionamento mencionados previamente o tratam especificamente no estado de objetos. Existem também trabalhos que lidam especificamente com versionamento de esquemas. TVOO (*Temporal Versioned Object Oriented*) (RODRIGUEZ; OGATA; YANO, 1999) é um modelo formal para evolução de esquemas que combina os conceitos de tempo e versão com a tecnologia de orientação a objetos. A principal característica é que novas definições para os esquemas não geram novas versões da base de dados. As definições antigas do esquema são consideradas alternativas de projeto e, conseqüentemente, o acesso aos dados existentes é feito de forma consistente sem perda de informação. OODMj (GRANDI et al., 2003) é um modelo formal para o suporte de versões temporais de esquemas em bancos de dados orientados a objetos. As definições do modelo são compatíveis com os padrões estabelecidos pelo grupo ODMG. As operações de modificação de esquemas são formalizadas através de semântica operacional, permitindo uma análise formal do comportamento de cada mudança.

Em (GALANTE, 2003) são discutidas questões relativas a modificações de esquemas de bancos de dados no modelo TVM, sendo apresentada semântica operacional para a linguagem de modificação de esquemas TVL/SE.

Este trabalho se insere tanto no contexto dos trabalhos relativos ao modelo TVM quanto no contexto da utilização de métodos formais para descrever linguagens de

consultas de bancos de dados.

1.4 Metodologia

Devido à complexidade do modelo TVM e da linguagem TVQL, escolheu-se organizar a formalização de forma incremental.

Inicialmente é apresentado um modelo formal, denominado cODM (*core Object Data Model*), para representar o esquema e o estado de um banco de dados baseado no modelo ODMG (CATTELL et al., 2000). O modelo ODMG é um padrão para bancos de dados orientados a objetos, especificando um modelo de dados, linguagem de definição de esquemas (ODL), linguagem de consulta (OQL) e várias interfaces para linguagens de programação. Sobre o modelo formal cODM é especificada a linguagem de consulta cOQL, inspirada na linguagem OQL do modelo ODMG.

Posteriormente são propostas extensões ao modelo cODM e à linguagem cOQL para suportar objetos versionados seguindo os conceitos apresentados no Modelo de Versões (GOLENDZINER, 1995), que é uma das bases de TVM. Tais extensões são denominadas, respectivamente, cVODM (*core Versioned Object Data Model*) e cVOQL (*core Versioned Object Query Language*).

Finalmente é definido o modelo cTVM (*core Temporal Versioned Model*), que adiciona registro de rótulos temporais a cVODM. O modelo cTVM captura a essência do registro de informações temporais do modelo TVM. Sobre cTVM é proposta a linguagem cTVQL, que serve como formalização das construções de TVQL.

Cada etapa, descrevendo modelo de dados e linguagem de consulta, segue a seguinte organização:

1. É apresentado informalmente o modelo de dados através de uma linguagem para especificação de esquemas e de uma descrição do estado do banco de dados.

2. A linguagem de consulta é definida através de sua sintaxe abstrata. São discutidas as construções e apresentados exemplos de consultas.

3. A avaliação de consultas é definida formalmente através de semântica operacional.

4. Define-se um sistema de tipos que categoriza as construções da linguagem de consulta.

5. É apresentada a prova de segurança do sistema de tipos, garantindo que consultas bem-tipadas não geram erros em tempo de execução.

1.5 Organização do texto

Os assuntos abordados nessa dissertação estão dispostos da seguinte maneira:

Capítulo 2: Neste capítulo é introduzido o modelo de banco de dados cODM e a linguagem de consulta cOQL. São apresentadas propriedades e considerações sobre o modelo proposto.

Capítulo 3: É apresentado o modelo cVODM e a linguagem de consulta cVOQL. O modelo cVODM estende cODM com o suporte a objetos versionados. A linguagem cVOQL inclui construções de acesso a objetos versionados presentes na linguagem TVQL.

Capítulo 4: Este capítulo define o modelo cTVM, que estende cVODM com o suporte a rótulos temporais. É definida a linguagem cTVQL, que captura a essência das construções temporais de TVQL.

Capítulo 5: São apresentadas as implementações das linguagens definidas nos capítulos anteriores.

Capítulo 6: Neste capítulo são revistas as principais contribuições deste trabalho e apresentadas conclusões e trabalhos futuros.

2 MODELO ORIENTADO A OBJETOS

Neste capítulo são definidos o modelo de dados cODM (*core Object Data Model*) e a linguagem de consulta cOQL (*core Object Query Language*), inspirados respectivamente no modelo de objetos de ODMG e na sua linguagem de consulta OQL (CATTELL et al., 2000).

2.1 Modelo cODM

O modelo de dados cODM (*core Object Data Model*) é uma simplificação do modelo de objetos do padrão ODMG. Seu foco é definir uma estrutura mínima orientada a objetos para a definição de uma linguagem de consulta.

2.1.1 Esquema de banco de dados

Esta seção apresenta informalmente o conceito de classe e o mecanismo de herança por refinamento, definindo o conceito inicial de *esquema de banco de dados cODM*.

2.1.1.1 Definição de classe

Um esquema de banco de dados consiste de um conjunto de declarações de classe. Classes determinam o formato do conjunto de objetos a ela associados. A linguagem para declaração de classes é inspirada em um fragmento de ODL (*Object Definition Language*), possuindo o formato apresentado a seguir.

$$\begin{array}{lcl} \mathit{class\_def} & ::= & \texttt{class}\ c\ \texttt{extends}\ p\ \texttt{extent}\ e \\ & & \{\ a_1 : \sigma_1,\ \ldots,\ a_n : \sigma_n\ \} \\ \sigma & ::= & \mathit{Int} \mid \mathit{Bool} \mid \mathit{String} \mid c \mid \mathit{Bag}(\sigma) \end{array}$$

A cláusula `extends` identifica a superclasse da classe definida, isto é, a classe da qual ela herdará todos os atributos. A cláusula `extent` identifica a extensão do banco de dados associada à cada classe. A extensão de uma classe é o nome dado ao conjunto de objetos persistentes de tal classe dentro do estado do banco de dados. As declarações de superclasse e de extensão são ambas obrigatórias.

Cada cláusula de $a : \sigma$ determina um nome e um tipo para um atributo. Por hora, basta identificar que os tipos podem ser básicos (Int, Bool e String), nomes de classes ou multiconjuntos (Bag) de elementos de outros tipos.

No modelo ODMG, relacionamentos entre classes são definidos sempre entre duas classes. Cada objeto de uma classe guarda no seu próprio estado referências para os

objetos da outra classe aos quais está relacionado. A única diferença entre relacionamentos e atributos é que os primeiros devem ser simétricos. Isto significa que, se um objeto guarda no seu estado uma referência a outro objeto, este outro por sua vez também deve referenciar o primeiro. A verificação de simetria dos relacionamentos faz parte da boa formação do estado de um banco de dados. Como relacionamentos utilizam os mesmos tipos e são acessados da mesma forma que atributos pela linguagem de consulta, não foram introduzidos em cODM sob a justificativa de manter simples o modelo formal.

O comportamento dos objetos está associado aos *métodos* definidos sobre estes. ODMG não faz assertivas a respeito da definição do comportamento dos objetos, estabelecendo somente a interface para a invocação destes. O código do método é especificado diretamente na linguagem de programação na qual exista um *binding* para ODMG. O modelo cODM não suporta métodos, já que não são essencialmente necessários para a modelagem pretendida. Outra característica do modelo ODMG que não está presente em cODM é a possibilidade de haver declarações de *chaves primárias*, igualmente por não serem essenciais.

Na Figura 2.1 é apresentado um exemplo de esquema composto por três classes, representando o cadastro de membros de uma instituição de ensino de pós graduação. A classe `Pessoa` define atributos básicos como nome e idade. As subclasses `Aluno` e `Professor` especializam a classe `Pessoa` com atributos específicos para cada caso (código de matrícula e titulação, respectivamente). Entre as classes `Aluno` e `Professor` existe um relacionamento, representado pelo par de atributos `Aluno.e_orientado` e `Professor.orienta`, que indica para cada aluno o seu orientador, e, para cada professor, os alunos que este orienta.

```
class Pessoa extends Object extent  Pessoas
{
   nome  : String
   idade : Int
}

class Aluno extends Pessoa extent Alunos
{
   matricula : String
   e_orientado : Professor
}

class Professor extends Pessoa extent Professores
{
   titulacao : String
   orienta : Bag(Aluno)
}
```

Figura 2.1: Exemplo de definição de esquema cODM.

2.1.1.2 Mecanismo de herança

Herança é um mecanismo que viabiliza a criação de novas classes *incrementalmente* a partir de outras já existentes. É uma técnica amplamente usada em linguagens de programação orientadas a objeto (C++, Java, Smalltalk). Nessas linguagens o conceito de *subtipo* está fortemente relacionado ao mecanismo de herança (através de cláusulas tais como *extends* e *implements*), se referindo tanto a estado (atributos) quanto a comportamento (métodos).

O mecanismo de herança do modelo cODM é simples: cada classe deve especificar uma e somente uma superclasse, da qual herdará todos os seus atributos. A classe especial *Object* está obrigatoriamente no topo da hierarquia de classes. Sua extensão é *Objects* e ela não define nenhum atributo.

2.1.2 Estado de banco de dados

Cada objeto é endereçado no banco de dados através de seu *identificador de objeto* (*oid*). O estado de cada objeto é formado por uma associação de identificadores para valores. Para exemplificar, são apresentados na Figura 2.2 dois objetos, um pertencente à classe `Aluno` e outro pertencente à classe `Professor`. Os identificadores presentes no estado dos objetos são definidos no esquema de classes, assim como os valores possíveis associados a estes.

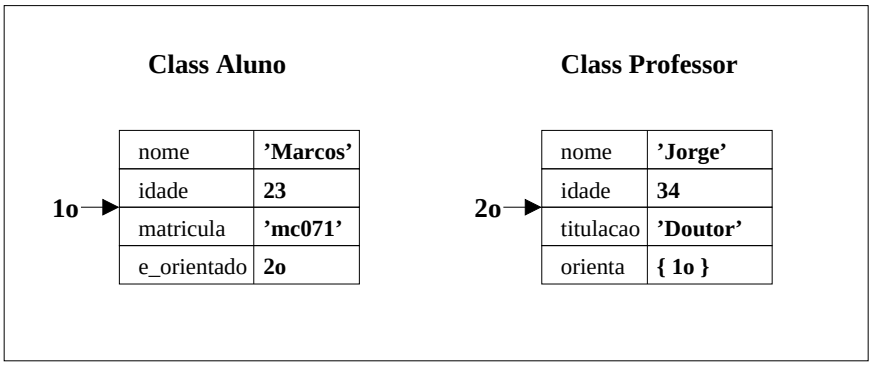

Figura 2.2: Exemplo de objetos de um banco de dados.

O estado do banco de dados consiste de um conjunto de extensões de classe, onde cada uma contém o conjunto de identificadores de objeto de sua classe. A extensão *Objects* contém todos os objetos do banco de dados, englobando também todas as outras extensões. Na Figura 2.3 é apresentado o exemplo de um estado de banco de dados contendo somente os objetos da Figura 2.2.

2.1.3 Linguagem de consulta cOQL

Esta seção apresenta informalmente a linguagem de consulta *core Object Query Language* (cOQL) para o modelo de dados cODM. A linguagem cOQL apresenta construções básicas da linguagem OQL (CATTELL et al., 2000), tendo seu sistema de tipos e semântica operacional inspirados respectivamente no sistema de tipos e semântica operacional de IOQL (BIERMAN, 2003).

A Figura 2.4 apresenta a sintaxe abstrata de consultas cOQL. Constituem-se como consultas válidas valores do banco de dados: booleanos, inteiros, *strings*, `nill`, identificadores de objetos, coleções de consultas e identificadores de extensões.

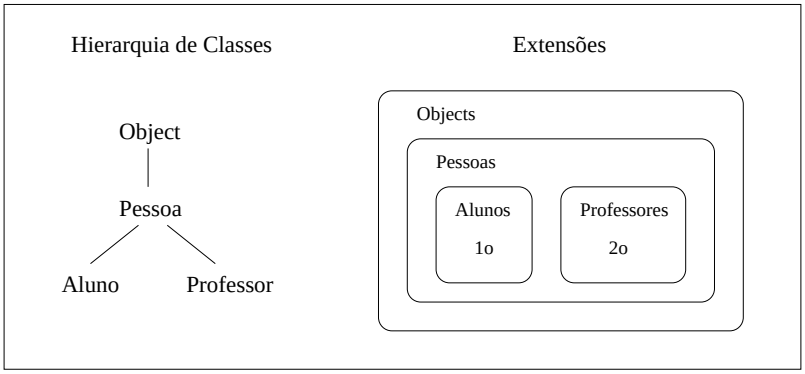

Figura 2.3: Relação entre classes e extensões.

q	$\in$	*Query*	
q	::=	b	booleanos
	|	i	inteiros
	|	s	*strings*
	|	*oid*	identificadores de objeto
	|	`nill`	valor indefinido
	|	$\{\ q_1,\ \ldots,\ q_n\ \}$	multiconjuntos de consultas, $n \geq 0$
	|	x	variável
	|	$q_1\ bop\ q_2$	operação binária
	|	$uop\ \ q_1$	operação unária
	|	$q.a$	acesso ao atributo a
	|	q`->`a	acesso a elementos do atributo a
	|	`if` q_1 `then` q_2 `else` q_3	condicional
	|	`select` q_a	
		`from` $q_1\ x_1,\ \ldots,\ q_n\ x_n$	seleção, $n \geq 0$
		`where` q_b	
bop	::=	$+ \mid - \mid *$	
	|	$< \mid <= \mid > \mid >= \mid = \mid <>$	
	|	`and` | `or`	
	|	`union` | `intersection` | `difference`	
uop	::=	`not` | $-$ | `set` | `size`	

Figura 2.4: Sintaxe abstrata de cOQL.

Os operadores binários e unários refletem as operações mais comuns sobre os dados básicos do modelo cODM, como operações inteiras (`+`, `-` e `*`), comparações entre valores inteiros (`<`, `>`, `<=`, `>=`, `=` e `<>`), operadores booleanos (`and`, `or` e `not`) e operações sobre coleções (`union`, `intersection`, `difference`). O operador `set` elimina elementos repetidos de coleções. O operador `size` retorna o número de elementos de uma dada coleção.

A linguagem cOQL possui o condicional `if-then-else` e a estrutura de seleção `select-from-where`. A presença da condição na parte `where` de uma estrutura `select-from-where` é obrigatória em cOQL. Uma consulta OQL da forma

```
select x.idade
from Professores x
```

deve ser escrita em cOQL sob a forma

```
select x.idade
from Professores x
where true
```

A cláusula `distinct`, que elimina elementos repetidos dos resultados de estruturas `select-from-where`, é obtida em cOQL através do operador unário `set`. A seguinte consulta apresenta todas as diferentes idades de objetos do tipo `Professor`:

```
set (select x.idade
    from Professores x
    where true)
```

Em OQL é utilizada a notação '.' para acessar membros de objetos tanto na parte `select` quanto na parte `from` de uma consulta. Entretanto, a semântica de cada caso é diferente. Por exemplo, considere a seguinte consulta OQL avaliada em relação a um esquema similar ao da Figura 2.1.

```
select x.orienta
from Professores x
```

Considerando o par `Aluno.e_orientado` e `Professor.orienta` como um relacionamento do modelo de dados de ODMG, `x` seria do tipo $Set(Aluno)$ e o resultado dessa consulta seria do tipo $Bag(Set(Aluno))$. A consulta retorna um multiconjunto cujos elementos são conjuntos de valores (um conjunto de alunos para cada professor). Considere agora a seguinte consulta OQL:

```
select x
from Professores.orienta x
```

Nesta consulta, o valor de retorno teria o tipo $Bag(Aluno)$, já que a variável x itera sobre todos os orientandos de todos os professores. Em OQL, a semântica do operador '.' depende da localização da expressão, sendo "acesso a membro" nas partes `select` e `where` e "iteração sobre elementos de membros do tipo coleção" na parte `from`.

Em cOQL, com o objetivo de tornar a semântica dos operadores independente de posição, se atribuiu a interpretação de '.' em `from` ao operador '`->`' e se manteve o operador '.' com a interpretação que possui na parte `select`. As consultas OQL previamente apresentadas são respectivamente escritas da seguinte forma em cOQL:

```
select x.orienta
from Professores x
where true

select x
from Professores->orienta x
where true
```

Na Figura 2.5 são apresentadas algumas consultas cOQL sobre o esquema de dados da Figura 2.1.

```
// Consulta sem acesso ao banco de dados
if 1 = 2
  then { }
  else { 'abc', 'xyz' }

// Acesso direto a extensões
Alunos

// Retorna todos os alunos
select x
from Alunos x
where true

// Seleção sem acesso ao banco de dados
select x
from { 10, 20, 30 } x, { 15, 25 } y
where x < y

// Consulta todos os diferentes nomes de orientadores cujos
// orientandos possuem menos de 25 anos
set ( select x.e_orientado.nome
      from Alunos x
      where x.idade < 25 )

// Retorna o numero de orientandos que possuem a mesma
// idade que seus orientadores
size ( select x
       from Alunos x, {x.e_orientado} y
       where x.idade = y.idade )
```

Figura 2.5: Exemplos de consultas cOQL.

2.2 Semântica formal de cOQL

Para dar semântica a cOQL foi utilizado o paradigma operacional no estilo *small-step*. Para poder utilizar o estado de banco de dados na semântica proposta, se faz necessário formalizar a estrutura de extensões e objetos.

2.2.1 Formalização de estados de banco de dados

Nesta seção é definida a formalização de um estado de banco de dados cODM. De forma similar ao modelo de dados de IOQL, define-se um estado do banco de dados através de dois elementos, um ambiente de objetos e um ambiente de extensões.

2.2.1.1 Ambiente de objetos

A seguir são apresentadas as definições necessárias para a formação de um ambiente de objetos.

$$
\begin{array}{lcl}
oid & \in & Oid \\
a & \in & Ident \\
objstate & \in & ObjState = Ident \xrightarrow{fin} Values \\
oe & \in & OE = Oid \xrightarrow{fin} (ClassId \times ObjState)
\end{array}
$$

Cada *oid* identifica um determinado objeto *unicamente* dentro do banco de dados. Idenficadores de objetos são elementos do conjunto infinito e contável *Oid*. O conjunto *Ident* representa identificadores de atributos definidos pelo usuário. O estado de um objeto é um *mapeamento finito* de *identificadores de atributos* para *valores* que podem ser atribuídos a eles. Um mapeamento finito de *oids* para pares de identificadores de classe e estados de objetos é chamado de *ambiente de objetos* (OE).

A seguir é apresentada a formação de valores.

$$
\begin{array}{lcll}
v & \in & Values & \\
v & ::= & b & \text{booleanos} \\
 & | & i & \text{inteiros} \\
 & | & s & \textit{strings} \\
 & | & oid & \text{identificadores de objeto} \\
 & | & \texttt{nill} & \text{valor indefinido} \\
 & | & \{\ v_1,\ \ldots,\ v_n\ \} & \text{multiconjuntos de valores,} \quad n \geq 0
\end{array}
$$

Valores pertencentes ao conjunto *Values* representam elementos básicos de informação, modelando quantidades, nomes, objetos e coleções. No modelo de dados cODM, valores podem ser *atômicos* ou *compostos*. Valores atômicos podem ser literais booleanos (`true` e `false`), números inteiros, cadeias finitas de caracteres (*strings*), identificadores de objeto (*oids*) ou `nill`, que representa valor indefinido. Valores compostos são multiconjuntos finitos de outros valores, sendo representados pela enumeração de seus componentes entre chaves.

2.2.1.2 Ambiente de extensões

Dentro do banco de dados, os objetos estão organizados em *extensões*. Uma extensão é uma coleção de identificadores de objeto que correspondem a todos os objetos de uma determinada classe presentes no estado do banco de dados.

Um mapeamento finito de identificadores de extensão para um conjunto finito de *oids* é denominado um *ambiente de extensões* (ee), como apresentado a seguir.

$$ee \quad \in \quad EE = ExtId \xrightarrow{fin} \mathcal{P}(Oid)$$

Define-se um *estado do banco de dados cODM* (*dbstate*) como um par de ambientes, um descrevendo extensões e outro descrevendo objetos, como apresentado a seguir.

$$dbstate \in DBState = EE \times OE$$

2.2.2 Relação de redução

Conhecida a estrutura do estado do banco de dados, pode-se definir a relação de redução entre consultas. A redução da consulta q para q' sob o estado (ee, oe) é denotada por $ee; oe \vdash q \rightarrow q'$. A relação de redução entre consultas é definida na Figura 2.6.

Para reduzir o número de regras da semântica operacional *small-step* de cOQL, são utilizados *contextos de avaliação* ($\mathcal{E}$), formados conforme a gramática apresentada a seguir.

$$\begin{aligned}\mathcal{E} ::= \; & \bullet \mid \{\vec{v}, \mathcal{E}, \vec{q}\} \mid \mathcal{E} \; bop \; q \mid v \; bop \; \mathcal{E} \mid uop \; \mathcal{E} \mid \mathcal{E}.a \mid \mathcal{E}\texttt{->}a \mid \\ & \texttt{if } \mathcal{E} \texttt{ then } q_1 \texttt{ else } q_2 \mid \texttt{select } q_a \texttt{ from } \mathcal{E} \; x_1, \ldots, q_n \; x_n \texttt{ where } q_b \mid \\ & \texttt{select } q \texttt{ from where } \mathcal{E}\end{aligned}$$

Um contexto de avaliação nada mais é do que uma consulta com um "buraco" ($\bullet$) que indica a posição sintática da próxima subconsulta a ser avaliada. A aplicação de uma dada consulta q a um dado contexto $\mathcal{E}$ é denotada $\mathcal{E}[q]$ e consiste da substituição da ocorrência de $\bullet$ por q, como apontam os seguintes exemplos:

$$(\texttt{if} \bullet \texttt{then } q_1 \texttt{ else } q_2)[q] = \texttt{if } q \texttt{ then } q_1 \texttt{ else } q_2$$

$$(5 + \bullet .idade)[oid] = 5 + oid.idade$$

No que segue, até o final desta subseção, serão exemplicadas as regras apresentadas na Figura 2.6.

A regra S-Ctx conduz a avaliação de subconsultas até se chegar em valores. Ela utiliza o contexto de avaliação para determinar qual das subconsultas deve ser avaliada em cada caso.

A regra S-Var reduz uma dada variável representando uma extensão para a coleção de identificadores de objeto correspondente através do ambiente de extensões.

$$\frac{ee; oe \vdash q \rightarrow q'}{ee; oe \vdash \mathcal{E}[q] \rightarrow \mathcal{E}[q']} \quad \text{(S-Ctx)}$$

$$\frac{ee(x) = \{\overrightarrow{oid}\}}{ee; oe \vdash x \rightarrow \{\overrightarrow{oid}\}} \quad \text{(S-Var)}$$

$$\frac{\eta_b(bop, v_1, v_2) = v_3}{ee; oe \vdash v_1\ bop\ v_2 \rightarrow v_3} \quad \text{(S-Binop)}$$

$$\frac{\eta_u(uop, v_1) = v_2}{ee; oe \vdash uop\ v_1 \rightarrow v_2} \quad \text{(S-Unop)}$$

$$\frac{oe(oid) = (c, objstate) \qquad objstate(a) = v}{ee; oe \vdash oid.a \rightarrow v} \quad \text{(S-Dot1)}$$

$$ee; oe \vdash \texttt{nill}.a \rightarrow \texttt{nill} \quad \text{(S-Dot2)}$$

$$ee; oe \vdash \{oid, \vec{v}\}\texttt{->}a \rightarrow oid.a\ \texttt{union}\ \{\vec{v}\}\texttt{->}a \quad \text{(S-Arrow1)}$$

$$ee; oe \vdash \{\texttt{nill}, \vec{v}\}\texttt{->}a \rightarrow \{\vec{v}\}\texttt{->}a \quad \text{(S-Arrow2)}$$

$$ee; oe \vdash \{\}\texttt{->}a \rightarrow \{\} \quad \text{(S-Arrow3)}$$

$$ee; oe \vdash \texttt{nill->}a \rightarrow \texttt{nill} \quad \text{(S-Arrow4)}$$

$$ee; oe \vdash \texttt{if true then}\ q_1\ \texttt{else}\ q_2 \rightarrow q_1 \quad \text{(S-If1)}$$

$$ee; oe \vdash \texttt{if false then}\ q_1\ \texttt{else}\ q_2 \rightarrow q_2 \quad \text{(S-If2)}$$

$$ee; oe \vdash \texttt{if nill then}\ q_1\ \texttt{else}\ q_2 \rightarrow \texttt{nill} \quad \text{(S-If3)}$$

$$\begin{array}{c} ee; oe \vdash \texttt{select}\ q_a\ \texttt{from}\ \{v_1, \ldots, v_m\}\ x_1, \ldots, q_n\ x_n\ \texttt{where}\ q_b \rightarrow \\ (\texttt{select}\ q_a\ \texttt{from}\ q_2\ x_2, \ldots, q_n\ x_n\ \texttt{where}\ q_b)[x_1 ::= v_1] \\ \texttt{union} \\ \texttt{select}\ q_a\ \texttt{from}\ \{v_2, \ldots, v_m\}\ x_1, \ldots, q_n\ x_n\ \texttt{where}\ q_b \end{array} \quad \text{(S-Select1)}$$

$$ee; oe \vdash \texttt{select}\ q_a\ \texttt{from}\ \{\}\ x_1, \ldots, q_m\ x_n\ \texttt{where}\ q_b \rightarrow \{\} \quad \text{(S-Select2)}$$

$$ee; oe \vdash \texttt{select}\ q_a\ \texttt{from nill}\ x_1, \ldots, q_m\ x_n\ \texttt{where}\ q_b \rightarrow \texttt{nill} \quad \text{(S-Select3)}$$

$$ee; oe \vdash \texttt{select}\ q_a\ \texttt{from where true} \rightarrow \{q_a\} \quad \text{(S-Select4)}$$

$$ee; oe \vdash \texttt{select}\ q_a\ \texttt{from where false} \rightarrow \{\} \quad \text{(S-Select5)}$$

$$ee; oe \vdash \texttt{select}\ q_a\ \texttt{from where nill} \rightarrow \{\} \quad \text{(S-Select6)}$$

Figura 2.6: Semântica operacional de cOQL.

A regra S-BINOP se baseia na aplicação da função auxiliar η_b para obtenção do resultado de operações binárias sobre valores. A função η_b estabelece funções comuns sobre inteiros e valores lógicos. Entretanto, os operadores `union`, `intersection` e `difference` possuem interpretação de multiconjunto, isto é, preservam repetição de elementos. Os exemplos a seguir ilustram esse efeito.

$$\begin{array}{lcll} \{1,2\} & \texttt{union} & \{1,3\} & = \{1,1,2,3\} \\ \{1,1,1,3,4\} & \texttt{intersection} & \{1,1,2,4\} & = \{1,1,4\} \\ \{1,1,1,3,4\} & \texttt{difference} & \{1,2,3,4,4\} & = \{1,1\} \end{array}$$

Para garantir a semântica tradicional de conjunto em `union`, `intersection` e `difference`, basta aplicar a operação unária `set` sobre os operandos e sobre o resultado de cada operador. Operações binárias retornam sempre `nill` quando no mínimo um de seus argumentos estiver indefinido, como apresentado a seguir.

$$\begin{array}{lcll} \texttt{nill} & \texttt{+} & \texttt{nill} & = \texttt{nill} \\ \texttt{nill} & \texttt{and} & \texttt{true} & = \texttt{nill} \\ \texttt{\{4,2\}} & \texttt{union} & \texttt{nill} & = \texttt{nill} \end{array}$$

De forma similar à regra S-BINOP, a regra S-UNOP utiliza a função auxiliar η_u para determinar o resultado da aplicação de funções unárias sobre valores. O tratamento para uma operação unária sobre `nill` sempre retorna `nill`.

$$\begin{array}{lll} \texttt{neg} & \texttt{true} & = \texttt{false} \\ \texttt{set} & \texttt{\{1,1,3,3,4\}} & = \texttt{\{1,3,4\}} \\ \textit{uop} & \texttt{nill} & = \texttt{nill} \end{array}$$

A regra S-DOT1 acessa o valor de membros de objeto através do ambiente de objetos. Referências a membros de `nill` resultam em `nill` por S-DOT2.

A avaliação de membros dentro de coleções é conduzida pelas regras S-ARROW1, S-ARROW2 e S-ARROW3, que aplicam o operador '.' sobre todos os valores definidos da coleção e resulta em uma coleção correspondente à união dos resultados. Se a coleção estiver indefinida, a avaliação resulta em `nill` por S-ARROW4.

Condicionais são avaliados por S-IF1 no caso de condição verdadeira, por S-IF2 no caso de condição falsa e por S-IF3 no caso de condição indefinida.

A semântica da avaliação de construções `select-from-where` segue o sentido de SQL. No modelo relacional, é calculado o produto cartesiano de todas as tabelas relacionadas na parte `from`. Sobre as tuplas desse produto cartesiano é executada a operação de seleção, que determina o subconjunto de tuplas que respeitam as condições da parte `where`. Finalmente, são selecionados somente os campos indicados na parte `select` através da operação de projeção.

Na linguagem cOQL, o produto cartesiano é simulado pela geração de todas as combinações possíveis de valores de variáveis declaradas na parte `from`. Ao longo da avaliação, as regras S-SELECT1, S-SELECT2 e S-SELECT3, através de substituição (Figura 2.7), geram todas as combinações possíveis de valores para todas as variáveis declaradas. A regra S-SELECT4 inclui na coleção de resposta os casos onde a condição da parte `where` é verdadeira, enquanto as regras S-SELECT5 e S-SELECT6 eliminam da resposta os casos em que a condição é falsa ou indefinida, respectivamente.

Nota-se que o resultado da avalição de uma consulta `select-from-where` é sempre a coleção vazia se uma das variáveis itera sobre a coleção vazia. Por exemplo, considere a seguinte consulta:

$i[x ::= v] = i$

$b[x ::= v] = b$

$s[x ::= v] = s$

$oid[x ::= v] = oid$

$\texttt{nill}[x ::= v] = \texttt{nill}$

$x[x ::= v] = v$

$x'[x ::= v] = x' \quad \text{se } x' \neq x$

$\{q_1, \ldots, q_n\}[x ::= v] = \{q_1[x ::= v], \ldots, q_n[x ::= v]\}$

$(q_1\ bop\ q_2)[x ::= v] = q_1[x ::= v]\ bop\ q_2[x ::= v]$

$(uop\ q_1)[x ::= v] = uop\ (q_1[x ::= v])$

$(q.a)[x ::= v] = q[x ::= v].a$

$(q\texttt{->}a)[x ::= v] = q[x ::= v]\texttt{->}a$

$(\texttt{if}\ q\ \texttt{then}\ q_1\ \texttt{else}\ q_2)[x ::= v] = \texttt{if}\ q[x ::= v]\ \texttt{then}\ q_1[x ::= v]\ \texttt{else}\ q_2[x ::= v]$

$$\begin{array}{lcl} (\texttt{select}\ q_a & & \texttt{select}\ q_a[x ::= v] \\ \texttt{from}\ q_1\ x_1, \ldots, q_n\ x_n & = & \texttt{from}\ q_1[x ::= v]\ x_1, \ldots, q_n[x ::= v]\ x_n \\ \texttt{where}\ q_b)[x ::= v] & & \texttt{where}\ q_b[x ::= v] \end{array}$$

Figura 2.7: Definição da operação de substituição.

```
select 5
from { 12, 42 } x, {} y
where true
```

Apesar de não haver nenhuma referência a `y` na parte `select` da consulta, pela semântica apresentada a consulta como um todo avalia para {}. Isso se dá devido à incapacidade de se *montar* uma configuração de variáveis que possua tanto `x` quanto `y`.

Outra característica dessa semântica é que, ao se declarar uma variável, esta fica disponível para a avaliação do conjunto de valores da variável subseqüente. Observe a seguinte consulta:

```
select x
from Professores x, x.orienta y
where x.idade < y.idade
```

É utilizada a variável `x`, primeira variável declarada da parte `from`, na avaliação do resultado do conjunto sobre o qual `y` irá iterar. Essa dependência entre variáveis estabelece uma ordem específica para a avaliação das coleções sobre as quais as variáveis iterarão.

Como visto, devido à existência de representação explícita de indefinição, se fez necessário estabelecer um tratamento para o caso de uma ou mais subconsultas resultarem em `nill`. Uma interpretação natural é que, nesses casos, a consulta como um todo seja indefinida (comportamento *estrito*). As regras S-Binop, S-Unop, S-Dot2, S-Arrow4 e S-If3 e S-Select3 seguem essa linha, reduzindo para a `nill` caso um de seus componentes seja indefinido. Entretanto, há alguns casos onde a avaliação passa por cima de `nill` e apresenta resultados definidos. É o caso da regras S-Arrow2 e S-Select6. O objetivo principal de tais construções é obter uma semântica mais próxima da construção `select-from-where` de SQL, que retorna um conjunto vazio de tuplas caso a expressão da parte `where` não seja verdadeira.

2.3 Sistema de tipos de cOQL

Uma das atribuições de um sistema de tipos é a eliminação de programas que levariam a situações de erro antes de sua avaliação através de uma categorização das construções de uma determinada linguagem (PIERCE, 2002). Sistemas de tipos podem ser dinâmicos, quando existe verificação de tipos durante a execução do programa, ou estáticos, quando a análise de tipos é realizada em tempo de compilação. O sistema de tipos proposto para cOQL é estático, sendo a linguagem de tipos a mesma usada na definição de esquema.

$$\begin{array}{lll} \sigma & \in & Type \\ \sigma & ::= & Int \mid Bool \mid String \mid c \mid Bag(\sigma) \end{array}$$

Em um sistema de banco de dados orientado a objetos, os tipos dos objetos definidos pelo usuário administrador do SGBD estão compreendidos no *esquema do banco de dados*, cuja formalização é apresentada a seguir.

2.3.1 Formalização de esquemas de banco de dados

Formalmente, um esquema de banco de dados cODM (ou esquema de classes) consiste de um mapeamento de identificadores de classe para definições de classe, como mostrado a seguir.

$$\begin{array}{lcll} p & \in & ClassId_{\perp} & = & ClassId \cup \{\perp\} \\ sch & \in & Schema & = & ClassId \stackrel{fin}{\rightharpoonup} Classdef \\ & & Classdef & = & ClassId_{\perp} \times ExtentId \times Ty \\ ty & \in & Ty & = & Ident \stackrel{fin}{\rightharpoonup} Type \end{array}$$

O formato dos objetos matemáticos propõe uma tradução quase direta das declarações sintáticas da linguagem de declaração de esquemas apresentada na Seção 2.1.1. Para cada identificador de classe c definido no esquema, associa-se uma tripla contendo o identificador da superclasse, o identificador de extensão, e uma função que associa tipos a atributos. Esta função define os tipos de *todos* os atributos da classe, incluindo os especificados em superclasses.

Em um esquema de classes bem-formado, *Object* é a única classe que não possui superclasse *definida*, tornando necessário que o conjunto de identificadores $ClassId_{\perp}$ possibilite tal indefinição. A seguir é apresentada a definição de *Object*, que não segue a sintaxe apresentada para a definição de esquemas.

```
class Object extent Objects { }
```

A classe *Object* está presente em todos os esquemas de classe, não podendo ser redefinida.

2.3.2 Julgamentos de tipos

Sistemas de tipos são normalmente apresentados sob a forma de sistemas de dedução formal, que, sob um dado contexto, associam tipos a consultas. Um julgamento de tipos para uma consulta cOQL q é da forma

$$sch; ee; oe; \Gamma \vdash q : \sigma$$

onde

- *sch* é um *esquema de dados* pertencente ao conjunto *Schema* definido na Seção 2.3.1;
- *ee* é um ambiente de extensões pertencente ao conjunto EE definido na Seção 2.2.1.2;
- *oe* é um ambiente de objetos pertencente ao conjunto OE definido na Seção 2.2.1.1;
- Γ é um ambiente de tipos.

Um ambiente de tipos, denotado por Γ, é um mapeamento de um conjunto finito de variáveis $x \in Ident$ para tipos $\sigma \in Types$.

$$\Gamma \quad \in \quad TypeEnv = Ident \xrightarrow{fin} Type$$

Se denota por $\emptyset$ o ambiente de tipos vazio, e por $\Gamma, x : \sigma$ o ambiente de tipos formado pela adição da associação $x : \sigma$ ao ambiente de tipos Γ. Essa adição pode representar tanto a expansão do ambiente de tipos Γ, se $x \notin Dom(\Gamma)$, quanto a redefinição de x em Γ, se $x \in Dom(\Gamma)$.

Uma consulta q é do tipo σ, sob o contexto $sch; ee; oe; \Gamma$, se e somente se existir uma árvore de derivação a partir das regras apresentadas na Figura 2.8 que prove $sch; ee; oe; \Gamma \vdash q : \sigma$. Nas regras, por clareza de notação, um contexto $sch; ee; oe; \Gamma$ é representado pelo símbolo K.

A Figura 2.8 contém o conjunto de regras de tipos para cOQL. No que segue, até o final da subseção, cada uma das regras será explicada.

As regras T-Bool, T-Int e T-Str são axiomas e atribuem tipos a literais. A regra T-Oid atribui a um identificador de objetos o tipo c de acordo com a classe armazenada no ambiente de objetos. A regra T-Nill associa um tipo qualquer σ do sistema de tipos ao valor `nill`.

A regra T-Bag associa o tipo $Bag(\sigma)$ a qualquer coleção cujos elementos são do tipo σ. A coleção vazia, por não conter elementos, é de qualquer tipo $Bag(\sigma)$.

Variáveis têm seu tipo determinado pela regra T-Var através do ambiente de tipos Γ, podendo ser identificadores de extensões ou então variáveis declaradas na cláusula `from` de uma estrutura `select-from-where`. Na árvore de derivação, há o incremento do ambiente de tipos nas premissas da regra T-Select, que define o tipo de variáveis *ligadas* dentro de cláusulas `select-from-where`.

Os conceitos de variável livre e variável ligada vêm de cálculo-λ (BARENDREGT, 1990). Uma ocorrência de uma variável x em uma consulta cOQL é dita *ligada* se aparecer na parte `select` ou na parte `where` de uma estrutura `select-from-where` e tiver sido declarada na cláusula `from`. A ocorrência de uma variável é também ligada se aparecer em uma expressão na parte `from` da consulta após ter sido associada à uma variável na lista de declarações. Por exemplo, a ocorrência da variável `x` é ligada na expressão `x->e_orientado` da seguinte consulta:

```
select x
from Alunos x, x->e_orientado y
where x.idade > y.idade
```

Uma ocorrência de uma variável é dita *livre* caso não seja ligada. O conjunto de variáveis livres de uma consulta q é denotado $FV(q)$. Variáves ligadas representam iteradores sobre elementos de coleções, enquanto variáveis livres representam as extensões do banco de dados. Por exemplo, considere a seguinte consulta que retorna todos os nomes de professores cuja idade é maior que o número de alunos.

```
select x.nome
from Professores x
where x.idade > size Alunos
```

$$K \vdash b : Bool \qquad \text{(T-Bool)}$$

$$K \vdash i : Int \qquad \text{(T-Int)}$$

$$K \vdash s : String \qquad \text{(T-Str)}$$

$$\frac{oe(oid) = (c, objstate)}{K \vdash oid : c} \qquad \text{(T-Oid)}$$

$$K \vdash \texttt{nill} : \sigma \qquad \text{(T-Nill)}$$

$$\frac{K \vdash q_1 : \sigma \quad \ldots \quad K \vdash q_n : \sigma \qquad n \geq 0}{K \vdash \{q_1, \ldots, q_n\} : Bag(\sigma)} \qquad \text{(T-Bag)}$$

$$\frac{\Gamma(x) = \sigma}{sch; ee; oe; \Gamma \vdash x : \sigma} \qquad \text{(T-Var)}$$

$$\frac{K \vdash q_1 : \sigma_1 \qquad K \vdash q_2 : \sigma_2 \qquad \delta_b(bop, \sigma_1, \sigma_2) = \sigma_3}{K \vdash q_1 \ bop \ q_2 : \sigma_3} \qquad \text{(T-Binop)}$$

$$\frac{K \vdash q : \sigma_1 \qquad \delta_u(uop, \sigma_1) = \sigma_2}{K \vdash uop \ q : \sigma_2} \qquad \text{(T-Unop)}$$

$$\frac{K \vdash q : c \qquad sch(c) = (p, e, ty) \qquad ty(a) = \sigma}{K \vdash q.a : \sigma} \qquad \text{(T-Dot)}$$

$$\frac{K \vdash q : Bag(c) \qquad sch(c) = (p, e, ty) \qquad ty(a) = Bag(\sigma)}{K \vdash q\texttt{->}a : Bag(\sigma)} \qquad \text{(T-Arrow)}$$

$$\frac{K \vdash q_1 : Bool \qquad K \vdash q_2 : \sigma \qquad K \vdash q_3 : \sigma}{K \vdash \texttt{if } q_1 \texttt{ then } q_2 \texttt{ else } q_3 : \sigma} \qquad \text{(T-If)}$$

$$\frac{\begin{array}{rcl} sch; ee; oe; \Gamma & \vdash & q_1 : Bag(\sigma_1) \\ sch; ee; oe; \Gamma, x_1 : \sigma_1 & \vdash & q_2 : Bag(\sigma_2) \\ & \vdots & \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_{n-1} : \sigma_{n-1} & \vdash & q_n : Bag(\sigma_n) \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n & \vdash & q_a : \sigma_a \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n & \vdash & q_b : Bool \end{array} \quad x_1 \text{ a } x_n \text{ diferentes} \qquad x_1, \ldots, x_n \notin Dom(\Gamma) \qquad n \geq 0}{sch; ee; oe; \Gamma \vdash \texttt{select } q_a \texttt{ from } q_1 \ x_1 \ , \ldots, q_n \ x_n \texttt{ where } q_b : Bag(\sigma_a)} \qquad \text{(T-Select)}$$

$$\frac{K \vdash q : \sigma_1 \qquad \sigma_1 <: \sigma_2}{K \vdash q : \sigma_2} \qquad \text{(T-Sub)}$$

Figura 2.8: Sistema de tipos de cOQL.

Os termos `x`, `Professores` e `Alunos` são variáveis. Entretanto, `x` aparece ligada tanto na subconsulta `x.nome` quanto na subconsulta `x.idade > size Alunos`. As variáveis `Alunos` e `Professores` aparecem livres e, portanto, representam extensões do banco de dados.

Uma situação que pode ocorrer em consultas OQL é o conflito de nomes quando se declaram variáveis com os mesmos nomes de extensões. Por exemplo, considere a seguinte consulta OQL:

```
select Pessoas
from Pessoas Pessoas, Pessoas Pessoas
```

Sobre tal consulta pode-se fazer uma série de questionamentos. Um deles seria se o termo `Pessoas` que aparece na parte `select` referencia uma das variáveis declaradas na parte `from` ou uma extensão do banco de dados. Poderia-se presumir que `Pessoas` na parte `select` é uma referência a uma das variáveis, mas qual delas? A terceira ocorrência de `Pessoas` na parte `from` da consulta representa uma extensão ou a primeira variável declarada? Quando é possível que haja repetição de nomes e quando se deve considerar duas ocorrências de um mesmo nome um erro? Em cOQL, tais questões foram resolvidas de forma simples: proibindo-se conflitos de nomes pelo sistema de tipos. Não se pode definir nenhuma variável que tenha o mesmo nome que uma extensão, tampouco na cláusula `from` pode-se repetir nomes de variáveis declaradas. Essa proibição é realizada através das restrições nas premissas da regra T-Select.

Tal tratamento para o conflito de nomes faz com que só exista um escopo global de variáveis, não havendo possibilidade de haver escopos aninhados. Enquanto que em linguagens de programação isso consiste de uma limitação importante, em algumas linguagens de consulta simples esta é uma característica convencional.

arg_1	arg_2	arg_3	resultado
$+, -, *$	Int	Int	Int
$<, <=, >, >=$	Int	Int	$Bool$
`and`, `or`	$Bool$	$Bool$	$Bool$
`union`, `intersect`, `difference`	$Bag(\sigma)$	$Bag(\sigma)$	$Bag(\sigma)$
$=, <>$	σ	σ	$Bool$

Figura 2.9: Função δ_b.

Os tipos de operações binárias e unárias são determinados respectivamente por T-Binop e T-Unop. São usadas as funções auxiliares δ_b, apresentada na Figura 2.9, e δ_u, apresentada na Figura 2.10, para determinar o tipo do resultado de cada operação.

arg_1	arg_2	resultado
$-$	Int	Int
`not`	$Bool$	$Bool$
`set`	$Bag(\sigma)$	$Bag(\sigma)$
`size`	$Bag(\sigma)$	Int

Figura 2.10: Função δ_u.

Se o termo q for do tipo c e a do tipo σ pela definição de c, a regra T-DOT associa o tipo σ a $q.a$. A regra T-ARROW age de maneira similar a T-DOT, porém acessando o tipo dos objetos dentro de coleções.

O tipo de uma construção `if-then-else` é dado pela regra T-IF, que verifica se a condição é do tipo *Bool* e dá à construção o tipo σ se as duas subexpressões forem deste mesmo tipo.

A regra T-SELECT tipa estruturas `select-from-where` considerando a expansão do ambiente de tipos com as variáveis declaradas na parte `from`. Seleções da forma `select-from-where` sempre retornam coleções de elementos. Uma estrutura `select-from-where` sempre possui tipo $Bag(\sigma)$, sendo σ o tipo da parte `select` da construção.

A última regra, T-SUB, se refere à relação de subtipo, detalhada a seguir.

2.3.2.1 Relação de subtipo

O conceito de subtipo é encontrado em diversas linguagens, sendo considerado essencial para o estilo de programação orientado a objetos (FISHER, 1996). A *relação de subtipo* é uma relação entre elementos de uma linguagem de tipos cuja interpretação diz respeito à substitutabilidade. Diz-se que um tipo S é subtipo de um tipo T (denotado $S <: T$) se e somente se em toda situação na qual se espera um elemento de T se puder utilizar um elemento do tipo S. Como exemplo, cita-se a relação existente entre números naturais e inteiros ($\mathbb{N} <: \mathbb{Z}$). Pela própria intuição da substitutabilidade, a relação de subtipo deve apresentar as propriedades *reflexiva* e *transitiva*. Para representar graficamente tal relação, normalmente são utilizados *Diagramas de Hasse*, onde são suprimidas as setas de reflexividade e transitividade e se considera a ordem da relação como sendo de baixo para cima.

Um tipo σ_1 é dito subtipo de um tipo σ_2 se existir uma árvore de derivação que prove $\sigma_1 <: \sigma_2$ a partir das regras da Figura 2.11. A cláusula `extends` vem do esquema de classes, indicando que c é subclasse de c'.

$$\sigma <: \sigma$$

$$\frac{\sigma_1 <: \sigma_2 \qquad \sigma_2 <: \sigma_3}{\sigma_1 <: \sigma_3}$$

$$\frac{c \texttt{ extends } c'}{c <: c'}$$

$$\frac{\sigma_1 <: \sigma_2}{Bag(\sigma_1) <: Bag(\sigma_2)}$$

Figura 2.11: Relação de subtipo.

Quando a relação de subtipo deriva de uma especificação explícita (de um esquema de classes, por exemplo), ela é dita *nominal*. Na Figura 2.12 é ilustrada a relação de subtipo entre os tipos de objetos definidos pelo esquema da Figura 2.1.

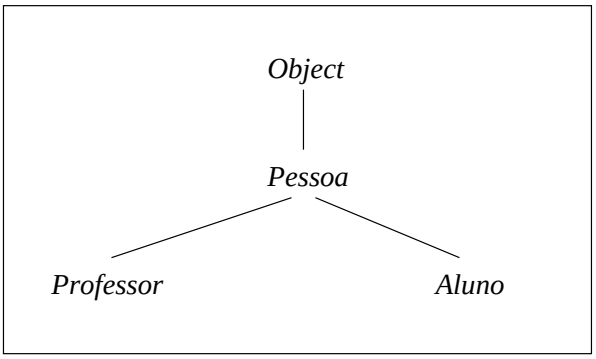

Figura 2.12: Relação de subtipo nominal entre classes.

2.4 Segurança do sistema de tipos

Sistemas de tipos, como mecanismos de validação de construções de uma dada linguagem, são ferramentas poderosas. Através de um sistema de tipos pode-se verificar uma série de restrições de integridade que poderiam levar a erros na execução de programas. Por exemplo, a expressão 5 `union` $\{1, 4\}$ é incoerente e seria rejeitada pelo sistema de tipos, já que o operador `union` só tem sentido quando ambos operandos são coleções. Além de eliminar construções semanticamente erradas, também possibilitam estimar o tipo do resultado de um programa previamente à sua avaliação. Por exemplo, pelo sistema de tipos de cOQL e pelo esquema da Figura 2.1 pode-se provar que o tipo de retorno da seguinte consulta cOQL é $Bag(Int)$.

```
select p.idade
from Pessoas p
where true
```

Diz-se que um sistema de tipo é *seguro* em relação a uma dada semântica formal se consultas bem-tipadas não geram erros em tempo de execução. Erros em tempo de execução, sob a ótica de semântica operacional, são expressões que não são valores e não podem ser avaliadas.

Quando há uma especificação operacional estilo *small-step* para a semântica, a prova da segurança do sistema de tipos pode ser feita através da técnica de prova conhecida por *syntatic soundness* (WRIGHT; FELLEISEN, 1994). Esta técnica garante a segurança do sistema de tipos se duas propriedades forem satisfeitas:

Progresso: uma consulta bem tipada q ou pode ser reduzida para outra consulta q' através das regras de redução da semântica operacional, ou então já é um valor final.

Preservação: ao se reduzir uma consulta bem-tipada q para q', o tipo da consulta é preservado.

Uma das características interessantes deste método de prova é a possibilidade de *extensionalidade*. Ao se modificar ou alterar aspectos específicos da linguagem e da semântica formal, há a possibilidade de se manter partes da prova referentes às construções que não foram alteradas. Tal característica foi levada em consideração quando se escolheu o paradigma operacional, já que se intensionava a representação

de versionamento e temporalização como extensões sobre um modelo-base orientado a objetos. Outra característica é a simplicidade da estrutura lógica de construção da prova, mesmo quando esta é extensa.

O estado do banco de dados e o esquema de classes são representados como ambientes durante a avaliação de consultas e julgamentos de tipo. Entretanto, até então não se estabeleceu formalmente nenhuma relação entre esses ambientes. Sem tal relação, não há como garantir a segurança do sistema de tipos de cOQL. Por exemplo, considere a seguinte consulta:

```
select p.nome
from Pessoas p
where p.idade > 13
```

Considerando o esquema da Figura 2.1, o sistema de tipos de cOQL associa o tipo $Bag(String)$ a esta consulta. Entretanto, se o estado do banco de dados não for restrito pelo esquema, nada impede que o atributo `nome` contenha valores booleanos nos objetos da classe `Pessoa`. Isto poderia fazer com que a avaliação desta consulta resultasse no valor {`true,false`}, invalidando a associação inicial $Bag(String)$ do sistema de tipos.

A relação entre um esquema sch e um estado de banco de dados (ee, oe) é denominada *relação de validade* ($sch \vdash_{sch} ee, oe$) e assegura que os ambientes que formam o estado do banco de dados respeitam as restrições de tipo impostas pelo esquema.

Os seguintes lemas garantem a segurança do sistema de tipos de cOQL se o estado do banco de dados for válido em relação ao esquema:

Lema 2.1 *(Progresso em cOQL) Se* $sch; ee; oe; \Gamma_{sch} \vdash q : \sigma$ *e* $sch \vdash_{sch} (ee, oe)$ *então ou* $q \in Values$ *ou existe* q' *tal que* $ee; oe \vdash q \rightarrow q'$.

Lema 2.2 *(Preservação em cOQL) Se* $sch; ee; oe; \Gamma_{sch} \vdash q : \sigma$, $ee; oe \vdash q \rightarrow q'$ *e* $sch \vdash_{sch} (ee, oe)$, *então* $sch; ee; oe; \Gamma_{sch} \vdash q' : \sigma$.

Na definição das propriedades de cOQL é utilizado o ambiente de tipos Γ_{sch}. Ele representa as associações de variáveis globais (extensões do banco de dados) a seus respectivos tipos (coleções de objetos). O ambiente de tipos Γ_{sch} é definido a seguir.

$$\Gamma_{sch} \stackrel{def}{=} \bigcup_{c \in Dom(sch)} e : Bag(c), \text{ onde } sch(c) = (p, e, a)$$

As provas do lema de progresso de consultas bem-tipadas e do lema da preservação de tipos são razoavelmente extensas. Devido a isso, serão apresentadas junto aos lemas auxiliares e a definição da relação de validade no Anexo I.

2.5 Considerações finais

Modelos formais são ferramentas úteis para melhor se estudar e discutir linguagens e sistemas. O propósito de uma dada formalização afeta significativamente as características específicas que serão modeladas, assim como o método utilizado para tal. Tanto a linguagem cOQL quanto o modelo de dados cODM foram baseados na formalização proposta por Bierman em (BIERMAN, 2003). Contudo IOQL e

cOQL divergem em alguns aspectos justamente por possuírem propósitos diferente. Enquanto IOQL visa o estudo do efeito da avaliação de construções que acessam e modificam o estado do banco de dados, cOQL se propõe a ser uma base para expressar a semântica de uma linguagem exclusivamente de consulta com construções temporais. Os pontos de divergência de cOQL são explicados a seguir com mais detalhe:

Indefinição explícita: Apesar de OQL possuir representação de indefinição explícita através do valor `nill`, provavelmente por simplicidade IOQL não o incluiu na sua gramática. No caso de cOQL a presença de um valor como `nill` é necessária, visto que pode ser utilizado para representar elementos temporais fora do seu tempo de validade quando for necessário especificar uma extensão temporal, por exemplo.

Adoção de `select-from-where`: A linguagem IOQL, por motivo de elegância, utiliza *comprehension syntax*(BUNEMAN et al., 1994). A linguagem cOQL manteve a forma sintática `select-from-where` por ser mais comum e largamente utilizada.

Ausência de métodos e de `new`: A linguagem cOQL se propõe a modelar uma linguagem somente de consulta. A eliminação de métodos e de criação de objetos permite que a relação de redução modifique somente a própria consulta, simplificando a semântica.

Utilização de multiconjuntos: A semântica da linguagem SQL é baseada essencialmente na possibilidade de haver repetições de elementos. Adotou-se o uso de multiconjuntos pela possibilidade de construção de consultas interessantes, como, por exemplo, poder contar quantas vezes um determinado valor aparece em um resultado.

Ausência de tuplas: Tuplas são fundamentais para modelos de banco de dados relacionais. Em cOQL, tuplas não são necessárias para o mecanismo de consulta, visto que os estados dos objetos são representados por funções e construções da forma `select-from-where` retornam sempre coleções de elementos. Acrescentar um construtor de tuplas à cOQL é, todavia, uma tarefa simples, como será visto na definição da linguagem cVOQL.

A linguagem cOQL possui uma semântica operacional simples, o que é uma vantagem para compreensão das estruturas e construção de provas formais. A título de metodologia, também é interessante ressaltar a seqüência de passos dados na especificação de cOQL. Em um primeiro momento são definidas as estruturas básicas: esquema, estado de banco de dados e linguagem de consulta. Posteriormente é estabelecida a relação de validade entre esquemas e estados de banco de dados, assim como a semântica operacional e o sistema de tipos para a linguagem de consulta. Com estes elementos se pode provar a segurança do sistema de tipos, finalizando a especificação. Na Figura 2.13 é apresentado um esboço da relação entre as estruturas que fazem parte da especificação de cOQL.

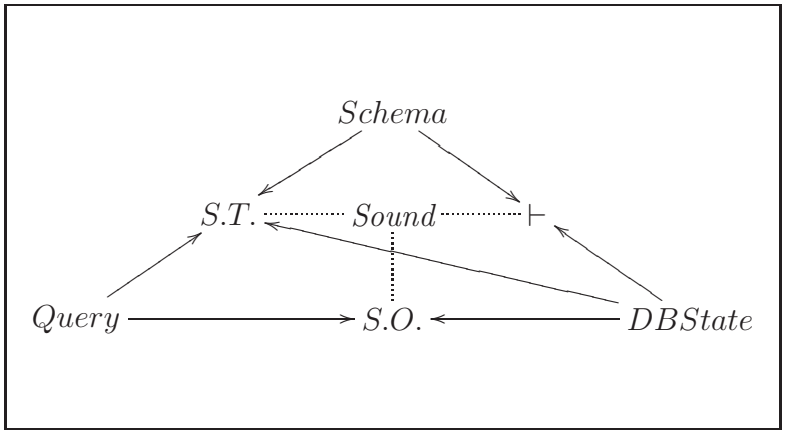

Figura 2.13: Componentes da especificação de cOQL.

3 MODELO VERSIONADO

Aplicações das mais diferentes áreas necessitam de mecanismos de suporte a processos de desenvolvimento evolutivo. Nesse tipo de processo é necessário armazenar diferentes estágios de uma mesma entidade em tempos distintos ou sob diferentes pontos de vista. Esse requisito é modelado através do conceito de *versão*. Bancos de dados versionados provêm suporte a versões alternativas de objetos, simplificando a modelagem de dados de aplicações que exigem tal comportamento. Neste capítulo é apresentado um modelo de banco de dados e uma linguagem de consulta que suportam versões alternativas de objetos.

3.1 Modelo cVODM

O Modelo cVODM (*core Versioned Object Data Model*) é definido como uma expansão do modelo cODM que suporta objetos versionados segundo os conceitos do Modelo TVM (MORO, 2001). O modelo cVODM serve como base para a linguagem de consulta cVOQL, formada pela adição de construções de acesso a versões propostas em TVQL à linguagem cOQL.

3.1.1 Esquema de banco de dados

A seguir é apresentada o formato de declarações de classe cVODM. A única diferença em relação a cODM é a expansão dos tipos possíveis para atributos, que agora englobam também produto de tipos.

$$
\begin{array}{lcl}
class\_def & ::= & \texttt{class}\ c\ \texttt{extends}\ c\ \texttt{extent}\ e \\
 & & \{\ a_1 : \sigma_1,\ \ldots,\ a_n : \sigma_n\ \} \\
\sigma & ::= & Int \mid Bool \mid String \mid c \mid Bag(\sigma) \mid \sigma_1 * \sigma_2
\end{array}
$$

Em cVODM, diferentemente do Modelo de Versões e do modelo TVM, não são utilizadas classes abstratas no topo da hierarquia para modelar o comportamento de versões e de objetos versionados. É utilizada somente a classe *Object* como raiz da hierarquia de classes, sendo objetos versionados suportados por todas as classes. Na Figura 3.1 é apresentado um exemplo de esquema de classes cVODM.

3.1.2 Estado de banco de dados

A principal diferença entre cODM e cVODM diz respeito ao estado do banco de dados. No Modelo de Versões e em TVM, objetos dividem-se entre *normais*, quanto estabelecem um único estado, e *versionados*, quando possuem um conjunto de versões organizadas sob a forma de árvore como é ilustrado na Figura 3.2. Dentre

```
class Pessoa extends Object extent  Pessoas
{
   nome  : String
   idade : Int
}

class Aluno extends Pessoa extent Alunos

{
   matricula : String
   e_orientado : Professor
}

class Professor extends Pessoa extent Professores
{
   titulacao : String
   orienta : Bag(Aluno)
}
```

Figura 3.1: Exemplo de definição de esquema cVODM.

as possíveis versões dentro de um objeto versionado, uma é considerada a versão corrente. Salvo quando se especifica iteração sobre todas as versões de um objeto, uma referência a objeto versionado sempre resulta na versão corrente.

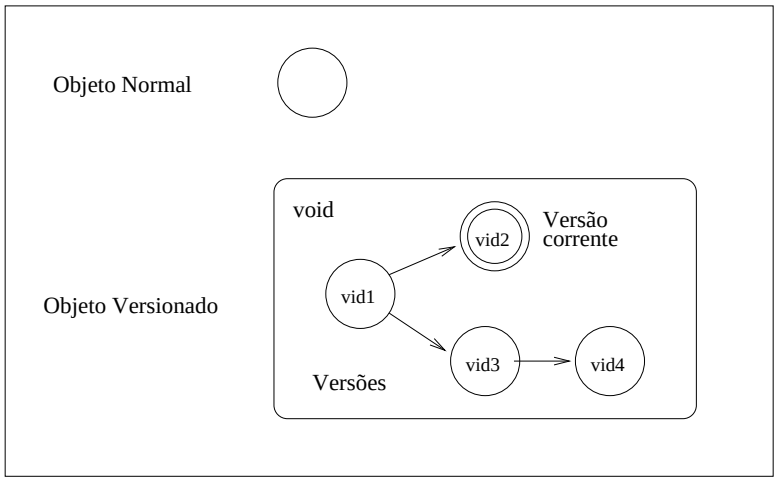

Figura 3.2: Objeto normal e objeto versionado no Modelo de Versões.

Em TVM, existe uma diferenciação semântica clara entre objetos versionados, mesmo os que possuem somente uma versão, e objetos normais. No Modelo de Versões, objetos normais e objetos versionados com exatamente uma versão são semanticamente equivalentes. O modelo cVODM, por questão de simplicidade, representa todos os objetos através de uma estrutura que suporta versionamento. Desta forma, qualquer diferença entre objetos normais e versionados deve ser modelada

em função do conteúdo dos objetos, e não de sua estrutura. Considerar estruturas específicas como casos especiais de estruturas mais genéricas geralmente permite especificações mais simples.

O Modelo de Versões utiliza exclusivamente o mecanismo de herança por extensão, apresentado em (BILIRIS, 1990). Já o Modelo TVM permite que se escolha, na definição de classes, se será utilizada herança por extensão ou por refinamento. Defende-se que a inclusão de herança por extensão tornaria muito complexo o modelo formal devido à utilização de um encadeamento para descrever o estado de um objeto. Desta forma, no modelo cVODM se optou pela manutenção do mecanismo de herança por refinamento utilizado em cODM.

3.1.3 Linguagem de consulta cVOQL

A linguagem de consulta cVOQL suporta versionamento de objetos, possuindo construções para iterar sobre versões e testar sua posição na árvore de derivação. Adicionalmente, também é introduzido um construtor de pares de consultas, possibilitando a formação de tuplas.

Na Figura 3.3 é apresentada a sintaxe abstrata de cVOQL, sendo as modificações em relação a cOQL marcadas por asterisco "*".

Em cVOQL são usados dois tipos de identificadores de objetos, um para objetos (*void*) e um para versões (*vid*), representando, respectivamente, referências dinâmicas e referências estáticas de TVM. Versões possuem praticamente a mesma estrutura no estado do banco de dados que *oids* em cOQL. Objetos versionados são referenciados por *void* e podem ser vistos como referências para a estrutura que compõe o estado de um objeto versionado.

No modelo cVODM pares de consultas são construídos através do construtor '`(,)`', como por exemplo `('a',12)`, `(0,true)` e `(4,3)`. Através de um encadeamento de pares pode ser obter tuplas de tamanho arbitrário, como `(true,('a','b))`. Sobre pares são definidas duas operações unárias, `first` e `second`, que retornam, respectivamente, o primeiro elemento e o segundo elemento de um par.

A inclusão de pares no modelo cVODM não diz respeito diretamente ao mecanismo de versionamento, porém é um componente importante de TVQL. Sua inclusão neste ponto da modelagem serve para ilustrar o acréscimo de novos tipos de dados a cOQL. A forma composicional na qual a linguagem foi contruída, seguindo a filosofia de OQL, torna fácil a incorporação de extensões como esta.

Em cVOQL, o acesso a todas as versões de um dado objeto é possível através da construção *q*`->versions`. Esta construção vem da definição da linguagem TVQL, onde versões são acessadas através do operador "." na parte `from` de uma estrutura `select-from-where`.

Comparações entre versões de um mesmo objeto versionado na árvore de derivação são possíveis através dos operadores binários `is_succ` e `is_pred`. A posição absoluta na árvore de derivação pode ser testada através dos operadores unários `is_leaf`, `is_root`, e `is_current`.

Na Figura 3.4 são apresentados alguns exemplos de consultas cVOQL baseadas no esquema de classes da Figura 3.1.

q	$\in$	*Query*	
q	::=	b	booleanos
	|	i	inteiros
	|	s	*strings*
	|	*void*	* identificadores de objeto
	|	*vid*	* identificadores de versão
	|	`nill`	valor indefinido
	|	$\{q_1, \ldots, q_n\}$	multiconjuntos de consultas, $n \geq 0$
	|	(q_1, q_2)	* pares de consultas
	|	x	variável
	|	q_1 *bop* q_2	operação binária
	|	*uop* q_1	operação unária
	|	$q.a$	acesso a membro
	|	q`->`a	acesso a membro em coleção
	|	q`->versions`	* acesso a todas as versões
	|	`if` q_1 `then` q_2 `else` q_3	condicional
	|	`select` q_a	
		`from` q_1 x_1, ..., q_n x_n	seleção, $n \geq 0$
		`where` q_b	
bop	::=	$+ \mid - \mid *$	
	|	$< \mid <= \mid > \mid >= \mid = \mid <>$	
	|	`and` | `or`	
	|	`union` | `intersection` | `difference`	
	|	`is_succ` | `is_pred`	* comparação entre versões
uop	::=	`not` | $-$ | `set` | `size`	
	|	`is_root` | `is_leaf` | `is_current`	* teste sobre versões
	|	`first` | `second`	* acesso a elementos de pares
v	$\in$	$Values$	
v	::=	b	booleanos
	|	i	inteiros
	|	s	*strings*
	|	*vid*	* identificadores de versão
	|	`nill`	valor indefinido
	|	$\{v_1, \ldots, v_n\}$	multiconjuntos de valores $n \geq 0$
	|	(v_1, v_2)	* pares de valores

Figura 3.3: Sintaxe abstrata de cVOQL.

```
// Retorna todas os titulos de todas as versões de professores
// cujos nomes são João
select x.titularidade
from Professores->versions x
where x.nome='João'

// Retorna todos os alunos cujo estado corrente corresponde
// à raiz da árvore de derivação de versões
select x
from Alunos->versions x
where (is_first x) and (is_current x)

// Retorna todos as versões de alunos que descendem da
// versão corrente
select x
from Alunos x, Alunos->versions y
where x is_succ y
```

Figura 3.4: Exemplos de consultas cVOQL.

3.2 Semântica formal de cVOQL

Esta seção estabelece a semântica formal de cVOQL de forma similar à apresentação da semântica de cOQL. Inicialmente é formalizado o estado do banco de dados cVODM e após definida a relação de redução.

3.2.1 Formalização do Estado de Banco de Dados

O estado de um banco de dados cVODM é formado por três ambientes: ambiente de extensões, ambiente de objetos e ambiente de versões, como apresentado a seguir.

3.2.1.1 Ambiente de Extensões

O ambiente de extensões de cVODM é similar ao ambiente de extensões de cODM, diferindo somente por referenciar conjuntos de *void*.

$$vee \in VEE = ExtId \xrightarrow{fin} \mathcal{P}(Void)$$

3.2.1.2 Ambiente de Objetos

A seguir é apresentada a formação de um ambiente de objetos.

$$
\begin{array}{rcl}
voe & \in & VOE = Void \xrightarrow{fin} ClassId \times Vobjstate \\
vobjstate & \in & Vobjstate = \mathcal{P}(Vid) \times Vid \times \mathcal{P}(Vid \times Vid) \\
vobjstate & ::= & (versions,\ current,\ deriv)
\end{array}
$$

O estado de um objeto versionado é composto por três elementos:

- *versions* é o conjunto de todas as versões do objeto versionado;
- *current* é a versão corrente do objeto;
- *deriv* representa a relação de derivação entre as versões do objeto versionado.

Em cVODM os objetos não definem diretamente os atributos, se valendo de versões para isso.

3.2.1.3 Ambiente de Versões

No estado de um banco de dados cVODM, o ambiente de versões possui estrutura similar ao ambiente de objetos de cODM, tendo por objetivo definir estados possíveis de objetos. A formação de um ambiente de versões é apresentada a seguir:

$$
\begin{array}{rcl}
ve \in VE & = & Vid \xrightarrow{fin} ClassId \times VersionState \\
vstate \in VersionState & = & Ident \rightarrow Stored
\end{array}
$$

Em cVODM existem dois tipos de referências a objetos: estáticas e dinâmicas. Referências estáticas apontam para versões (*vid*) e referências dinâmicas apontam para objetos (*void*), que referenciam versões específicas de acordo com a sua configuração. Em cVODM, assim como em TVM, tanto referências dinâmicas quanto referências estáticas podem fazer parte do estado dos objetos. É apresentado a seguir o conjunto *Stored* de valores armazenáveis do modelo cVODM, que difere de $Values$ justamente por incluir referências dinâmicas.

k	$\in$	$Stored$	
k	$::=$	b	booleanos
	$\mid$	i	inteiros
	$\mid$	s	strings
	$\mid$	$void$	identificadores de objeto
	$\mid$	vid	identificadores de versão
	$\mid$	$\{k_1,\ \ldots,\ k_n\}$	multiconjuntos $n \geq 0$
	$\mid$	(k_1, k_2)	pares

3.2.1.4 Estado de Banco de dados cVODM

Um estado de banco de dados cVODM é um tripla composta por um ambiente de extensões, um ambiente de objetos e um ambiente de versões.

$$vdbstate \in VDBState = VEE \times VOE \times VE$$

3.2.2 Relação de Redução

A relação de redução entre consultas cVOQL introduz poucas modificações em relação à semântica de cOQL. A redução da consulta q para q' sob o estado $vee; voe; ve$ é denotada

$$vee; voe; ve \vdash q \rightarrow q'$$

As regras de redução são apresentadas na Figura 3.5, sendo as modificações em relação as regras de cOQL marcadas com um '*'.

A seguir, são explicadas as modificações nas regras em relação à semântica dada a cOQL.

Para que a regra VS-Ctx possa conduzir apropriadamente a avaliação de subconsultas, o contexto de avaliação $\mathcal{E}$ precisa ser expandido para englobar os novos contrutores.

$$\mathcal{E} ::= \ldots \mid (\mathcal{E}, q) \mid (v, \mathcal{E}) \mid \mathcal{E}\texttt{->versions}$$

A regra VS-Void avalia uma referência dinâmica (*void*) para sua versão corrente de acordo com o ambiente de objetos.

Construções da forma q`->versions` acessam todas as versões possíveis de um dado objeto através da regra VS-Versions1. As regras VS-Versions2, VS-Versions3 e VS-Versions4 aplica a mesma semântica do operador q`->`a para os outros casos possíveis de consulta.

As funções η_b e η_u são expandidas para incluir as operações que testam propriedades de versões a partir da relação *deriv* definida no estado dos objetos versionados. O operador `is_succ` retorna `true` se ambas as versões pertencem ao mesmo objeto versionado e a primeira versão sucede a segunda versão na árvore de derivação. O operador `is_pred` funciona de forma análoga, retornando `true` se o primeira versão precede a segunda na árvore de derivação. Se as versões não pertencem ao mesmo objeto versionado, simplesmente ambas as operações retornam `false`. Os seguintes exemplos consideram o estado de objeto versionado da figura 3.2.

```
vid1 is_succ vid3 = false
vid4 is_succ vid1 = true
vid1 is_pred vid5 = false
vid1 is_pred vid4 = true
```

A posição absoluta na árvore de derivação pode ser testada através dos operadores unários, `is_leaf`, que testa se a versão é uma das folhas, e `is_root`, que testa se ela é a raiz. A versão corrente é testada pela operação unária `is_current`. Tendo por base o objeto versionado apresentado na Figura 3.2, tem-se os seguintes resultados:

```
is_root vid1 = true
is_leaf vid2 = false
is_current vid2 = true
is_current vid1 = false
```

As operações unárias `first` e `second` são definidas para acessar elementos de pares, como apresentado a seguir:

$$\frac{vee; voe; ve \vdash q \rightarrow q'}{vee; voe; ve \vdash \mathcal{E}[q] \rightarrow \mathcal{E}[q']} \quad \text{(VS-Ctx)}$$

$$\frac{voe(void) = (c, (versions, current, deriv))}{vee; voe; ve \vdash void \rightarrow current} \quad \text{(* VS-Void)}$$

$$\frac{vee(x) = \{\overrightarrow{void}\}}{vee; voe; ve \vdash x \rightarrow \{\overrightarrow{void}\}} \quad \text{(VS-Var)}$$

$$\frac{\eta_b(bop, v_1, v_2) = v_3}{vee; voe; ve \vdash v_1\ bop\ v_2 \rightarrow v_3} \quad \text{(VS-Binop)}$$

$$\frac{\eta_u(uop, v_1) = v_2}{vee; voe; ve \vdash uop\ v_1 \rightarrow v_2} \quad \text{(VS-Unop)}$$

$$\frac{ve(vid) = (c, vstate) \qquad vstate(a) = k}{vee; voe; ve \vdash vid.a \rightarrow k} \quad \text{(VS-Dot1)}$$

$$vee; voe; ve \vdash \texttt{nill}.a \rightarrow \texttt{nill} \quad \text{(VS-Dot2)}$$

$$vee; voe; ve \vdash \{vid, \vec{v}\}\texttt{->}a \rightarrow vid.a\ \texttt{union}\ \{\vec{v}\}\texttt{->}a \quad \text{(VS-Arrow1)}$$

$$vee; voe; ve \vdash \{\texttt{nill}, \vec{v}\}\texttt{->}a \rightarrow \{\vec{v}\}\texttt{->}a \quad \text{(VS-Arrow2)}$$

$$vee; voe; ve \vdash \{\}\texttt{->}a \rightarrow \{\} \quad \text{(VS-Arrow3)}$$

$$vee; voe; ve \vdash \texttt{nill->}a \rightarrow \texttt{nill} \quad \text{(VS-Arrow4)}$$

$$\frac{voe(void) = (c, versions, current, deriv) \qquad vid \in versions}{vee; voe; ve \vdash \{vid, \vec{v}\}\texttt{->versions} \rightarrow versions\ \texttt{union}\ \{\vec{v}\}\texttt{->versions}} \quad \text{(* VS-Versions1)}$$

$$vee; voe; ve \vdash \{\texttt{nill}, \vec{v}\}\texttt{->versions} \rightarrow \{\vec{v}\}\texttt{->versions} \quad \text{(* VS-Versions2)}$$

$$vee; voe; ve \vdash \{\}\texttt{->versions} \rightarrow \{\} \quad \text{(* VS-Versions3)}$$

$$vee; voe; ve \vdash \texttt{nill->versions} \rightarrow \texttt{nill} \quad \text{(* VS-Versions4)}$$

$$vee; voe; ve \vdash \texttt{if true then}\ q_1\ \texttt{else}\ q_2 \rightarrow q_1 \quad \text{(VS-If1)}$$

$$vee; voe; ve \vdash \texttt{if false then}\ q_1\ \texttt{else}\ q_2 \rightarrow q_2 \quad \text{(VS-If2)}$$

$$vee; voe; ve \vdash \texttt{if nill then}\ q_1\ \texttt{else}\ q_2 \rightarrow \texttt{nill} \quad \text{(VS-If3)}$$

$$\begin{array}{c} vee; voe; ve \vdash \texttt{select}\ q_a\ \texttt{from}\ \{v_1, \ldots, v_m\}\ x_1, \ldots, q_n\ x_n\ \texttt{where}\ q_b \rightarrow \\ (\texttt{select}\ q_a\ \texttt{from}\ q_2\ x_2, \ldots, q_n\ x_n\ \texttt{where}\ q_b)[x_1 ::= v_1] \\ \texttt{union} \\ \texttt{select}\ q_a\ \texttt{from}\ \{v_2, \ldots, v_m\}\ x_1, \ldots, q_n\ x_n\ \texttt{where}\ q_b \end{array} \quad \text{(VS-Select1)}$$

$$vee; voe; ve \vdash \texttt{select}\ q_a\ \texttt{from}\ \{\}\ x_1, \ldots, q_m\ x_n\ \texttt{where}\ q_b \rightarrow \{\} \quad \text{(VS-Select2)}$$

$$vee; voe; ve \vdash \texttt{select}\ q_a\ \texttt{from nill}\ x_1, \ldots, q_m\ x_n\ \texttt{where}\ q_b \rightarrow \texttt{nill} \quad \text{(VS-Select3)}$$

$$vee; voe; ve \vdash \texttt{select}\ q_a\ \texttt{from where true} \rightarrow \{q_a\} \quad \text{(VS-Select4)}$$

$$vee; voe; ve \vdash \texttt{select}\ q_a\ \texttt{from where false} \rightarrow \{\} \quad \text{(VS-Select5)}$$

$$vee; voe; ve \vdash \texttt{select}\ q_a\ \texttt{from where nill} \rightarrow \{\} \quad \text{(VS-Select6)}$$

Figura 3.5: Semântica operacional de cVOQL.

```
first (12,3) = 12
second ('a',true) = true
second ('a',(1,2)) = (1,2)
```

Por fim, a operação de substituição deve ser modificada para contemplar as mudanças na gramática como apresentado na Figura 3.6.

$$
\begin{array}{lcl}
void[x ::= v] & = & void \\
vid[x ::= v] & = & vid \\
(q_1, q_2)[x ::= v] & = & (q_1[x ::= v], q_2[x ::= v]) \\
(q\texttt{->versions})[x ::= v] & = & (q[x ::= v])\texttt{->versions}
\end{array}
$$

Figura 3.6: Acréscimos à operação de substituição em cVOQL.

3.3 Sistema de tipos de cVOQL

O conjunto de tipos de consultas cVOQL é o mesmo usado em cOQL com a adição de *produtos de tipos*. A seguir são apresentados os tipos de consultas cVOQL.

$$
\begin{array}{lcl}
\sigma & \in & Type \\
\sigma & ::= & Int \mid Bool \mid String \mid c \mid Bag(\sigma) \mid \sigma_1 * \sigma_2
\end{array}
$$

3.3.1 Formalização de Esquema

A seguir é apresentada a formalização de um esquema de classes cVODM, cuja única diferença em relação a cODM é a modificação do conjunto *Type*.

$$
\begin{array}{llcl}
p & \in\ ClassId_\perp & = & ClassId \cup \{\perp\} \\
vsch & \in\ Vschema & = & ClassId \xrightarrow{fin} Classdef \\
 & Classdef & = & ClassId_\perp \times ExtentId \times Ty \\
ty & \in\ Ty & = & Ident \xrightarrow{fin} Type
\end{array}
$$

3.3.2 Julgamentos de tipo

Um julgamento de tipo de uma consulta cVOQL q possui a forma

$$vsch; vee; voe; ve; \Gamma \vdash q : \sigma$$

onde

- *vsch* é um *esquema de dados* pertencente ao conjunto $Vschema$ definido na Seção 3.3.1;

$$K \vdash b : Bool \qquad \text{(VT-Bool)}$$

$$K \vdash i : Int \qquad \text{(VT-Int)}$$

$$K \vdash s : String \qquad \text{(VT-Str)}$$

$$\frac{voe(void) = (c, vobjstate)}{K \vdash void : c} \qquad \text{(* VT-Void)}$$

$$\frac{ve(vid) = (c, vstate)}{K \vdash vid : c} \qquad \text{(* VT-Vid)}$$

$$K \vdash \texttt{nill} : \sigma \qquad \text{(VT-Nill)}$$

$$\frac{K \vdash q_1 : \sigma \quad \ldots \quad K \vdash q_n : \sigma \qquad n \geq 0}{K \vdash \{q_1, \ldots, q_n\} : Bag(\sigma)} \qquad \text{(VT-Bag)}$$

$$\frac{K \vdash q_1 : \sigma_1 \qquad K \vdash q_2 : \sigma_2}{K \vdash (q_1, q_2) : \sigma_1 * \sigma_2} \qquad \text{(* VT-Pair)}$$

$$\frac{\Gamma(x) = \sigma}{vsch; vee; voe; ve; \Gamma \vdash x : \sigma} \qquad \text{(VT-Var)}$$

$$\frac{K \vdash q_1 : \sigma_1 \qquad K \vdash q_2 : \sigma_2 \qquad \delta_b(bop, \sigma_1, \sigma_2) = \sigma_3}{K \vdash q_1\ bop\ q_2 : \sigma_3} \qquad \text{(VT-Binop)}$$

$$\frac{K \vdash q : \sigma_1 \qquad \delta_u(uop, \sigma_1) = \sigma_2}{K \vdash uop\ q : \sigma_2} \qquad \text{(VT-Unop)}$$

$$\frac{K \vdash q : c \qquad vsch(c) = (p, e, ty) \qquad ty(a) = \sigma}{K \vdash q.a : \sigma} \qquad \text{(VT-Dot)}$$

$$\frac{K \vdash q : Bag(c) \qquad vsch(c) = (p, e, ty) \qquad ty(a) = Bag(\sigma)}{K \vdash q\texttt{->}a : Bag(\sigma)} \qquad \text{(VT-Arrow)}$$

$$\frac{K \vdash q : Bag(c)}{K \vdash q\texttt{->versions} : Bag(c)} \qquad \text{(* VT-Versions)}$$

$$\frac{K \vdash q_1 : Bool \qquad K \vdash q_2 : \sigma \qquad K \vdash q_3 : \sigma}{K \vdash \texttt{if}\ q_1\ \texttt{then}\ q_2\ \texttt{else}\ q_3 : \sigma} \qquad \text{(VT-If)}$$

$$\frac{\begin{array}{rcl} vsch; vee; voe; ve; \Gamma & \vdash & q_1 : Bag(\sigma_1) \\ vsch; vee; voe; ve; \Gamma, x_1 : \sigma_1 & \vdash & q_2 : Bag(\sigma_2) \\ & \vdots & \\ vsch; vee; voe; ve; \Gamma, x_1 : \sigma_1, \ldots, x_{n-1} : \sigma_{n-1} & \vdash & q_n : Bag(\sigma_n) \\ vsch; vee; voe; ve; \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n & \vdash & q_a : \sigma_a \\ vsch; vee; voe; ve; \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n & \vdash & q_b : Bool \\ x_1 \text{ a } x_n \text{ diferentes} \qquad x_1, \ldots, x_n \notin Dom(\Gamma) & & n \geq 0 \end{array}}{vsch; vee; voe; ve; \Gamma \vdash \texttt{select}\ q_a\ \texttt{from}\ q_1\ x_1\ , \ldots, q_n\ x_n\ \texttt{where}\ q_b : Bag(\sigma_a)} \qquad \text{(VT-Select)}$$

$$\frac{K \vdash q : \sigma_1 \qquad \sigma_1 <: \sigma_2}{K \vdash q : \sigma_2} \qquad \text{(VT-Sub)}$$

Figura 3.7: Sistema de tipos de cVOQL.

- *vee* é um ambiente de extensões pertencente ao conjunto VEE definido na Seção 3.2.1.1;

- *voe* é um ambiente de objetos pertencente ao conjunto VOE definido na Seção 3.2.1.2;

- *ve* é um ambiente de versões, definido na Seção 3.2.1.3;

- Γ é um ambiente de tipos.

As regras para derivar julgamentos de tipos são apresentadas na Figura 3.7. No que segue, são explicadas as regras acrescentadas e modificadas em relação ao sistema de tipos de cOQL.

A regra VT-OID é substituída por duas regras, VT-VOID e VT-VID, que tipam, respectivamente, identificadores de objetos e identificadores de versões. A regra VT-VOID utiliza a informação armazenada no ambiente de objetos para tipar referências dinâmicas *void*. A regra VT-VID, similarmente, define o tipo de uma referências estáticas *vid*. A regra VT-PAIR dá tipos a pares de consultas, baseado nos tipos de cada componente do par. A regra VT-BINOP fornece tipos para operações binárias utilizando a função δ_b. Para incluir as operações definidas em cVOQL, a função δ_b é expandida como apresentado na Figura 3.8.

arg_1	arg_2	arg_3	resultado
`is_succ`, `is_pred`	c	c	*Bool*

Figura 3.8: Expansão da função δ_b.

A regra VT-UNOP fornece tipos para operações unárias utilizando a função auxiliar δ_u. A função δ_u é expandida como apresentado na Figura 3.9.

arg_1	arg_2	resultado
`is_root`, `is_leaf`, `is_current`	c	*Bool*
`first`	$\sigma_1 \times \sigma_2$	σ_1
`second`	$\sigma_1 \times \sigma_2$	σ_2

Figura 3.9: Expansão da função δ_u.

A regra VT-VERSIONS garante que a construção `q->versions` só será aplicada sobre coleções de objetos.

A relação de subtipo, utilizada pela regra VT-SUB, também deve ser expandida para incluir tipos de pares. A Figura 3.10 apresenta a relação de subtipo de cVOQL.

3.4 Segurança do sistema de tipos

Ao propor modificações sobre o modelo de dados e sobre a linguagem de consulta, espera-se que elas não introduzam inconsistências entre a semântica operacional e o sistema de tipos. Feitas as devidas modificações na relação de validade entre esquemas e estados de banco de dados, pode-se garantir que o sistema de tipos de cVOQL é seguro em relação à semântica operacional através dos seguintes lemas.

$$\sigma <: \sigma$$

$$\frac{\sigma_1 <: \sigma_2 \qquad \sigma_2 <: \sigma_3}{\sigma_1 <: \sigma_3}$$

$$\frac{c \texttt{ extends } c'}{c <: c'}$$

$$\frac{\sigma_1 <: \sigma_2}{Bag(\sigma_1) <: Bag(\sigma_2)}$$

$$\frac{\sigma_1 <: \sigma_1' \qquad \sigma_2 <: \sigma_2'}{\sigma_1 * \sigma_2 <: \sigma_1' * \sigma_2'}$$

Figura 3.10: Relação de subtipo de cVOQL.

Lema 3.1 *(Progresso em cVOQL) Se* $vsch; vee; voe; ve; \Gamma_{vsch} \vdash q : \sigma$ *e* $vsch \vdash_{vsch} (vee, voe, ve)$*, então ou* $q \in Values$ *ou existe* q' *tal que* $vee; voe; ve \vdash q \rightarrow q'$*.*

Lema 3.2 *(Preservação em cVOQL) Se* $vsch; vee; voe; ve; \Gamma_{vsch} \vdash q : \sigma$*,* $vee; voe; ve \vdash q \rightarrow q'$ *e* $vsch \vdash_{vsch} (vee, voe, ve)$*, então* $vsch; vee; voe; ve; \Gamma_{vsch} \vdash q' : \sigma$*.*

O ambiente de tipos Γ_{vsch} é definido da mesma forma que Γ_{sch}.

$$\Gamma_{vsch} \stackrel{def}{=} \bigcup_{c \in Dom(vsch)} e : Bag(c), \text{ onde } vsch(c) = (p, e, a)$$

A definição da relação de validade e as provas formais são apresentadas no Anexo II.

3.5 Considerações finais

Neste capítulo o modelo cODM e a linguagem cOQL foram remodelados para incorporar o conceito de objetos versionados, presente no Modelo TVM. Por se tratar de uma simplificação, várias características não foram incluídas, como o ciclo de vida de versões, herança por extensão e diferenciação estrutural entre objetos normais e objetos versionados.

É interessante observar que, em cVOQL, o conceito de versionamento não acarreta nenhuma modificação no sistema de tipos. Em um primeiro momento, poder-se-ia pensar na estrutura de versionamento como um construtor que geraria valores versionados a partir de valores básicos. Assim como `{11,23}` define um conjunto de inteiros, poderia haver algo como `ver({1,4,5},4,{1->4,1->5})` para construir um "inteiro versionado". Desta forma seria possível encapsular o comportamento de versionamento em um tipo de dados abstrato, instanciando quando necessário e acessando através de operadores como `current` ou `versions`. Tal escolha, entretanto, levaria a uma diferenciação do tratamento dado a inteiros, sob os quais

pode-se aplicar as operações tradicionais, e "inteiros versionados", cujos valores seriam acessíveis somente através de uma interface pré-estabelecida, implicando na definição de um novo tipo parametrizado $Versioned(Int)$. Isto faria surgir duas possibilidades para o estado de objetos, ambas presentes em extensões tais como *Objects*, levando à necessidade de se especificar dois comportamentos distintos para objetos na semântica operacional. Tratar objetos normais como objetos versionados com exatamente uma versão simplifica a semântica e o sistema de tipos. Se houver a necessidade de uma diferenciação, esta pode ser introduzida sem maiores problemas nas restrições da relação de validade.

4 MODELO TEMPORAL E VERSIONADO

O objetivo geral deste trabalho é prover semântica operacional para as construções temporais da linguagem TVQL. Para tal, é necessário representar a estrutura de um banco de dados que suporte rótulos temporais segundo o modelo TVM. Neste capítulo é definido o modelo de dados cTVM (*core Temporal Versions Model*). Sobre este modelo é especificada a linguagem cTVQL, estabelecendo uma semântica formal para as construções temporais propostas em TVQL.

4.1 Modelo TVM

O Modelo Temporal de Versões, apresentado em (MORO, 2001), é um modelo orientado a objetos que trata homogeneamente os conceitos de tempo e versões. Classes do modelo TVM podem ser ou *temporais versionadas* ou *normais*. Uma classe temporal versionada pode ter, além de atributos convencionais, atributos *temporais*. Na Figura 4.1 é apresentado um exemplo de classe temporal versionada, identificada como tal pela cláusula `hasVersions`.

```
class Pessoa extent Pessoas hasVersions
{
temporal nome  : String ;
         rg    : Int ;
         fone  : String ;
temporal idade : Int ;
}
```

Figura 4.1: Exemplo de definição de classe TVM.

Quando um atributo não-temporal tem seu valor alterado, o valor anterior é sobrescrito e não há mais como recuperá-lo. Quando um atributo temporal é alterado, o valor anterior é registrado como válido até a data da alteração e o novo valor é registrado como válido a partir da data da alteração, permitindo que uma consulta realizada em diversos pontos do tempo apresente resultados diferentes. Quando um atributo normal de um objeto versionado é alterado, se cria uma nova versão do objeto com o valor atualizado e a versão corrente passa a ser esta nova versão a partir de então. Objetos versionados registram alterações de valores não-temporais através de novas versões que são criadas.

4.2 Modelo cTVM

Assim como TVM incorpora conceitos do Modelo de Versões, o modelo cTVM incorpora elementos do modelo cVODM. Inicialmente será apresentada a forma de representação temporal e da atualização de valores ao longo do tempo, e depois como isto é integrado ao modelo cVODM.

4.2.1 Representação do tempo

Quando considera-se o conceito de tempo em bancos de dados, torna-se necessário estabelecer como ele será representado. A seguir é proposta uma representação simples que permite a utilização de instantes e intervalos pela semântica operacional de cTVQL.

4.2.1.1 Instantes

O tempo em cTVM é considerado *linear* e *discreto*, isto é, representado por um linha contínua na qual existe uma distância mínima entre dois elementos consecutivos.

Muitos sistemas de bancos de dados utilizam formatos legíveis por humanos, tais como `'30/12/1999'` ou `'13:20'`, para representação de datas e horas. Neste trabalho, entretanto, serão utilizados números inteiros para representar instantes específicos de tempo. Para tornar a semântica mais clara, é interessante que se diferencie quando um dada ocorrência de número inteiro está no contexto aritmético e quando está no contexto temporal. Será utilizada uma sintaxe específica para o caso de referência à dimensão temporal, sendo os inteiros posfixados pelo sufixo '`t`', como `1t`, `20t` e `-3t`, por exemplo. O conjunto de todos os instantes de tempo é denotado $Time$.

$$t \in Time = \{\ldots, -\texttt{2t}, -\texttt{1t}, \texttt{0t}, \texttt{1t}, -\texttt{2t}, \ldots\}$$

O conjunto $Time$ é totalmente ordenado segundo a relação $<$, como apresentado a seguir.

$$\ldots < -\texttt{2t} < -\texttt{1t} < \texttt{0t} < \texttt{1t} < \texttt{2t} < \ldots$$

4.2.1.2 Intervalos

Um intervalo é um conjunto não-vazio e contíguo de instantes temporais, sendo representado pelos seus limites inferior e superior. Por exemplo, são intervalos `[0t..13t]`, `[-10t..23t]` e `[1t..1t]`. Ambos os limites do intervalo são elementos do conjunto $Time$. Pode haver casos de intervalos sem limite inferior, limite superior ou ambos os limites. Por exemplo, o intervalo `[0t..]` comporta todos os instantes a partir de `0t`, o intervalo `[..1t]` engloba todos os elementos até `1t` e o intervalo `[..]` contém elementos do conjunto $Time$. O conjunto de todos os possíveis intervalos é denotado $Interval$.

$$\begin{array}{lcl} u & \in & Interval \subseteq \mathcal{P}(Time) \\ u & ::= & [t_1..t_2] \quad | \quad [t..] \quad | \quad [..t] \quad | \quad [..] \end{array}$$

Por simplicidade, quando ambos os limites estiverem definidos, a ordem dos valores não será relevante para a descrição dos intervalos. Assim sendo, `[0t..10t]` equivale a `[10t..0t]`.

4.2.1.3 Valores temporais

Com base no modelo de tempo proposto, pode-se então definir de que forma a introdução de uma linha de tempo no modelo afeta a representação e o comportamento de valores. Valores convencionais das linguagens cOQL e cVOQL, tais como `true` ou {`12, 42`}, podem ser considerados *constantes* em relação à passagem do tempo. Em contrapartida, pode-se pensar em valores que sofram modificações em pontos específicos ao longo da linha de tempo.

Uma forma possível de pensar um valor temporal é como uma função do tipo $Time \rightarrow Values$, que associa um diferente valor para cada instante de tempo. Por exemplo, a função que resulta em `nill` em qualquer instante de tempo menor que `0t` e que resulta `5` em qualquer instante de tempo maior ou igual a `0t` pode ser descrita em *notação lambda* como

$$\lambda t.(\texttt{if } t < \texttt{0t then nill else 5})$$

Os atributos marcados como temporais no Modelo TVM podem apresentar valores diferentes dependendo do instante temporal em que são acessados. Considerando a classe apresentada na Figura 4.1, pode-se ver um valor associado ao atributo `idade` como uma função do tipo $Time \rightarrow Int$. Poder-se-ia introduzir ao sistema de tipos um construtor de tipos temporais T descrito a seguir, que gera *tipos temporais* a partir de tipos já existentes.

$$T(\sigma) \equiv Time \rightarrow \sigma$$

Assim sendo, se definiria na semântica operacional construtores para elementos de $T(\sigma)$ e o valor do atributo `idade` poderia ser considerado do tipo $T(Int)$.

Entretanto, tal abordagem causa um problema considerando um dos objetivos da linguagem de consulta TVQL. É defendido em (SNODGRASS, 2000) que uma característica desejável de uma linguagem de consulta temporal é que o tratamento de dados temporais e de dados não-temporais seja transparente ao usuário, isto é, ambos devam ser tratados de forma uniforme. A linguagem TVQL segue tal assertiva, permitindo que ambos sejam livremente operados. Por exemplo, a seguinte consulta TVQL deveria ser válida e bem-tipada.

```
select x.idade + x.rg
from Pessoas x
```

Considerando que `x.idade` é do tipo $T(Int)$ e `x.rg` é do tipo Int, se torna necessário tratar o caso da soma de elementos matemáticos de tipos diferentes, no caso inteiros com funções do tipo $Time \rightarrow Int$.

A solução proposta para permitir que valores não-temporais e valores temporais sejam arbitrariamente operados sem ferir restrições de tipo é supor que sempre se está trabalhando com valores temporais. Considerar todos os valores, tais como `4`, `true` e `nill`, como funções constantes do tipo $Time \rightarrow Values$ permite que estes sejam operados arbitrariamente com outros valores temporais ao mesmo tempo que mantém o seu significado original de não variarem ao longo do tempo. Além disso, esta proposição continua permitindo que se diferencie valores não-temporais e valores temporais. Isto pode ser representado pelo fato do valor temporal ser representado

por uma função *constante*. A partir deste ponto no texto, funções constantes serão chamadas *valores fixos* e funções que apresentam modificações, *valores variáveis*.

Sintaticamente, a representação de valores fixos é a mesma que a utilizada nos modelos anteriores, diferindo exclusivamente na semântica. Enquanto que o elemento sintático `4` representa o numeral 4 em cOQL e cVOQL, em cTVQL ele representa a função $\lambda t.4$. Valores variáveis, por sua vez, são construídos através do construtor sintático ($\texttt{v}_1$ /t/ $\texttt{v}_2$). Este construtor recebe como parâmetros dois valores quaisquer e um instante de tempo, e o valor que representa é equivalente a v_1 em todos os instantes menores que t e equivalente a v_2 em todos os instantes maiores ou iguais a t. Uma descrição em notação lambda do seu significado é

$$(v_1 \ /t/ \ v_2) \stackrel{def}{=} \lambda w.\texttt{if } w < t \texttt{ then } v_1(w) \texttt{ else } v_2(w)$$

Por exemplo, o seguinte valor temporal retorna `nill` até o intante `0t`, `4` de `1t` até `9t` e `-20` em todos os instantes de tempo maiores que `10t`.

$$\texttt{nill /0t/ 4 /10t/ } -20$$

4.2.2 Esquema de banco de dados cTVM

O esquema de dados de TVM diferencia classes que podem possuir objetos temporais e versionados de classes normais. Tais informações têm efeito principalmente sobre o mecanismo de atualização do banco de dados, o que não é tratado neste trabalho. Devido à forma escolhida para representar valores, a diferenciação entre elementos temporais e não-temporais é transparente para a linguagem de consulta, tornando desnecessário diferenciar atributos temporais e não-temporais. Assim como em cVODM, defende-se que, caso necessário, tal diferenciação seja feita através das restrições de validade entre o esquema e o estado. A seguir, é apresentada a sintaxe para definição de classes do modelo cTVM.

$$\begin{array}{lll} \textit{class\_def} & ::= & \texttt{class } c \texttt{ extends } c \texttt{ extent } e \\ & & \{\ a_1 : \sigma_1,\ \ldots,\ a_n : \sigma_n\ \} \\ \sigma & ::= & \textit{Int} \mid \textit{Bool} \mid \textit{String} \mid \textit{Time} \mid \textit{Interval} \mid c \mid \textit{Bag}(\sigma) \mid \sigma_1 * \sigma_2 \end{array}$$

Foram incluídos ao conjunto de tipos possíveis o conjunto $Time$ de instantes temporais, descrito na Seção 4.2.1.1, e o conjunto $Interval$, descrito na Seção 4.2.1.2. Na Figura 4.2 é apresentado um exmplo de esquema de banco de dados cTVM.

4.2.3 Estado de banco de dados cTVM

O estado do banco de dados cTVM incorpora *rótulos temporais* às estruturas apresentadas no modelo cVODM. Cada rótulo temporal corresponde ao *intervalo de vida* de cada construção. No modelo TVM, o tempo de vida de um objeto corresponde a dois aspectos: *intervalo de transação*, quando o objeto foi inserido no banco de dados mas ainda não está ativo, e *intervalo de validade*, quando é válido e pode ser consultado. Por simplicidade, em cTVM só será utilizado um desses elementos, o intervalo de validade, para indicar o tempo de vida do objeto. A partir deste ponto os termos "tempo de validade" e "tempo de vida" serão indistintamente utilizados para representar a mesma entidade.

A figura 4.3 ilustra a estrutura interna de um objeto versionado temporal. Cada objeto temporal possui um intervalo de validade associado (*alive*). Da mesma forma,

```
class Pessoa extends Object extent  Pessoas
{
   nome     : String
   endereco : String
   nasc     : Time
   telefone : String
}

class Aluno extends Pessoa extent Alunos
{
   matricula   : String
   e_orientado : Professor
}

class Professor extends Pessoa extent Professores
{
   titulacao : String
   orienta   : Bag(Aluno)
}
```

Figura 4.2: Exemplo de definição de esquema cTVM.

cada versão possui seu respectivo intervalo de validade. Nota-se que o tempo é linear para cada objeto e é ramificado para as versões de um objeto.

Em um objeto temporal versionado, as modificações de versão corrente são registradas em *current*. Em um objeto versionado de cVODM somente a versão corrente atual é marcada.

A Figura 4.4 apresenta uma representação do estado de uma versão temporal que possui dois atributos, `a` e `b`. Mesmo que se estivesse diferenciando valores temporais de não-temporais, um valor como `a`, que não sofre variação, deveria ser considerado como temporal já que a expressão `vid.a` só tem sentido no intervalo de tempo que `vid` é válido. Por sua vez, o atributo `b` assume vários valores durante o tempo de vida da versão, `10, 5, 10 e 8`.

4.2.3.1 Restrições Temporais

Devido à estrutura de inclusão entre os elementos com rótulos temporais: atributos dentro de versões e versões dentro de objetos, é necessário que se estabeleçam restrições temporais entre os tempos de validade entre esses diversos rótulos. Tais restrições são apresentadas no modelo TVM, sendo aqui adaptadas para o modelo cTVM como segue:

- o tempo de validade de todas as versões está incluso no tempo de validade do objeto versionado que as engloba;

- o tempo de validade de cada atributo deve estar contido dentro do tempo de validade da versão que o define;

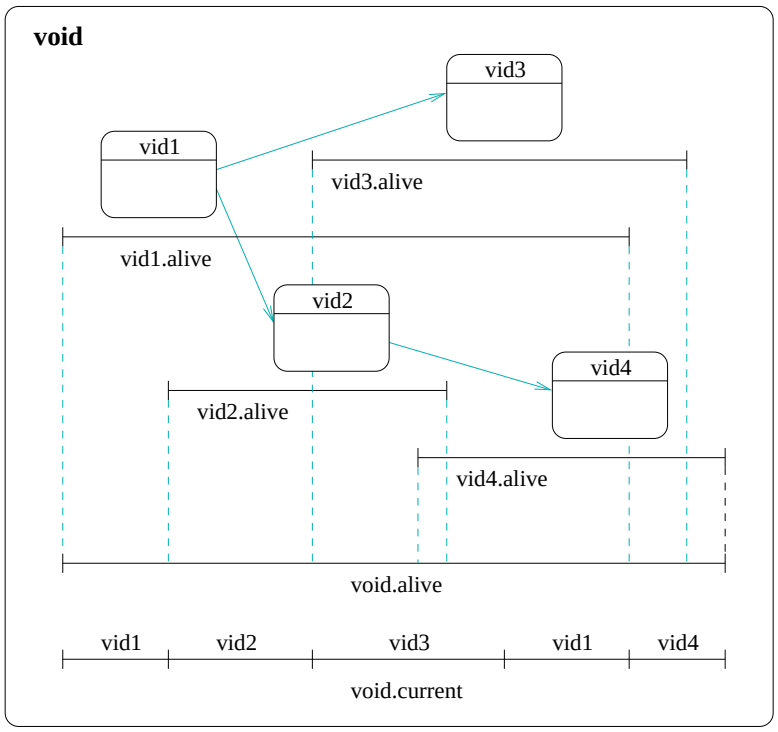

Figura 4.3: Estrutura de um objeto versionado temporal.

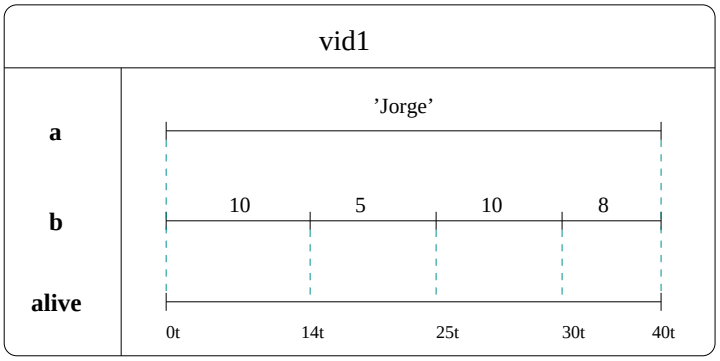

Figura 4.4: Estrutura de uma versão temporal.

- se a versão x deriva diretamente de y, então o início do tempo de validade de x está dentro do tempo de validade de y.

Apesar de atributos não possuírem rótulos temporais explícitos, como já apresentado, devem ser tratados como elementos temporais. A formalização de cTVQL provê uma representação adequada para que se possa expressar a relação de integridade temporal.

4.2.4 Linguagem cTVQL

Apesar do modelo de dados TVM seguir a estruturação orientada a objetos, a linguagem de consulta TVQL é baseada na linguagem de consulta SQL do modelo relacional. Tal peculiaridade constitui uma dificuldade relevante para relacionar a linguagem de consulta e o modelo de dados. Para compatibilizar estes elementos, em cTVQL optou-se por utilizar a mesma filosofia de projeto da linguagem de consulta OQL. Isto significa que cTVQL mantém uma sintaxe para realização de consultas próxima à estrutura `select-from-where` de SQL mas é diferente no sentido de ser definida composicionalmente.

Na Figura 4.5 é apresentada a sintaxe abstrata de cTVQL, sendo as modificações em relação a cVOQL marcadas por um asterisco "*".

A linguagem cTVQL adiciona suporte a representação de instantes e intervalos assim como operações de comparação entre instantes (`<`, `>`, `<=`, `>=`, `<>`, `=`). São introduzidas também as operações de comparação entre intervalos `intersect`, `overlap` e `equal`, que resultam em `true` se os intervalos se interseccionam em algum ponto do tempo, se o primeiro intervalo contém o segundo intervalo ou se ambos são iguais, respectivamente. As operações `after`, `before`, `into` recebem como parâmetro um instante e um intervalo e indicam se o instant está localizado *antes* do intervalo, *após* o intervalo ou *dentro* do intervalo, respectivamente.

Além da expansão dos valores básicos da linguagem, existem construções relacionadas diretamente à questões de ordem temporal. A sintaxe $\texttt{q}_1$ `/t/` $\texttt{q}_2$ constrói consultas que resultam em q_1 até o instante t e em q_2 a partir de t. O elemento `now` serve como uma referência para o instante de tempo atual do banco de dados. O operador `at` permite que se acesse qualquer valor em um ponto específico do tempo. Os operadores `valid`, `ever`, `everb` e `present` possuem um papel central na definição de cTVQL, sendo explicados com maior detalhe posteriormente.

Na Figura 4.6 estão apresentados alguns exemplos de consultas cTVQL.

4.3 Semântica Formal de cTVQL

Nesta seção é apresentada a semântica formal da linguagem cTVQL. Inicialmente são descritos os ambientes que formalizam o estado de banco de dados cTVM a partir de rótulos e valores temporais. Após é apresentada a relação de redução entre consultas.

4.3.1 Formalização do estado de banco de dados

Nesta seção são apresentados os ambientes que compõem um estado de banco de dados cTVM.

q	$\in$	*Query*	
q	::=	b	booleanos
	|	i	inteiros
	|	s	*strings*
	|	t	* instante temporal
	|	u	* intervalo temporal
	|	*void*	identificadores de objeto
	|	*vid*	identificadores de versão
	|	`nill`	valor indefinido
	|	$\{ q_1, \ldots, q_n \}$	multiconjuntos de consultas, $n \geq 0$
	|	(q_1 , q_2)	pares de consultas
	|	$q_1 \ /t/ \ q_2$	* soma temporal
	|	`now`	* instante atual
	|	x	variável
	|	q_1 *bop* q_2	operação binária
	|	*uop* q_1	operação unária
	|	$q.a$	acesso a atributo
	|	q`->`a	acesso a elementos de atributo
	|	q`->versions`	acesso a todas as versões
	|	`if` q_1 `then` q_2 `else` q_3	condicional
	|	`select` q_a	
		`from` $q_1\ x_1, \ldots, q_n\ x_n$	seleção, $n \geq 0$
		`where` q_b	
bop	::=	$+ \mid - \mid *$	
	|	$< \mid <= \mid > \mid >= \mid = \mid <>$	
	|	`and` | `or`	
	|	`union` | `intersection` | `difference`	
	|	`is_succ` | `is_pred`	
	|	`intersect` | `overlap` | `equal`	* comparações entre intervalos
	|	`before` | `into` | `after`	* comparações entre instantes e intervalos
	|	`at`	* valor em tempo específico
uop	::=	`not` | $-$ | `set` | `size`	
	|	`is_root` | `is_leaf` | `is_current`	
	|	`first` | `second`	
	|	`ever` | `everb` | `present` | `valid`	* operadores temporais
v	$\in$	*Values*	
v	::=	b	booleanos
	|	i	inteiros
	|	s	*strings*
	|	t	* instante temporal
	|	u	* intervalo temporal
	|	*vid*	identificadores de versão
	|	`nill`	valor indefinido
	|	$\{ v_1, \ldots, v_n \}$	multiconjuntos de valores $n \geq 0$
	|	(v_1 , v_2)	pares de valores
	|	$v_1 \ /t/ \ v_2$	* soma temporal de valores

Figura 4.5: Sintaxe abstrata de cTVQL.

```
// Retorna os nomes atuais de pessoas
// cujos registros estiveram ativos durante o período [0t..10t]
present (
select x.nome
from Pessoas x
where valid(x) intersect [0t..10t]
)

// Retorna os nomes atuais das pessoas que, em algum ponto do tempo,
// moraram no endereço 'Rua Jau 90'
present(
select x
from Pessoas x
where everb (x.endereco = 'Rua Jau 90')
)
```

Figura 4.6: Exemplos de consultas cTVQL.

4.3.1.1 Ambiente de extensões

O ambiente de extensões de cTVM é similar ao ambiente de extensões de cVODM, sendo uma função de identificadores de extensão para conjuntos de identificadores de objetos.

$$tee \in TEE = ExtentId \xrightarrow{fin} \mathcal{P}(Void)$$

4.3.1.2 Ambiente de objetos

A seguir é apresentada a formação de um ambiente de objetos versionados e temporais.

$$\begin{array}{rcl} tvoe & \in & TVOE = Void \xrightarrow{fin} ClassId \times Tvobjstate \times Interval \\ tvobjstate & \in & Tvobjstate = \mathcal{P}(Vid) \times Current \times \mathcal{P}(Vid \times Vid) \\ tvobjstate & ::= & (versions,\ current,\ deriv) \\ current & \in & Current \\ current & ::= & vid \mid \texttt{nill} \mid current_1\ /t/\ current_2 \end{array}$$

Na imagem das funções *tvoe* é adicionado um componente que representa o intervalo de validade do objeto. O estado de um objeto versionado temporal é composto por três elementos:

- *versions* é o conjunto de todas as versões do objeto versionado;

- *current* é a versão corrente do objeto, no caso um valor temporal que pode ser nulo;
- *deriv* representa a relação de derivação entre as versões do objeto versionado;

Uma diferença importante entre os modelos cVODM e cTVQL é que a versão corrente do objeto pode variar ao longo do tempo, levando à necessidade de se redefinir a formação do elemento *current*. Fora do tempo de validade do objeto versionado, a versão corrente deve ser necessariamente nula, sendo isto garantido pela relação de validade.

4.3.1.3 Ambiente de versões

A seguir é apresentada a formação de um ambiente de versões temporais. A principal diferença é a inclusão do intervalo de validade da versão à imagem de TVE.

$$\begin{array}{lcll} tve & \in & TVE & = & Vid \xrightarrow{fin} ClassId \times TversionState \times Interval \\ tvstate & \in & TversionState & = & Ident \rightarrow Tstored \end{array}$$

Versões temporais, assim como no modelo cVODM, também podem referenciar objetos, sendo necessário que se estabeleça o conjunto $Tstored$ de elementos armazenáveis.

$$\begin{array}{lcll} k & \in & Tstored & \\ k & ::= & b & \text{booleanos} \\ & | & i & \text{inteiros} \\ & | & s & \text{strings} \\ & | & void & \text{identificadores de objeto} \\ & | & vid & \text{identificadores de versão} \\ & | & \{k_1, \ \ldots, \ k_n\} & \text{multiconjuntos} \quad n \geq 0 \\ & | & (k_1, k_2) & \text{pares} \\ & | & k_1 \ /t/ \ k_2 & \text{soma temporal} \end{array}$$

4.3.2 Estado de banco de dados cTVM

Um estado de banco de dados cTVM é um tripla composta por um ambiente de extensões, um ambiente de objetos e um ambiente de versões.

$$tvdbstate \in TVDBState = TEE \times TVOE \times TVE$$

4.3.3 Relação de redução

Assim como nos modelos anteriores, o ambiente para a redução entre consultas cTVQL inclui o estado de banco de dados. Entretanto, devido à existência de uma referência ao tempo corrente na linguagem (`now`), é necessário que se especifique em que ponto do tempo a consulta está sendo feita. Assim sendo, a redução da consulta q para a consulta q' sob o banco de dados $tee; tvoe; tve$ e no tempo t é denotada

$$tee; tvoe; tve; t \vdash q \rightarrow q'$$

As regras de redução são apresentadas nas Figuras 4.7 e 4.8. No que segue, são explicadas as modificações em relação à semântica de cVOQL. As regras novas da semântica operacional são indicadas por um "*".

Assim como em cVOQL, para que a regra TS-Ctx conduza a avaliação de subconsultas é necessário que se expanda o contexto de avaliação como apresentado a seguir.

$$\mathcal{E} ::= \ldots \mid (\mathcal{E}\ /t/\ q) \mid (v\ /t/\ \mathcal{E})$$

A regra TS-Now simplesmente substitui o elemento `now` pelo instante no qual a consulta está sendo feita.

As regras TS-DotT, TS-ArrowT, TS-VersionsT, TS-IfT, TS-SelectT1 e TS-SelectT2 simplesmente dividem a avaliação pelos dois ramos temporais.

As regras TS-Binop e TS-Unop, em cTVQL, merecem uma atenção maior, visto que grande parte do comportamento temporal envolve operadores definidos através das funções η_b e η_u. Todos os antigos operadores, como soma de inteiros e união de multiconjuntos ganham semântica temporal, isto é, passam a valer por toda a faixa de valores de t. Por exemplo,

```
4 * 2 = 8
(6 /5t/ 7) + 4 = (10 /5t/ 11)
(3 /0t/ 4) - (2 /1t/ 1) = (1 /0t/ 2 /1t/ 3)
(3 /5t/ 5) + (2 /5t/ 0) = 5
```

Também são introduzidas operações de comparação que envolvem instantes e intervalos temporais. Por exemplo,

```
2t  before [10t..20t] = true
20t before [0t..]     = false
20t after  [..25t]    = false
5t  into   [..]       = true
```

Essas operações de comparação possuem comportamento similar às outras operações quando os valores são modificados ao longo do tempo, como apresentado a seguir:

```
[20t..10t] equal     [10t..20t] = true
[..1t]     intersect [0t..]     = true
[..25t]    overlap   [..0t]     = true
```

As operações `at`, `valid`, `present` e `ever`, entretanto, dizem respeito diretamente às características temporais do modelo e serão apresentadas com mais detalhes nas próximas subseções.

$$\frac{tvee; tvoe; tve; t \vdash q \rightarrow q'}{tvee; tvoe; tve; t \vdash \mathcal{E}[q] \rightarrow \mathcal{E}[q']} \quad \text{(TS-Ctx)}$$

$$\frac{tvoe(void) = (c, (versions, current, deriv), alive)}{tvee; tvoe; tve; t \vdash void \rightarrow current} \quad \text{(TS-Void)}$$

$$tvee; tvoe; tve; t \vdash \texttt{now} \rightarrow t \quad \text{(* TS-Now)}$$

$$\frac{tvee(x) = \{\overrightarrow{void}\}}{tvee; tvoe; tve; t \vdash x \rightarrow \{\overrightarrow{void}\}} \quad \text{(TS-Var)}$$

$$\frac{\eta_b(bop, v_1, v_2) = v_3}{tvee; tvoe; tve; t \vdash v_1 \ bop \ v_2 \rightarrow v_3} \quad \text{(TS-Binop)}$$

$$\frac{\eta_u(uop, v_1) = v_2}{tvee; tvoe; tve; t \vdash uop \ v_1 \rightarrow v_2} \quad \text{(TS-Unop)}$$

$$\frac{tve(vid) = (c, tvstate, alive) \qquad tvstate(a) = q}{tvee; tvoe; tve; t \vdash vid.a \rightarrow q} \quad \text{(TS-Dot1)}$$

$$tvee; tvoe; tve; t \vdash \texttt{nill}.m \rightarrow \texttt{nill} \quad \text{(TS-Dot2)}$$

$$tvee; tvoe; tve; t \vdash (v_1 \ /t/ \ v_2).m \rightarrow (v_1.m) \ /t/ \ (v_2.m) \quad \text{(* TS-DotT)}$$

$$tvee; tvoe; tve; t \vdash \{vid, \vec{v}\}\texttt{->}a \rightarrow vid.a \ \texttt{union} \ \{\vec{v}\}\texttt{->}a \quad \text{(TS-Arrow1)}$$

$$tvee; tvoe; tve; t \vdash \{\texttt{nill}, \vec{v}\}\texttt{->}a \rightarrow \{\vec{v}\}\texttt{->}a \quad \text{(TS-Arrow2)}$$

$$tvee; tvoe; tve; t \vdash \{\}\texttt{->}a \rightarrow \{\} \quad \text{(TS-Arrow3)}$$

$$tvee; tvoe; tve; t \vdash \texttt{nill->}a \rightarrow \texttt{nill} \quad \text{(TS-Arrow4)}$$

$$tvee; tvoe; tve; t \vdash (v_1 \ /t/ \ v_2)\texttt{->}m \rightarrow (v_1\texttt{->}m) \ /t/ \ (v_2\texttt{->}m) \quad \text{(* TS-ArrowT)}$$

$$\frac{tvoe(void) = (c, (versions, current, deriv), alive) \qquad vid \in versions}{tvee; tvoe; tve; t \vdash \{vid, \vec{v}\}\texttt{->versions} \rightarrow versions \ \texttt{union} \ \{\vec{v}\}\texttt{->versions}} \quad \text{(TS-Versions1)}$$

$$tvee; tvoe; tve; t \vdash \{\texttt{nill}, \vec{v}\}\texttt{->versions} \rightarrow \{\vec{v}\}\texttt{->versions} \quad \text{(TS-Versions2)}$$

$$tvee; tvoe; tve; t \vdash \{\}\texttt{->versions} \rightarrow \{\} \quad \text{(TS-Versions3)}$$

$$tvee; tvoe; tve; t \vdash \texttt{nill->versions} \rightarrow \texttt{nill} \quad \text{(TS-Versions4)}$$

$$tvee; tvoe; tve; t \vdash (v_1 \ /t/ \ v_2)\texttt{->versions} \rightarrow v_1\texttt{->versions} \ /t/ \ v_2\texttt{->versions} \quad \text{(* TS-VersionsT)}$$

Figura 4.7: Semântica operacional de cTVQL.

$$tvee; tvoe; tve; t \vdash \texttt{if true then } q_1 \texttt{ else } q_2 \rightarrow q_1 \qquad \text{(TS-IF1)}$$

$$tvee; tvoe; tve; t \vdash \texttt{if false then } q_1 \texttt{ else } q_2 \rightarrow q_2 \qquad \text{(TS-IF2)}$$

$$tvee; tvoe; tve; t \vdash \texttt{if nill then } q_1 \texttt{ else } q_2 \rightarrow \texttt{nill} \qquad \text{(TS-IF3)}$$

$$tvee; tvoe; tve; t \vdash \texttt{if } (v_1 \ /t/ \ v_2) \texttt{then } q_1 \texttt{ else } q_2 \rightarrow (\texttt{if } v_1 \texttt{ then } q_1 \texttt{ else } q_2) \ /t/ \ (\texttt{if } v_2 \texttt{ then } q_1 \texttt{ else } q_2) \qquad \text{(* TS-IFT)}$$

$$tvee; tvoe; tve; t \vdash \texttt{select } q_a \texttt{ from } \{v_1, \ldots, v_m\} \ x_1, \ldots, q_n \ x_n \texttt{ where } q_b \rightarrow (\texttt{select } q_a \texttt{ from } q_2 \ x_2, \ldots, q_n \ x_n \texttt{ where } q_b)[x_1 ::= v_1] \ \texttt{union} \ \texttt{select } q_a \texttt{ from } \{v_2, \ldots, v_m\} \ x_1, \ldots, q_n \ x_n \texttt{ where } q_b \qquad \text{(TS-SELECT1)}$$

$$tvee; tvoe; tve; t \vdash \texttt{select } q_a \texttt{ from } \{\} \ x_1, \ldots, q_m \ x_n \texttt{ where } q_b \rightarrow \{\} \qquad \text{(TS-SELECT2)}$$

$$tvee; tvoe; tve; t \vdash \texttt{select } q_a \texttt{ from nill } x_1, \ldots, q_m \ x_n \texttt{ where } q_b \rightarrow \texttt{nill} \qquad \text{(TS-SELECT3)}$$

$$tvee; tvoe; tve; t \vdash \texttt{select } q_a \texttt{ from } (v_1 \ /t/ \ v_2) \ x_1, \ldots, q_n \ x_n \texttt{ where } q_b \rightarrow (\texttt{select } q_a \texttt{ from } v_1 \ x_1, \ldots, q_n \ x_n \texttt{ where } q_b) \ /t/ \ (\texttt{select } q_a \texttt{ from } v_2 \ x_1, \ldots, q_n \ x_n \texttt{ where } q_b) \qquad \text{(* TS-SELECTT1)}$$

$$tvee; tvoe; tve; t \vdash \texttt{select } q_a \texttt{ from where true} \rightarrow \{q_a\} \qquad \text{(TS-SELECT4)}$$

$$tvee; tvoe; tve; t \vdash \texttt{select } q_a \texttt{ from where false} \rightarrow \{\} \qquad \text{(TS-SELECT5)}$$

$$tvee; tvoe; tve; t \vdash \texttt{select } q_a \texttt{ from where nill} \rightarrow \{\} \qquad \text{(TS-SELECT6)}$$

$$tvee; tvoe; tve; t \vdash \texttt{select } q_a \texttt{ from where } (v_1 \ /t/ \ v_2) \rightarrow (\texttt{select } q_a \texttt{ from where } v_1) \ /t/ \ (\texttt{select } q_a \texttt{ from where } v_2) \qquad \text{(* TS-SELECTT2)}$$

Figura 4.8: Semântica operacional de cTVQL (continuação).

4.3.3.1 Semântica do operador `at`

O operador `at` retorna um valor fixo a partir de um valor temporal qualquer e um instante de tempo.

$$
\begin{array}{rcl}
b \ \texttt{at}\ t & = & b \\
s \ \texttt{at}\ t & = & s \\
i \ \texttt{at}\ t & = & i \\
t_1 \ \texttt{at}\ t & = & t_1 \\
u \ \texttt{at}\ t & = & u \\
(v_1, v_2) \ \texttt{at}\ t & = & (v_1 \ \texttt{at}\ t)\ ,\ (v_2 \ \texttt{at}\ t) \\
\{v_1, \ldots, v_n\} \ \texttt{at}\ t & = & \{\ (v_1 \ \texttt{at}\ t),\ \ldots,\ (v_1 \ \texttt{at}\ t)\ \} \\
(v_1\ /t_1/\ v_2) \ \texttt{at}\ t & = & \begin{cases} v_1 \ \texttt{at}\ t & \text{se } t < t_1 \\ v_2 \ \texttt{at}\ t & \text{se } t \geq t_1 \end{cases}
\end{array}
$$

Caso o segundo parâmetro do operador varie temporalmente, o resultado segue a sua variação.

```
(3 /0t/ 4 /10t/ 5) at (-4t /15t/ 8t) = (3 /15t/ 4)
```

O operador `at` é especialmente importante em cTVQL, pois é utilizado na definição de outros operadores temporais, como será apresentado.

4.3.3.2 Semântica do operador `valid`

Em cTVQL, um valor é sempre definido em todos os instantes temporais possíveis. Através da sintaxe utilizada para representar as mudanças de valores, pode-se subdividir um determinado valor temporal em intervalos nos quais um dado valor não é modificado Por exemplo, o valor temporal

$$v = \texttt{5 /10t/ 4 /20t/ 0}$$

é dividido em três intervalos específicos, um para cada diferente resultado de v `at` t. A operação `valid` recebe como argumento um valor qualquer e retorna a fragmentação da linha do tempo que ele causa. Desta forma,

$$\texttt{valid (5 /10t/ 4 /20t/0)} = \texttt{[..9] /10t/ [10..19] /20t/ [20..]}$$

4.3.3.3 Semântica do operador `present`

Em cTVQL, o operador `present` é simplesmente um *alias* para `at now`. Por exemplo,

```
present (10 /0t/ 11) =  (10 /0t/ 11) at now
```

4.3.3.4 Semântica dos operadores `ever` *e* `everb`

Em TVQL, a palavra reservada `ever` pode ser utilizada em dois contextos. Em uma das situações `ever` é aplicado sobre expressões booleanas e indica que a condição é verdadeira em algum ponto do tempo. Por exemplo, em TVQL a seguinte consulta retorna todos os nomes atuais de pessoas que, em algum ponto do tempo, tiveram `3333-4444` como número de telefone.

```
select x.nome
from Pessoas x
where ever x.telefone = '3333-4444'
```

A outra situação ocorre é quando `ever` é aplicado na parte `select` e representa que o resultado da consulta é a união de todos os resultados ao longo do tempo. Por exemplo, a seguinte consulta retorna todos os nomes das pessoas que já tiveram o número de telefone `3333-4444`. Note que é retornado sempre o nome que elas tinham na época que a condição é verdadeira.

```
select ever x.nome
from Pessoas x
where x.telefone = '3333-4444'
```

Em cTVQL se designará uma sintaxe específica para cada uso de `ever`, permitindo que a semântica da construção seja mais facilmente identificada. O operador `ever` representa a semântica em TVQL da palavra reservada `ever` na parte `select` da consulta. Já o operador `everb` representa a semântica da palavra reservada `ever` na parte `where`. A vantagem de se dividir sintaticamente é a mesma obtida pela separação do significado do operador '.', pois garante o mesmo significado para cada operador independentemente da posição que se encontre na consulta.

A semântica do operador `ever` de cTVQL, aplicado sobre coleções, pode ser descrita como:

$$\texttt{ever}\ v = \bigcup_{t \in Time} v\ \texttt{at}\ t$$

A seguir são apresentados os seguintes exemplos da aplicação do operador `ever`.

```
ever {} /5t/ nill = nill
ever {1} /8t/ {4} = {1,4}
ever {  1 /4t/ 5 , 3 /8t/ 5  } = { 1,3, 5,3, 5,5 }
ever {  0 /0t/ 5 /5t/ 10 , 5 /0t/ 0 /5t/ 10 } = { 0,5, 10,10 }
```

É interessante atentar para a semântica da repetição de elementos. Pode-se pensar na semântica de `ever` como duas fases: na primeira são gerados todos os diferentes resultados de valores possíveis pelo operador `at`, e sobre este valor, é aplicado a operação de `union` quando há mudança no resultado. Considerando o último exemplo:

```
ever { 0 /0t/ 5 /5t/ 10 , 5 /0t/ 0 /5t/ 10 } =
1) gerar resultados: {0,5} /0t/ {5,0} /5t/ {10,10} =
2) simplificar:      {0,5} /5t/ {10,10} =
3) aplicar union:    {0,5,10,10}
```

O operador `everb` opera sobre valores booleanos e possui a seguinte semântica:

$$\texttt{everb}\ v = \begin{cases} \texttt{true} & \text{se } \exists t(v\ \texttt{at}\ t = \texttt{true}) \\ \texttt{false} & \text{se } \nexists t(v\ \texttt{at}\ t = \texttt{true}) \end{cases}$$

É importante notar que, para se manter a mesma semântica que possui em TVQL, em cTVQL o operador `ever` deve ser aplicado externamente à construção `select-from-where`. Uma consulta TVQL da forma

```
select ever p.nome
from Pessoas p
where p.endereco = 'Rua Jau 98'
```

é escrita em cTVQL como

```
ever (
  select p.nome
  from Pessoas p
  where p.endereco = 'Rua Jau 98'
)
```

4.3.3.5 Equivalência entre valores

Ao se estabelecer regras sintáticas para a formação de consultas, é importante que se consiga definir de forma consistente a informação de *equivalência* entre elementos. Por exemplo, apesar de sintaticamente {1,2,3} ser diferente de {2,3,1}, ambos representam o mesmo elemento matemático. O conceito de equivalência é a base para o operador '=' e só não foi abordado anteriormente, nas linguagens cOQL e cVOQL, porque a equivalencia semântica entre representações de pares e conjuntos é bem conhecida. Entretanto, ao adicionarmos as construção v_1 $/t/$ v_2, tal conceito não é trivial. Por exemplo, o que se deve concluir quando há o registro de uma modificação mas não há alteração no valor em um dado ponto da linha do tempo. O valor 1 $/0t/$ 2 $/10t/$ 2 é equivalente a 1 $/0t/$ 2 ?

Para resolver tal questionamento, deve ser definida formalmente a relação de equivalência entre valores cTVQL. Um valor v_1 é considerado equivalente a v_2 se, e somente se existe uma árvore de derivação que prove isso a partir das regras da Figura 4.9.

As regras Divisão em Temp e Estufamento em Temp são importantes para se individualizar um valor semântico que pode ser apresentado sob vários formatos diferentes, como por exemplo

```
5   =   5 /6t/ 5  =  5 /6t/ (23 /0t/ 5)
```

A formalização da relação de equivalência também é importante para impedir que se confundam elementos que podem parecer equivalentes apesar de não o serem. Por exemplo, se fosse considerado somente os resultados do operador `at` poderia-se ter a impressão que os seguintes valores são duas representações sintáticas do mesmo elemento semântico.

```
{true,false} /0t/ {false,true}  =  { true/0t/false, false/0t/true } ?
```

Entretanto, basta avaliar o resultado da seguinte consulta sobre eles para perceber que não são efetivamente equivalentes:

```
present (
  select x
  from ({true,false} /0t/ {false,true})  x
  where everb x )
```

$$v \equiv v \qquad \text{(Reflexidade)}$$

$$\frac{v_1 \equiv v_2 \qquad v_2 \equiv v_3}{v_1 \equiv v_3} \qquad \text{(Transitividade)}$$

$$\frac{v_1 \equiv v_2}{v_2 \equiv v_1} \qquad \text{(Simetria)}$$

$$\{\vec{v_a}, v_1, v_2, \vec{v_b}\} \equiv \{\vec{v_a}, v_2, v_1, \vec{v_b}\} \qquad \text{(Permutação em Bag)}$$

$$\frac{v \equiv v'}{\{\vec{v_a}, v, \vec{v_b}\} \equiv \{\vec{v_a}, v', \vec{v_b}\}} \qquad \text{(Substituição em Bag)}$$

$$\frac{v_1 \equiv v_1'}{v_1, v_2 \equiv v_1', v_2} \qquad \text{(Substituição1 em Par)}$$

$$\frac{v_2 \equiv v_2'}{v_1, v_2 \equiv v_1, v_2'} \qquad \text{(Substituição2 em Par)}$$

$$v_1 \ /t_1/ \ (v_2 \ /t_2/ \ v_3) \equiv (v_1 \ /t_1/ \ v_2) \ /t_2/ \ v_3) \qquad \text{(Associatividade em Temp)}$$

$$\frac{v \equiv v'}{v \ /t/ \ v' \equiv v} \qquad \text{(Divisão em Temp)}$$

$$\frac{t_1 \geq t_2}{v_1 \ /t_1/ \ (v_2 \ /t_2/ \ v_3) \equiv v_1 \ /t_1/ \ v_3} \qquad \text{(Estufamento em Temp)}$$

$$\frac{v_1 \equiv v_1'}{v_1 \ /t/ \ v_2 \equiv v_1' \ /t/ \ v_2} \qquad \text{(Substituição1 em Temp)}$$

$$\frac{v_2 \equiv v_2'}{v_1 \ /t/ \ v_2 \equiv v_1 \ /t/ \ v_2'} \qquad \text{(Substituição2 em Temp)}$$

Figura 4.9: Equivalência entre valores cTVQL.

```
present (
  select x
  from { true/0t/false, false/0t/true } x
  where everb x )
```

Supondo `now = 10t`, a primeira consulta tem como resultado {`true`} enquanto a segunda gera {`true,false`}.

4.4 Sistema de tipos de cTVQL

Esta seção apresenta o sistema de tipos de cTVQL. O conjunto de tipos de consultas cTVQL conta com a adição dos tipos que representam intervalos e instantes temporais, como reapresentado a seguir.

$$\begin{array}{lll} \sigma & \in & Type \\ \sigma & ::= & Int \mid Bool \mid String \mid Time \mid Interval \mid c \mid Bag(\sigma) \mid \sigma_1 * \sigma_2 \end{array}$$

4.4.1 Formalização de esquemas de dados

A formalização de esquemas de banco de dados do modelo cTVM não apresenta nenhuma modificação em relação ao modelo cVODM além das modificações sobre o conjunto $Type$.

$$\begin{array}{llll} p & \in & ClassId_{\perp} & = & ClassId \cup \{\perp\} \\ tsch & \in & Tvschema & = & ClassId \xrightarrow{fin} Classdef \\ & & Classdef & = & ClassId_{\perp} \times ExtentId \times Ty \\ ty & \in & Ty & = & Ident \xrightarrow{fin} Type \end{array}$$

4.4.2 Julgamentos de tipos

Um julgamento de tipos de consultas cTVQL possui a forma

$$tsch; tee; tvoe; tve; \Gamma \vdash q : \sigma$$

onde

- $tsch$ é um *esquema de dados* pertencente ao conjunto $Vschema$ definido na Seção 4.4.1;
- tee é um ambiente de extensões pertencente ao conjunto TEE definido na Seção 4.3.1.1;
- $tvoe$ é um ambiente de objetos pertencente ao conjunto $TVOE$ definido na Seção 4.3.1.2;
- tve é um ambiente de versões pertencente ao conjunto TVE, definido na Seção 4.3.1.3;

$$K \vdash b : Bool \qquad \text{(TT-Bool)}$$

$$K \vdash i : Int \qquad \text{(TT-Int)}$$

$$K \vdash s : String \qquad \text{(TT-Str)}$$

$$K \vdash t : Time \qquad \text{(* TT-Time)}$$

$$K \vdash u : Interval \qquad \text{(* TT-Interval)}$$

$$\frac{tvoe(void) = (c, (versions, current, deriv), alive)}{K \vdash void : c} \qquad \text{(TT-Void)}$$

$$\frac{tve(vid) = (c, tvstate, alive)}{K \vdash vid : c} \qquad \text{(TT-Vid)}$$

$$K \vdash \texttt{nill} : \sigma \qquad \text{(TT-Nill)}$$

$$\frac{K \vdash q_1 : \sigma \quad \ldots \quad K \vdash q_n : \sigma \qquad n \geq 0}{K \vdash \{q_1, \ldots, q_n\} : Bag(\sigma)} \qquad \text{(TT-Bag)}$$

$$\frac{K \vdash q_1 : \sigma_1 \qquad K \vdash q_2 : \sigma_2}{K \vdash (q_1 \ ; \ q_2) : \sigma_1 * \sigma_2} \qquad \text{(TT-Pair)}$$

$$\frac{K \vdash q_1 : \sigma \qquad K \vdash q_2 : \sigma}{K \vdash (q_1 \ /t/ \ q_2) : \sigma} \qquad \text{(* TT-Temporal)}$$

$$K \vdash \texttt{now} : Time \qquad \text{(* TT-Now)}$$

$$\frac{\Gamma(x) = \sigma}{K \vdash x : \sigma} \qquad \text{(TT-Var)}$$

$$\frac{K \vdash q_1 : \sigma_1 \qquad K \vdash q_2 : \sigma_2 \qquad \delta_b(bop, \sigma_1, \sigma_2) = \sigma_3}{K \vdash q_1 \ bop \ q_2 : \sigma_3} \qquad \text{(TT-Binop)}$$

$$\frac{K \vdash q : \sigma_1 \qquad \delta_u(uop, \sigma_1) = \sigma_2}{K \vdash uop \ q : \sigma_2} \qquad \text{(TT-Unop)}$$

$$\frac{K \vdash q : c \qquad tsch(c) = (p, e, ty) \qquad ty(a) = \sigma}{K \vdash q.a : \sigma} \qquad \text{(TT-Dot)}$$

$$\frac{K \vdash q : Bag(c) \qquad tsch(c) = (p, e, ty) \qquad ty(a) = Bag(\sigma)}{K \vdash q\texttt{->}a : Bag(\sigma)} \qquad \text{(TT-Arrow)}$$

$$\frac{K \vdash q : Bag(c)}{K \vdash q\texttt{->versions} : Bag(c)} \qquad \text{(TT-Versions)}$$

$$\frac{K \vdash q_1 : Bool \qquad K \vdash q_2 : \sigma \qquad K \vdash q_3 : \sigma}{K \vdash \texttt{if } q_1 \texttt{ then } q_2 \texttt{ else } q_3 : \sigma} \qquad \text{(TT-If)}$$

$$\frac{\begin{array}{rcl} tsch; tee; tvoe; tve; \Gamma & \vdash & q_1 : Bag(\sigma_1) \\ tsch; tee; tvoe; tve; \Gamma, x_1 : \sigma_1 & \vdash & q_2 : Bag(\sigma_2) \\ & \vdots & \\ tsch; tee; tvoe; tve; \Gamma, x_1 : \sigma_1, \ldots, x_{n-1} : \sigma_{n-1} & \vdash & q_n : Bag(\sigma_n) \\ tsch; tee; tvoe; tve; \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n & \vdash & q_a : \sigma_a \\ tsch; tee; tvoe; tve; \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n & \vdash & q_b : Bool \\ x_1 \text{ a } x_n \text{ diferentes} \qquad x_1, \ldots, x_n \notin Dom(\Gamma) & & n \geq 0 \end{array}}{tsch; tee; tvoe; tve; \Gamma \vdash \texttt{select } q_a \texttt{ from } q_1 \ x_1 \ , \ldots, q_n \ x_n \texttt{ where } q_b : Bag(\sigma_a)} \qquad \text{(TT-Select)}$$

$$\frac{K \vdash q : \sigma_1 \qquad \sigma_1 <: \sigma_2}{K \vdash q : \sigma_2} \qquad \text{(TT-Sub)}$$

Figura 4.10: Sistema de tipos de cTVQL.

- Γ é um ambiente de tipos.

Na Figura 4.10 são apresentadas as regras do sistema de tipos de cTVQL.

Os axiomas TT-Time e TT-Interval dão os respectivos tipos a instantes e intervalos. O axioma TT-Now dá tipo *Time* à referência construção `now`.

A regra TT-Temporal dá tipos a construções formadas pela sintaxe q_1 `/t/` q_2 se ambos os componentes possuírem o mesmo tipo. Esta regra garante que valores temporais mantêm o mesmo tipo em todos os possíveis instantes temporais.

arg_1	arg_2	arg_3	resultado
`intersect`, `overlap`, `equal`	*Interval*	*Interval*	*Bool*
`before`, `into`, `after`	*Time*	*Interval*	*Bool*
`at`	σ	*Time*	σ

Figura 4.11: Expansão da função δ_b.

As funções δ_b e δ_u são expandidas como apresentado nas Figuras 4.11 e 4.12.

arg_1	arg_2	resultado
`ever`	$Bag(\sigma)$	$Bag(\sigma)$
`everb`	*Bool*	*Bool*
`valid`	σ	*Interval*
`present`	σ	σ

Figura 4.12: Expansão da função δ_u.

4.5 Segurança de tipos de cTVQL

Em cTVQL a segurança do sistema de tipos é assegurada através dos seguintes lemas.

Lema 4.1 *(Progresso em cTVQL) Se* $tsch; tee; tvoe; tve; \Gamma_{tsch} \vdash q : \sigma$ *e* $tsch \vdash_{tsch} (tee, tvoe, tve)$*, então ou* $q \in Values$ *ou existe* q' *tal que* $tee; tvoe; tve; t \vdash q \rightarrow q'$.

Lema 4.2 *(Preservação em cTVQL) Se* $tsch; tee; tvoe; tve; \Gamma_{tsch} \vdash q : \sigma$, $tee; tvoe; tve; t \vdash q \rightarrow q'$ *e* $tsch \vdash_{tsch} (tee, tvoe, tve)$*, então* $tsch; tee; tvoe; tve; \Gamma_{tsch} \vdash q' : \sigma$.

As provas dos lemas não serão apresentadas nesta dissertação, constando como um dos trabalhos futuros.

4.6 Considerações finais

Neste capítulo foram apresentados brevemente o modelo de dados cTVM e a linguagem de consulta cTVQL. Apesar de, por questões de tempo, não se ter conseguido concluir a prova de segurança do sistema de tipos, as mudanças estruturais na semântica operacional e no sistema de tipos são pequenas e tornam tal prova relativamente fácil.

As vantagens da abordagem proposta de lidar com representações sintáticas de funções é interessante pois acarreta em poucas modificações na semântica operacional e sistema de tipos, podendo ser aplicada a outras especificações que seguem este mesmo paradigma.

Outra contribuição é uma apresentação da semântica das palavras reservadas `ever` e `present`, permitindo que sejam aplicadas em pontos arbitrários da consulta. Defende-se que definir a linguagem de consulta de forma composicional, aos moldes de OQL, é uma abordagem mais adequada ao se considerar um modelo de dados orientado a objetos.

Há um trabalho recente (KICK, 2003) sobre a idéia de utilizar mônadas para representar a introdução de uma dimensão temporal em sistemas. Durante a especificação de cTVQL, tal *insight* foi utilizado como guia para algumas decisões de projeto. Em um primeiro momento, talvez a semântica de cTVQL possa ser considerada uma aplicação desta idéia sob um domínio específico. Esta relação, entretanto, necessita ser melhor investigada posteriormente.

5 IMPLEMENTAÇÕES

Neste capítulo são apresentados as implementações de interpretadores para as linguagens propostas nos capítulos anteriores.

5.1 Motivação

No desenvolvimento de modelos de bancos de dados, é comum que se descrevam novas construções com base em estruturas já existentes. Por exemplo, no modelo TVM o comportamento temporal é especificado por meio de classes abstratas que definem uma série de atributos e métodos. Uma vantagem direta desta abordagem é a rápida obtenção de uma implementação, sob a forma de um *middleware*. Em contrapartida, ao se descrever um determinado modelo de dados com mudanças significativas, nem sempre se obtém de forma fácil tal mapeamento para outro modelo.

Quando se dispõe da semântica operacional de uma dada linguagem, pode-se implementar facilmente protótipos de interpretadores utilizando uma linguagem de programação que siga o paradigma lógico, como Prolog, ou o paradigma funcional, como Lisp, ML ou Haskell. Tais protótipos baseados na semântica operacional, apesar de normalmente não serem muito eficientes, podem servir como plataforma de testes enquanto não se conclui um interpretador completo para as linguagens.

5.2 Arquitetura

Na Figura 5.1 é apresentada a arquitetura básica dos três programas. Consulta, esquema e estado de banco de dados são introduzidos no sistema através de campos de textos. Cada campo de texto é analisado léxica e sintaticamente, chegando-se a uma representação interna das estruturas. A partir disto são definidos três processos:

1. *Query Evaluator*, que retorna o resultado de um passo de avaliação sobre a consulta, se possível;

2. *Type Checker*, que determina o tipo mais específico da consulta apresentada, se possível;

3. *Integrity Checker*, que verifica se o estado do banco de dados está ou não consistente em relação ao esquema.

Os processos *Query Evaluator* e *Integrity Checker* correspondem, respectivamente, à transcrição da relação de redução e à verificação da relação de validade.

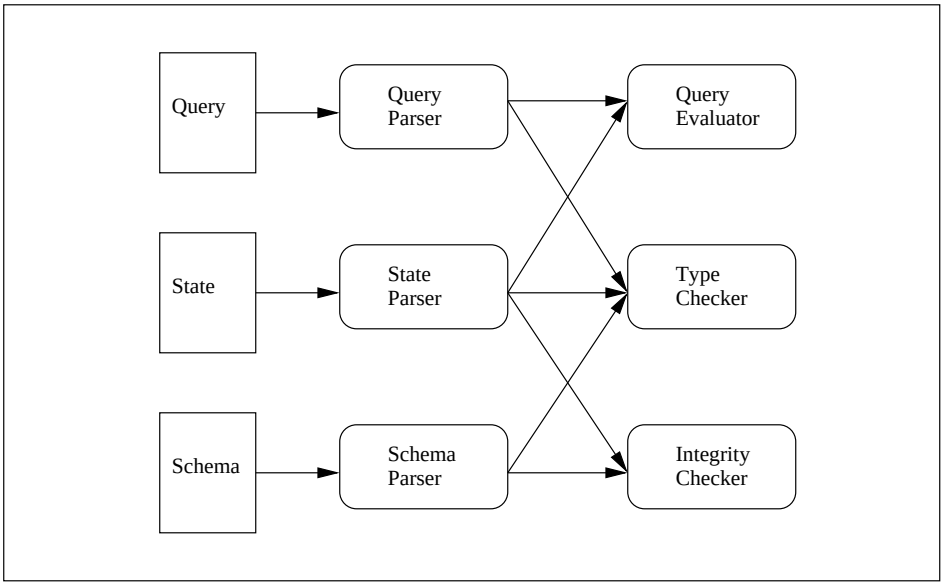

Figura 5.1: Arquitetura dos protótipos de interpretador.

No caso do processo *Type Checker*, entretanto, existe uma particularidade importante. Diferentemente da relação de redução, que é funcional, existem consultas que são associadas a diversos tipos de dados indistintamente. Por exemplo, o literal `nill` pode ser de qualquer tipo, dependendo do contexto. Nos três sistemas de tipos apresentados, não há como definir uma função que, a partir de `nill`, sempre retorne um tipo correto. A saída para se obter um avaliador de tipos funcional foi a adição de um tipo *botton* ($\perp$) e a utilização da operação de *least upper bound* nas regras de julgamento de tipo. As peculiaridades de tal solução, entretanto, fogem ao escopo deste trabalho e, portanto, não serão apresentadas em detalhe.

Os protótipos foram programados utilizando a linguagem Ocaml, uma extensão de ML. Para a geração de analisadores léxicos e sintáticos foram utilizadas as ferramentas `ocamllex` e `ocamlyacc`. A interface gráfica foi criada utilizando-se a biblioteca LiblTK, um *binding* de TK para Ocaml. Nas Figuras 5.2, 5.3 e 5.4 são apresentadas as interfaces das três implementações desenvolvidas. Existe disponibilidade de código executável para as plataformas Linux e Windows. Código fonte, licença, executáveis e instruções de uso estão disponíveis em `http://www.inf.ufrgs.br/ rma/ctvql`.

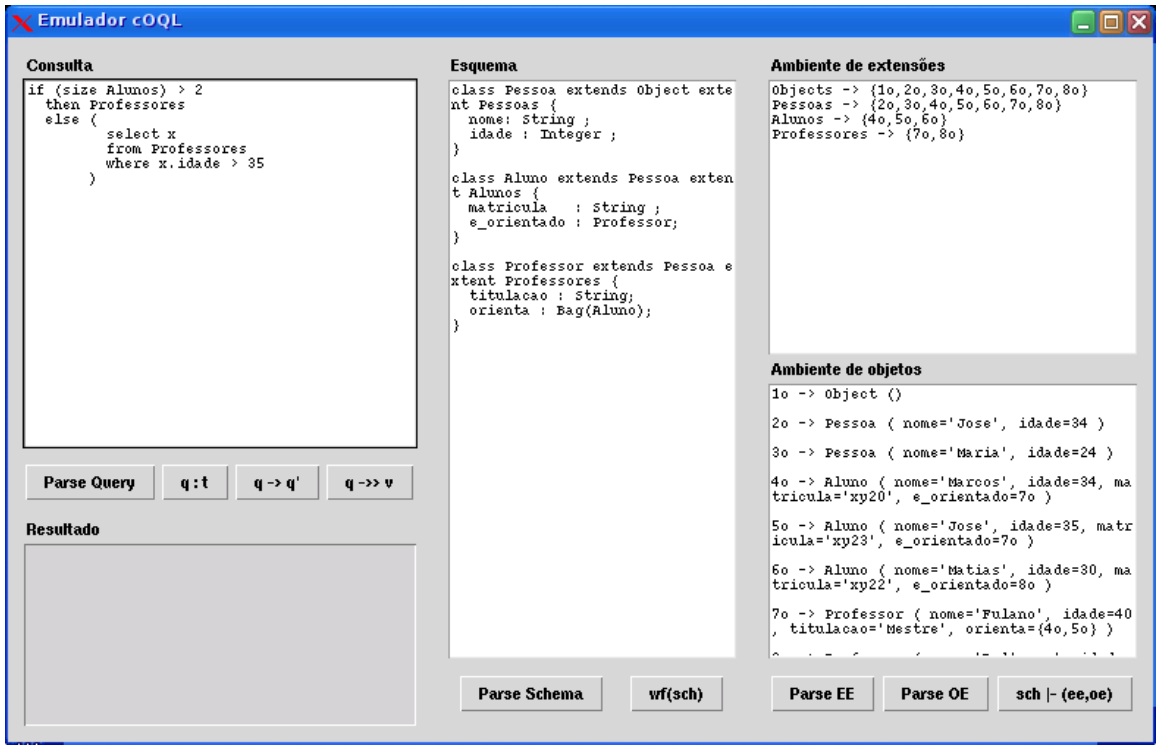

Figura 5.2: Interface do interpretador cOQL.

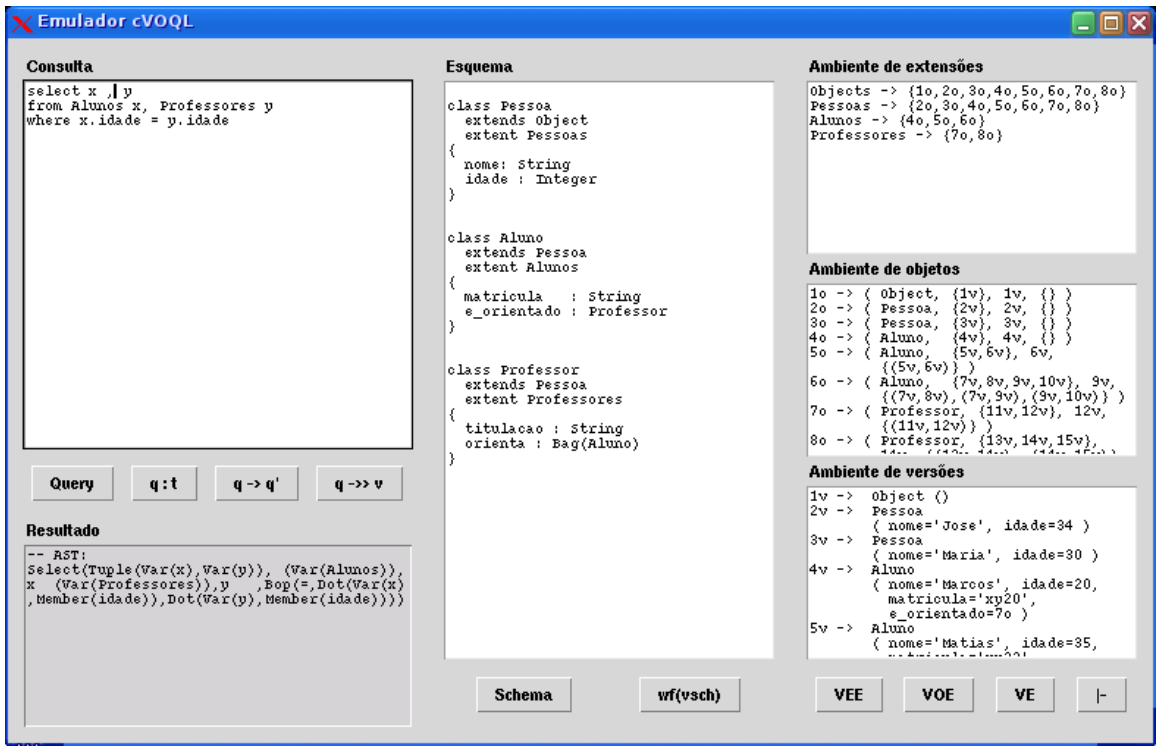

Figura 5.3: Interface do interpretador cVOQL.

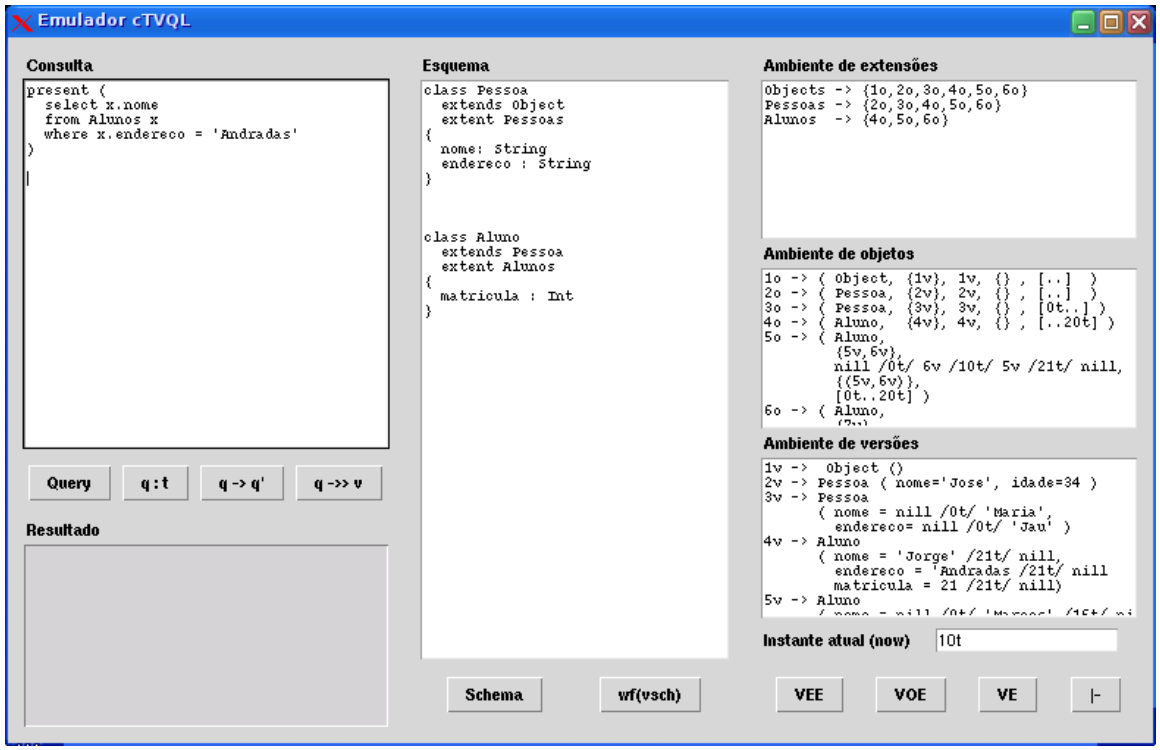

Figura 5.4: Interface do interpretador cTVQL.

6 CONCLUSÕES

Este trabalho definiu semântica operacional e sistema de tipos para um núcleo funcional da linguagem de consulta TVQL denominado cTVQL. Para tal, foi necessária a formalização de uma parte do modelo TVM, compreendendo tanto esquema quanto estado de banco de dados.

Devido à relativa carência de modelos formais estabelecidos para o tratamento de linguagens de consulta sobre bancos de dados orientados a objetos e à complexidade do modelo TVM, o processo de formalização da linguagem TVQL se desenvolveu de forma gradual. Inicialmente foram definidos semântica operacional e sistema de tipos para o subconjunto da linguagem relacionado ao paradigma orientado a objetos. Posteriormenete, a linguagem foi estendida com o suporte a objetos versionados e, por último, foram adicionadas características temporais. Esta organização da modelagem permitiu um melhor tratamento dos problemas de cada camada de abstração. Por exemplo, questões relativas à construções do modelo orientado a objeto, tais como a semântica do operador '.' ou de referências indefinidas não fossem tratadas específico em cTVQL por não dizer respeito às características temporais do modelo.

Neste trabalho defendeu-se a importância da propriedade de segurança do sistema de tipos. A segurança do sistema de tipos é considerada uma propriedade essencial de um bom sistema de tipos, garantindo ao programador que as consultas bem-tipadas retornam valores do tipo previsto e não geram erros em tempo de execução. Provou-se que os sistemas de tipos de cOQL, cVOQL são seguros em relação à semântica operacional.

Este trabalho defendeu fundamentalmente a aplicação de uma abordagem formal na pesquisa de modelos de dados e linguagens de consulta. Através de um *framework* como o apresentado no Capítulo 2 (cODM e cOQL), é possível que sejam exploradas várias extensões ao modelo, assim como um estudo formal de suas propriedades.

A seguir são enumeradas as principais contribuições deste trabalho:

1. definição de semântica operacional *small-step* e sistema de tipos para as linguagens cOQL, cVOQL e cTVQL;

2. verificação formal da propriedade de segurança dos sistema de tipos de cOQL e cVOQL;

3. implementação de protótipos de interpretadores para as linguagens cOQL, cVOQL e cTVQL.

4. proposta de representação de semântica temporal de modelos de dados e de uma representação para definição de valores temporais.

Neste trabalho foram realizadas inúmeras simplificações de modelos para possibilitar um melhor tratamento formal. Entretanto, alguns dos pontos relacionados podem ser melhor explorados a partir das formalizações propostas. A seguir são apresentados algumas possibilidades de trabalhos futuros.

1. prova formal da segurança do sistema de tipos de cTVQL;
2. investigação de questões relativas a diferenciação entre objetos normais e objetos temporais e versionados;
3. Formalização do comportamento dinâmico do estado de banco de dados e de uma linguagem de modificação de dados;
4. estudo sobre a inclusão de relacionamentos aos modelos propostos. Sobre relacionamentos poderia-se analisar tanto a forma de representação quanto a introdução de semântica temporal;
5. estudar de forma abrangente outras linguagens de consulta temporais e talvez propor a inclusão de outros operadores temporais à TVQL além de `ever` e `present`;
6. relacionar as construções de TVQL à lógica modal temporal;
7. verificar a aplicabilidade da mônada temporal proposta para a semântica de cTVQL à outras linguagens de consulta que possuam semântica operacional definida;
8. integração com o modelo formal proposto para evolução de esquemas apresentado em (GALANTE, 2003).

REFERÊNCIAS

BARENDREGT, H. P. Functional Programming and Lambda Calculus. In: **Handbook of Theoretical Computer Science**: v.b. Amsterdam: Elsevier, 1990. p.321–363.

BIERMAN, G. M. Formal semantics and analysis of object queries. In: SIGMOD, 2003. **Proceedings...** [S.l.: s.n.], 2003.

BILIRIS, A. Modeling Design Object Relationships in PEGASUS. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 6., 1990, Los Angeles. **Proceedings...** Los Alamitos: IEEE Computer Society, 1990. p.228–236.

BUNEMAN, P.; LIBKIN, L.; SUCIU, D.; TANNEN, V.; WONG, L. Comprehension Syntax. **SIGMOD Record**, [S.l.], v.23, n.1, p.87–96, 1994.

CATTELL, R. G. G. et al. (Ed.). **The Object Data Standard**: ODMG 3.0. [S.l.]: Morgan Kaufmann, 2000.

COLAZZO, D. et al. Types for Correctness of Queries over Semistructured Data. In: WEBDB, 2002. **Proceedings...** [S.l.: s.n.], 2002. p.19–24.

DRAPER, D. et al. **XQuery 1.0 and XPath 2.0 Formal Semantics**. Disponível em: <`http://www.w3.org/TR/xquery-semantics`>. Acesso em: set. 2005.

EDELWEISS, N. **Sistemas de Informação de Escritórios**: um modelo para especificações temporais. 1994. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

FISHER, K. S. **Type System for Object-Oriented Languages**. 1996. Tese (Doutorado em Ciência da Computação) — Stanford University.

GALANTE, R. M. **Modelo Temporal de Versionamento com Suporte à Evolução de Esquemas**. 2003. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

GOLENDZINER, L. G. **Um modelo de versões para bancos de dados orientados a objetos**. 1995. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

GRANDI, F. et al. A Temporal Data Model and Management System for Normative Texts in XML Format. In: INTERNATIONAL WORKSHOP ON WEB INFORMATION AND DATA MANAGEMENT, WIDM, 5., 2003. **Proceedings...** [S.l.: s.n.], 2003.

KICK, M. **Coalgebraic Modelling of Timed Processes**. 2003. Tese (Doutorado em Ciência da Computação) — School of Informatics - University of Edinburgh.

MORO, M. M. **Modelo Temporal de Versões**. 2001. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

MORO, M. M.; EDELWEISS, N.; ZAUPA, A. P.; SANTOS, C. S. dos. TVQL - Temporal Versioned Query Language. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, 13., 2002, Aix-en-Provence, France. **Proceedings...** Berlin: Springer-Verlag, 2002. p.618–627. (Lecture Notes in Computer Science, v.2453).

NIELSON, H. R.; NIELSON, F. **Semantics with Applications**: a formal introduction. [S.l.]: Wiley Professional Computing, 1992.

PIERCE, B. C. **Types and Programming Languages**. [S.l.]: MIT Press, 2002.

RODRIGUEZ, L.; OGATA, H.; YANO, Y. TVOO: a temporal versioned object-oriented data model. **Inf. Sci.**, New York, NY, USA, v.114, n.1-4, p.281–300, 1999.

SIMÉON, J.; WADLER, P. The essence of XML. In: POPL, 2003. **Proceedings...** [S.l.: s.n.], 2003. p.1–13.

SNODGRASS, R. T. **Developing time-oriented database applications in SQL**. San Francisco, CA, USA: Morgan Kaufmann, 2000.

W3C XML WORK GROUP. **XML - Extensible Markup Language**. Disponível em <`http://www.w3.org/XML`>. Acesso em: set. 2005.

WRIGHT, A. K.; FELLEISEN, M. A Syntactic Approach to Type Soundness. **Information and Computation**, [S.l.], v.115, n.1, p.38–94, 1994.

ZAMULIN, A. V. An Object Algebra for the ODMG Standard. In: ADBIS, 2002. **Proceedings...** [S.l.: s.n.], 2002. p.291–304.

ZAMULIN, A. V. Formal Semantics of the ODMG 3.0 Object Query Language. In: ADBIS, 2003. **Proceedings...** [S.l.: s.n.], 2003. p.293–307.

ANEXO A DEFINIÇÕES E PROVAS REFERENTES A COQL

Esquema bem-formado

$wf(sch)$ se e somente se

1. cada classe está definida uma só vez;
2. duas classes diferentes não estão associadas à mesma extensão;
3. a classe $Object$ está no topo da hierarquia de classes;
4. todas as classes (com exceção de $Object$) possuem superclasse definida no esquema;
5. não há ciclos na hierarquia de classes;
6. nenhuma classe redefine um atributo definido em uma classe que esteja acima na hierarquia;
7. todas as classes referenciadas nos tipos dos atributos pertencem ao esquema.

Relação de validade

$sch \vdash_{sch} (ee, oe)$ se e somente se

1. o esquema sch é bem-formado.
2. $\bigcup_{c \in Dom(sch)} \{e \mid sch(c) = (p, e, ty)\} = Dom(ee)$
3. $\bigcup_{e \in Dom(ee)} ee(e) = Dom(oe)$
4. se $sch(c) = (p, e, ty)$ e $sch(p) = (p', e', ty')$, então $ee(e) \subseteq ee(e')$
5. se $sch(c) = (p, e, ty)$ então $sch; ee; oe; \Gamma \vdash ee(e) : Bag(c)$
6. se $oe(oid) = (c, objstate)$, $sch(c) = (p, e, ty)$ e $ty(a) = \sigma$ então $sch; ee; oe; \Gamma \vdash objstate(a) : \sigma$

Lema 2.1 *(Progresso) Se* $sch; ee; oe; \Gamma_{sch} \vdash q : \sigma$ *e* $sch \vdash_{sch} (ee, oe)$*, então ou* $q \in Values$ *ou existe* q' *tal que* $ee; oe \vdash q \rightarrow q'$*.*

Prova. Por indução sobre a estrutura da consulta q.

Casos $q \equiv b$, $q \equiv i$, $q \equiv s$, $q \equiv oid$, $q \equiv \texttt{nill}$, $q \equiv \{\vec{v}\}$

Triviais, já que q é valor.

Caso $q \equiv \{q_1, \ldots, q_n\}, \quad q \notin Values$

A consulta q é tipada pela regra T-Bag.

$$\frac{K \vdash q_1 : \sigma \quad \ldots \quad K \vdash q_n : \sigma}{K \vdash \{q_1, \ldots, q_n\} : Bag(\sigma)}$$

Já que q não é valor, existe no mínimo um componente não-valor. Seja q_k o primeiro não-valor da esquerda para direita em $\{q_1, \ldots, q_n\}$. Desta forma, tem-se a consulta q no formato $\{\vec{v}, \bullet, \vec{q}\}[q_k]$. Pelas premissas de T-Bag, q_k é bem-tipado sob K. Pela hipótese indutiva, ou q_k é valor ou então progride para q'_K sob ee, oe. Como q_k não é valor, só resta que $ee, oe \vdash q_k \rightarrow q'_k$. A consulta q então progride por S-Ctx.

$$\frac{ee; oe \vdash q_k \rightarrow q'_k}{ee; oe \vdash \{\vec{v}, \bullet, \vec{q}\}[q_k] \rightarrow \{\vec{v}, \bullet, \vec{q}\}[q'_k]}$$

Caso $q \equiv x$

A consulta q é tipada por T-Var.

$$\frac{\Gamma_{sch}(x) = \sigma}{sch; ee; oe; \Gamma_{sch} \vdash x : \sigma}$$

Pela restrição 2 da relação de validade e pela definição de Γ_{sch} (Seção 2.4), $Dom(ee) = Dom(\Gamma_{sch})$. Assim, $ee(x) = \{\overrightarrow{oid}\}$, satisfazendo os pré-requisitos da regra S-Var.

$$\frac{ee(x) = \{\overrightarrow{oid}\}}{ee; oe \vdash x \rightarrow \{\overrightarrow{oid}\}}$$

Caso $q \equiv q_1 \ bop \ q_2$

A regra T-Binop faz uso da função auxiliar δ_b para determinar tipos para q a partir da operação e dos tipos dos operandos. Pelas premissas de T-Binop, tanto q_1 quanto q_2 são bem-tipados sob K.

$$\frac{K \vdash q_1 : \sigma_1 \qquad K \vdash q_2 : \sigma_2 \qquad \delta_b(bop, \sigma_1, \sigma_2) = \sigma_3}{K \vdash q_1 \ bop \ q_2 : \sigma_3}$$

- se $q_1 \in Values$, há duas possibilidades:
  - se $q_2 \in Values$, como $K \vdash q_1 : \sigma_1$, $K \vdash q_2 : \sigma_2$ e $\delta_b(bop, \sigma_1, \sigma_2) = \sigma_3$, pelas premissa de T-Binop, e pelo lema 3 (Operações binárias), tem-se $\eta_b(bop, q_1, q_2) = v$. Desta forma, q progride por S-Binop.

    $$\frac{\eta_b(bop, q_1, q_2) = v}{ee; oe \vdash q_1 \ bop \ q_2 \rightarrow v}$$

  - se $q_2 \notin Values$, como q_2 é bem tipada, pela hipótese indutiva tem-se que $ee; oe \vdash q_2 \rightarrow q'_2$. Logo, q progride por S-Ctx.

    $$\frac{ee; oe \vdash q_2 \rightarrow q'_2}{ee; oe \vdash (v \ bop \ \bullet)[q_2] \rightarrow (v \ bop \ \bullet)[q'_2]}$$

- se $q_1 \notin Values$, como q_1 é bem tipada, pela hipótese indutiva tem-se que $ee; oe \vdash q_1 \rightarrow q'_1$. Logo, q progride por S-Ctx.

  $$\frac{ee; oe \vdash q_1 \rightarrow q'_1}{ee; oe \vdash (\bullet \ bop \ q)[q_1] \rightarrow (\bullet \ bop \ q)[q'_1]}$$

Caso $q \equiv uop\ q_1$

A regra T-Unop faz uso da função auxiliar δ_u para determinar tipos para q.

$$\frac{K \vdash q_1 : \sigma_1 \qquad \delta_u(uop, \sigma_1) = \sigma_2}{K \vdash uop\ q_1 : \sigma_2}$$

- se $q_1 \in Values$, como $K \vdash q_1 : \sigma_1$ e $\delta_u(uop, q_1) = q_2$, pelo lema 4 (Operações unárias) tem-se que $\eta_u(uop, q_1) = v$. Logo, q progride por S-Unop1.

$$\frac{\eta_u(uop, q_1) = v}{ee; oe \vdash uop\ q_1 \rightarrow v}$$

- se $q_1 \notin Values$, como a consulta q_1 é bem tipada, pela hipótese indutiva tem-se que $ee; oe \vdash q_1 \rightarrow q_1'$. Logo, q progride por S-Ctx.

$$\frac{ee; oe \vdash q_1 \rightarrow q_1'}{ee; oe \vdash (uop\ \bullet)[q_1] \rightarrow\ (uop\ \bullet)[q_1']}$$

Caso $q \equiv q_1.a$

A consulta q é tipada pela regra T-Dot.

$$\frac{K \vdash q : c \qquad sch(c) = (p, e, ty) \qquad ty(a) = \sigma}{K \vdash q.a : \sigma}$$

- $q_1 \in Values$. Pelas premissas de T-Dot, q_1 é do tipo c, sendo c uma das classes do esquema de dados. Pelo sistema de tipos, valores do tipo c podem ser `nill` ou *oid*.

  Caso $q_1 \equiv oid$, q_1 é tipado por T-Oid.

  $$\frac{oe(oid) = (c, objstate)}{K \vdash oid : c}$$

  Pelas premissas de T-Dot, de T-Oid e pela restrição 6 da relação de validade tem-se

  $$\frac{oe(oid) = (c, objstate) \qquad sch(c) = (p, e, ty) \qquad ty(a) = \sigma}{sch; ee; oe; \emptyset \vdash objstate(a) : \sigma}$$

  Logo existe v tal que $objstate(m) = v$, possibilitando que q progrida por S-Dot1

  $$\frac{oe(oid) = (c, objstate) \qquad objstate(a) = v}{ee; oe \vdash oid.a \rightarrow v}$$

  Caso $q_1 \equiv nill$, q progride por S-Dot2.

  $$ee; oe \vdash \texttt{nill}.a \rightarrow \texttt{nill}$$

- se $q_1 \notin Values$, como a consulta q_1 é bem tipada, pela hipótese indutiva tem-se que $ee; oe \vdash q_1 \rightarrow q_1'$. Logo, q progride por S-Ctx.

$$\frac{ee; oe \vdash q_1 \rightarrow q_1'}{ee; oe \vdash (\bullet.a)[q_1] \rightarrow (\bullet.a)[q_1']}$$

Caso $q \equiv q_1\texttt{->}a$

A consulta q é tipada pela regra T-Arrow.

$$\frac{K \vdash q : Bag(c) \qquad sch(c) = (p, e, ty) \qquad ty(a) = Bag(\sigma)}{K \vdash q\texttt{->}a : Bag(\sigma)}$$

- se $q_1 \in Values$, q_1 pode ser ou `nill` ou uma coleção de valores do tipo c, que podem ser tanto identificadores de objeto ou `nill`.

  Caso $q_1 \equiv \{oid, \vec{v}\}$, q progride através de S-Arrow1.

  $$ee; oe \vdash \{oid, \vec{v}\}\texttt{->}a \rightarrow oid.a \texttt{ union } \{\vec{v}\}\texttt{->}a$$

  Caso $q_1 \equiv \{\texttt{nill}, \vec{v}\}$, q progride através de S-Arrow2.

  $$ee; oe \vdash \{\texttt{nill}, \vec{v}\}\texttt{->}a \rightarrow \{\vec{v}\}\texttt{->}a$$

  Caso $q_1 \equiv \{\}$, q progride através de S-Arrow3.

  $$ee; oe \vdash \{\}\texttt{->}a \rightarrow \{\}$$

  Caso $q_1 \equiv \texttt{nill}$, q progride através de S-Arrow4.

  $$ee; oe \vdash \texttt{nill->}a \rightarrow \texttt{nill}$$

- se $q_1 \notin Values$, como a consulta q_1 é bem tipada, pela hipótese indutiva tem-se que $ee; oe \vdash q_1 \rightarrow q_1'$. Logo, q progride por S-Ctx.

  $$\frac{ee; oe \vdash q_1 \rightarrow q_1'}{ee; oe \vdash (\bullet\texttt{->}a)[q_1] \rightarrow (\bullet\texttt{->}a)[q_1']}$$

Caso $q \equiv \texttt{if } q_1 \texttt{ then } q_2 \texttt{ else } q_3$

A consulta q é tipada por T-If.

$$\frac{K \vdash q_1 : Bool \qquad K \vdash q_2 : \sigma \qquad K \vdash q_3 : \sigma}{K \vdash \texttt{if } q_1 \texttt{ then } q_2 \texttt{ else } q_3 : \sigma}$$

- se $q_1 \in Values$, e como $K \vdash q_1 : Bool$, há três possibilidades.

  Caso $q_1 \equiv \texttt{true}$, q progride por S-If1.

  $$ee; oe \vdash \texttt{if true then } q_2 \texttt{ else } q_3 \rightarrow q_2$$

  Caso $q_1 \equiv \texttt{false}$, q progride por S-If2.

  $$ee; oe \vdash \texttt{if false then } q_2 \texttt{ else } q_3 \rightarrow q_3$$

  Caso $q_1 \equiv \texttt{nill}$, q progride por S-If3.

  $$ee; oe \vdash \texttt{if nill then } q_2 \texttt{ else } q_3 \rightarrow \texttt{nill}$$

- se $q_1 \notin Values$, como a consulta q_1 é bem tipada, pela hipótese indutiva tem-se que $ee; oe \vdash q_1 \rightarrow q_1'$. Logo, q progride por S-Ctx.

  $$\frac{ee; oe \vdash q_1 \rightarrow q_1'}{ee; oe \vdash (\texttt{if } \bullet \texttt{ then } q_2 \texttt{ else } q_3)[q_1] \rightarrow (\texttt{if } \bullet \texttt{ then } q_2 \texttt{ else } q_3)[q_1']}$$

Caso $q \equiv \texttt{select } q_a \texttt{ from } q_1\ x_1,\ \ldots,\ q_n\ x_n \texttt{ where } q_b$

A consulta q é tipada por T-Select.

$$\frac{\begin{array}{rcl} sch; ee; oe; \Gamma & \vdash & q_1 : Bag(\sigma_1) \\ sch; ee; oe; \Gamma, x_1 : \sigma_1 & \vdash & q_2 : Bag(\sigma_2) \\ & \vdots & \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_{n-1} : \sigma_{n-1} & \vdash & q_n : Bag(\sigma_n) \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n & \vdash & q_a : \sigma_a \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n & \vdash & q_b : Bool \\ x_1 \text{ a } x_n \text{ diferentes} & & x_1, \ldots, x_n \notin Dom(\Gamma) \end{array}}{sch; ee; oe; \Gamma \vdash \texttt{select } q_a \texttt{ from } q_1\ x_1\ , \ldots, q_n\ x_n \texttt{ where } q_b : Bag(\sigma_a)}$$

- caso $n > 0$, pelas premissas de T-SELECT, $sch; ee; oe; \Gamma \vdash q_1 : Bag(\sigma_1)$
  - se $q_1 \in Values$, há três possibilidades.

    Caso $q_1 \equiv \{v_1, \ldots, v_m\}$, q progride por S-SELECT1.

$$ee; oe \vdash \texttt{select}\ q_a\ \texttt{from}\ \{v_1, \ldots, v_m\}\ x_1, \ldots, q_n\ x_n\ \texttt{where}\ q_b \rightarrow (\texttt{select}\ q_a\ \texttt{from}\ q_2\ x_2, \ldots, q_n\ x_n\ \texttt{where}\ q_b)[x_1 ::= v_1] \texttt{ union } \texttt{select}\ \{v_2, \ldots, v_m\}\ x_1, \ldots, q_n\ x_n\ \texttt{where}\ q_b$$

    Caso $q_1 \equiv \{\}$, q progride por S-SELECT2.

$$ee; oe \vdash \texttt{select}\ q_a\ \texttt{from}\ \{\}\ x_1, \ldots, q_n\ x_n\ \texttt{where}\ q_b \rightarrow \{\}$$

    Caso $q_1 \equiv \texttt{nill}$, q progride por S-SELECT3.

$$ee; oe \vdash \texttt{select}\ q_a\ \texttt{from nill}\ x_1, \ldots, q_n\ x_n\ \texttt{where}\ q_b \rightarrow \texttt{nill}$$

  - se $q_1 \notin Values$, como a consulta q_1 é bem tipada sob $sch; ee; oe; \Gamma$, pela hipótese indutiva tem-se que $ee; oe \vdash q_1 \rightarrow q_1'$. Logo, q progride por S-CTX.

$$\frac{ee; oe \vdash q_1 \rightarrow q_1'}{ee; oe \vdash (\texttt{select}\ q_a\ \texttt{from}\ \bullet\, x_1, \ldots, q_n\ x_n\ \texttt{where}\ q_b)[q_1] \rightarrow (\texttt{select}\ q_a\ \texttt{from}\ \bullet\, x_1, \ldots, q_n\ x_n\ \texttt{where}\ q_b)[q_1']}$$

- caso $n = 0$, pelas premissas de T-SELECT2, como não há variáveis sendo declaradas, $sch; ee; oe; \Gamma \vdash q_b : Bool$.
  - se $q_b \in Values$, há três possibilidades.

    Caso $q_b \equiv \texttt{true}$, q progride por S-SELECT4.

$$ee; oe \vdash \texttt{select}\ q_a\ \texttt{from where true} \rightarrow \{q_a\}$$

    Caso $q_b \equiv \texttt{false}$, q progride por S-SELECT5.

$$ee; oe \vdash \texttt{select}\ q_a\ \texttt{from where false} \rightarrow \{\}$$

    Caso $q_b \equiv \texttt{nill}$, q progride por S-SELECT6.

$$ee; oe \vdash \texttt{select}\ q_a\ \texttt{from where nill} \rightarrow \{\}$$

  - se $q_b \notin Values$, como a consulta q_b é bem tipada, pela hipótese indutiva tem-se que $ee; oe \vdash q_b \rightarrow q_b'$. Logo, q progride por S-CTX.

$$\frac{ee; oe \vdash q_b \rightarrow q_b'}{ee; oe \vdash (\texttt{select}\ q_a\ \texttt{from where}\ \bullet)[q_b] \rightarrow (\texttt{select}\ q_a\ \texttt{from where}\ \bullet)[q_b']}$$

Caso a consulta q seja tipada pela regra T-SUB, q precisa ser bem-tipada sob o mesmo contexto K nas premissas.

$$\frac{K \vdash q : \sigma_1 \qquad \sigma_1 <: \sigma_2}{K \vdash q : \sigma_2}$$

O progresso da avaliação é assegurado pela hipótese indutiva.

Lema 2.2 *(Preservação) Se* $sch; ee; oe; \Gamma_{sch} \vdash q : \sigma$, $ee; oe \vdash q \to q'$ *e* $sch \vdash_{sch} (ee, oe)$, *então* $sch; ee; oe; \Gamma_{sch} \vdash q' : \sigma$.

Prova. Por indução sobre a estrutura de q.

Casos $q \equiv b$, $q \equiv i$, $q \equiv s$, $q \equiv oid$, $q \equiv \texttt{nill}$, $q \equiv \{\vec{v}\}$

Triviais, já que não existe q' tal que $ee; oe \vdash q \to q'$.

Caso $q \equiv \{q_1, \ldots, q_n\}, \quad q \notin Values$

A consulta q é tipada por T-Bag.

$$\frac{K \vdash q_1 : \sigma \quad \ldots \quad K \vdash q_n : \sigma \qquad n > 0}{K \vdash \{q_1, \ldots, q_n\} : Bag(\sigma)}$$

- Caso q progrida por S-Ctx, $\mathcal{E} \equiv \{\vec{v}, \bullet, \vec{q}\}$ e $q' \equiv \{\vec{v}, \bullet, \vec{q}\}[q'_k]$.

$$\frac{ee; oe \vdash q_k \to q'_k}{ee; oe \vdash \{\vec{v}, \bullet, \vec{q}\}[q_k] \to \{\vec{v}, \bullet, \vec{q}\}[q'_k]}$$

Pelas premissas de T-Bag, todos os componentes q_1 a q_n são bem tipados. Dessa forma, como q_k é bem-tipado e progride sob os mesmos ambientes que q, pode-se dizer que $K \vdash q'_k : \sigma$ por hipótese indutiva sobre a preservação. Isto basta para garantir a preservação do tipo em q', que é tipado por T-Bag.

$$\frac{K \vdash q_1 : \sigma \ \ldots \ K \vdash q'_k : \sigma \ \ldots \ K \vdash q_n : \sigma \qquad n > 0}{K \vdash \{q_1, \ldots, q'_k, \ldots, q_n\} : Bag(\sigma)}$$

Caso $q \equiv x$

A consulta q é tipada por T-Var.

$$\frac{\Gamma_{sch}(x) = \sigma}{sch; ee; oe; \Gamma_{sch} \vdash x : \sigma}$$

Pela formação de Γ_{sch} tem-se $\sigma \equiv Bag(c)$.

- caso a consulta q progrida por S-Var, $q' \equiv \{oid_1, \ldots, oid_n\}$.

$$\frac{ee(x) = \{oid_1, \ldots, oid_n\}}{ee; oe \vdash x \to \{oid_1, \ldots, oid_n\}}$$

Pela restrição 5 da relação de validade, se $sch(c) = (p, e, ty)$ então $sch; ee; oe; \Gamma sch \vdash ee(e) : Bag(c)$, garantindo a preservação de tipos.

Caso $q \equiv q_1 \ bop \ q_2$

A consulta q é tipada por T-Binop.

$$\frac{K \vdash q_1 : \sigma_1 \qquad K \vdash q_2 : \sigma_2 \qquad \delta_b(bop, \sigma_1, \sigma_2) = \sigma_3}{K \vdash q_1 \ bop \ q_2 : \sigma_3}$$

- caso q progrida por S-Binop, q_1 e q_2 são valores e $q' \equiv \eta_b(bop, q_1, q_2)$.

$$\frac{\eta_b(bop, q_1, q_2) = v}{ee; oe \vdash q_1 \ bop \ q_2 \to v}$$

Pelas premissas de T-Binop, $K \vdash q_1 : \sigma_1$ e $K \vdash q_2 : \sigma_2$. Pelo lema 3 (Operações binárias), tem-se $\eta_b(bop, q_1, q_2) = v$ e $K \vdash v : \sigma_3$.

- caso q progrida por S-Ctx, há duas possibilidades.
  Se $\mathcal{E} \equiv (\bullet\ bop\ q_2)$, $q' \equiv (\bullet\ bop\ q_2)[q_1']$.

$$\frac{ee; oe \vdash q_1 \rightarrow q_1'}{ee; oe \vdash (\bullet\ bop\ q_2)[q_1] \rightarrow (\bullet\ bop\ q_2)[q_1']}$$

  Como $K \vdash q_1 : \sigma_1$ e $ee; oe \vdash q_1 \rightarrow q_1'$, por hipótese indutiva sobre a preservação, $K \vdash q_1' : \sigma_1$. Desta forma, $K \vdash q' : \sigma_3$ por T-Binop.

$$\frac{K \vdash q_1' : \sigma_1 \qquad K \vdash q_2 : \sigma_2 \qquad \delta_b(bop, \sigma_1, \sigma_2) = \sigma_3}{K \vdash q_1'\ bop\ q_2 : \sigma_3}$$

  Se $\mathcal{E} \equiv (v\ bop\ \bullet)$, q_1 é valor e $q' \equiv (v\ bop\ \bullet)[q_2']$.

$$\frac{ee; oe \vdash q_2 \rightarrow q_2'}{ee; oe \vdash (q_1\ bop\ \bullet)[q_2] \rightarrow (q_1\ bop\ \bullet)[q_2']}$$

  Como $K \vdash q_2 : \sigma_2$ e $ee; oe \vdash q_2 \rightarrow q_2'$, por hipótese indutiva sobre a preservação, $K \vdash q_2' : \sigma_2$. Desta forma, $K \vdash q' : \sigma_3$ por T-Binop.

$$\frac{K \vdash q_1 : \sigma_1 \qquad K \vdash q_2' : \sigma_2 \qquad \delta_b(bop, \sigma_1, \sigma_2) = \sigma_3}{K \vdash q_1\ bop\ q_2' : \sigma_3}$$

Caso $q \equiv uop\ q_1$

A consulta q é tipada por T-Unop.

$$\frac{K \vdash q_1 : \sigma_1 \qquad \delta_u(uop, \sigma_1) = \sigma_2}{K \vdash uop\ q_1 : \sigma_2}$$

- caso q progrida por S-Unop, q_1 é valor e $q' \equiv \eta_u(uop, q_1)$.

$$\frac{\eta_u(uop, q_1) = v}{ee; oe \vdash uop\ q_1 \rightarrow v}$$

  Pelas premissas de T-Unop, $K \vdash q_1 : \sigma_1$ e $\delta_u(uop, \sigma_1) = \sigma_2$. Pelo lema 3 (Operações binárias), tem-se então que $\eta_u(uop, q_1) = v$ e $K \vdash v : \sigma_2$.
- caso q progrida por S-Ctx, $\mathcal{E} \equiv (uop\ \bullet)$ e $q' \equiv (uop\ \bullet)[q_1']$.

$$\frac{ee; oe \vdash q_1 \rightarrow q_1'}{ee; oe \vdash (uop\ \bullet)[q_1] \rightarrow (uop\ \bullet)[q_1']}$$

  Como $K \vdash q_1 : \sigma_1$ e $ee; oe \vdash q_1 \rightarrow q_1'$, por hipótese indutiva sobre a preservação, $K \vdash q_1' : \sigma_1$. Desta forma, $K \vdash q' : \sigma_2$ por T-Unop.

$$\frac{K \vdash q_1' : \sigma_1 \qquad \delta_u(uop, \sigma_1) = \sigma_2}{K \vdash uop\ q_1' : \sigma_2}$$

Caso $q \equiv q_1.a$

A consulta q é tipada por T-Dot.

$$\frac{K \vdash q_1' : c \qquad sch(c) = (p, e, ty) \qquad ty(a) = \sigma}{K \vdash q_1'.a : \sigma}$$

- caso q progrida por S-Dot1, $q' \equiv v_1$.

$$\frac{oe(oid) = (c, objstate) \qquad objstate(m) = v_1}{ee; oe \vdash oid.a \rightarrow v_1}$$

  Pelas premissas de T-Dot e pela restrição 6 da relação de validade tem-se

$$\frac{oe(oid) = (c, objstate) \qquad sch(c) = (p, e, ty) \qquad ty(a) = \sigma}{sch; ee; oe; \Gamma_{sch} \vdash objstate(a) : \sigma}$$

  Desta forma, $sch; ee; oe; \Gamma_{sch} \vdash v_1 : \sigma$.

- caso q progrida por S-Dot2, $q' \equiv \texttt{nill}$.
  O valor `nill` é tipado por T-Nill.

$$K \vdash \texttt{nill} : \sigma$$

- caso q progrida por S-Ctx, $\mathcal{E} \equiv (\bullet.a)$ e $q' \equiv (\bullet.a)[q_1']$

$$\frac{ee; oe \vdash q_1 \rightarrow q_1'}{ee; oe \vdash (\bullet.a)[q_1] \rightarrow (\bullet.a)[q_1']}$$

  Como $K \vdash q_1 : c$ e $ee; oe \vdash q_1 \rightarrow q_1'$, por hipótese indutiva sobre a preservação, $K \vdash q_1' : c$. Desta forma, $K \vdash q' : \sigma$ por T-Dot.

$$\frac{K \vdash q_1' : c \qquad sch(c) = (p, e, ty) \qquad ty(a) = \sigma}{K \vdash q_1'.a : \sigma}$$

Caso $q \equiv q_1\texttt{->}a$

A consulta q é tipada por T-Arrow.

$$\frac{K \vdash q_1 : Bag(c) \qquad sch(c) = (p, e, ty) \qquad ty(a) = Bag(\sigma)}{K \vdash q\texttt{->}a : Bag(\sigma)}$$

- caso q progrida por S-Arrow1, $q' \equiv oid.a$ `union` $\{\vec{v}\}\texttt{->}a$

$$ee; oe \vdash \{oid, \vec{v}\}\texttt{->}a \rightarrow oid.a \texttt{ union } \{\vec{v}\}\texttt{->}a$$

  Neste caso, q' resulta em uma operação binária (`union`) sobre duas consultas.
  - primeira consulta.

    $q_1 \equiv \{oid, \vec{v}\}$. Pelas premissas de T-Arrow, $K \vdash q_1 : Bag(c)$. O tipo de q_1 é dado por T-Bag.

$$\frac{K \vdash oid : c \quad K \vdash v_1 : c \quad \ldots \quad K \vdash q_n : \sigma}{K \vdash \{oid, v_1, \ldots, v_n\} : Bag(c)}$$

    O tipo de $oid.a$ é determinado por T-Dot. Pelas premissas de T-Arrow e de T-Bag, tem-se

$$\frac{K \vdash oid : c \qquad sch(c) = (p, e, ty) \qquad ty(a) = Bag(\sigma)}{K \vdash oid.a : Bag(\sigma)}$$

  - segunda consulta.

    $q_1 \equiv \{oid, \vec{v}\}$. Pelas premissas de T-Arrow, $K \vdash q_1 : Bag(c)$. O tipo de q_1 é dado por T-Bag.

$$\frac{K \vdash oid : c \quad K \vdash v_1 : c \quad \ldots \quad K \vdash q_n : \sigma}{K \vdash \{oid, v_1, v_n\} : Bag(c)}$$

    Pelas premissas de T-Bag, $K \vdash \{v_1, \ldots, v_n\} : Bag(c)$. Por T-Arrow tem-se

$$\frac{K \vdash \{v_1, \ldots, v_n\} : Bag(c) \qquad sch(c) = (p, e, ty) \qquad ty(a) = Bag(\sigma)}{K \vdash \{v_1, \ldots, v_n\}\texttt{->}a : Bag(\sigma)}$$

  Assim, como ambos os operandos são do tipo $Bag(\sigma)$, por T-Binop tem-se:

$$\frac{K \vdash oid.a : Bag(\sigma) \qquad K \vdash \{\vec{v}\}\texttt{->}a : Bag(\sigma) \qquad \delta_b(\texttt{union}, Bag(\sigma), Bag(\sigma)) = Bag(\sigma)}{K \vdash oid.a \texttt{ union } \{\vec{v}\} : Bag(\sigma)}$$

- caso q progrida por S-Arrow2, $q' \equiv \{\vec{v}\}\texttt{->}a$.

$$ee; oe \vdash \{\texttt{nill}, \vec{v}\}\texttt{->}a \rightarrow \{\vec{v}\}\texttt{->}a$$

  $q_1 \equiv \{\texttt{nill}, \vec{v}\}$. Pelas premissas de T-Arrow, $K \vdash q_1 : Bag(c)$. O tipo de q_1 é dado por T-Bag.

$$\frac{K \vdash \texttt{nill} : c \quad K \vdash v_1 : c \quad \ldots \quad K \vdash q_n : \sigma}{K \vdash \{oid, v_1, \ldots, v_n\} : Bag(c)}$$

  Pelas premissas de T-Bag, $K \vdash \{v_1, \ldots, v_n\} : Bag(c)$. Por T-Arrow tem-se

$$\frac{K \vdash \{v_1, \ldots, v_n\} : Bag(c) \qquad sch(c) = (p, e, ty) \qquad ty(a) = Bag(\sigma)}{K \vdash \{v_1, \ldots, v_n\}\texttt{->}a : Bag(\sigma)}$$

- caso q progrida por S-Arrow3, $q' \equiv \{\}$

$$ee; oe \vdash \{\}\texttt{->}m \rightarrow \{\}$$

Neste caso, o tipo de q' é dado por T-Bag.

$$K \vdash \{\} : Bag(\sigma)$$

- caso q progrida por S-Arrow4.

$$ee; oe \vdash \texttt{nill->}a \rightarrow \texttt{nill}$$

Neste caso, o tipo de q' é dado por T-Nill.

$$K \vdash \texttt{nill} : Bag(\sigma)$$

- caso q progrida por S-Ctx, $\mathcal{E} \equiv (\bullet\texttt{->}a)$ e $q' \equiv (\bullet\texttt{->}a)[q_1']$.

$$\frac{ee; oe \vdash q_1 \rightarrow q_1'}{ee; oe \vdash (\bullet\texttt{->}a)[q_1] \rightarrow (\bullet\texttt{->}a)[q_1']}$$

Como $K \vdash q_1 : Bag(c)$ e $ee; oe \vdash q_1 \rightarrow q_1'$, por hipótese indutiva sobre a preservação, $K \vdash q_1' : Bag(c)$. Desta forma, $K \vdash q' : Bag(\sigma)$ por T-Arrow.

$$\frac{K \vdash q_1' : Bag(c) \qquad sch(c) = (p, e, ty) \qquad ty(a) = Bag(\sigma)}{K \vdash q_1'\texttt{->}a : Bag(\sigma)}$$

Caso $q \equiv \texttt{if } q_1 \texttt{ then } q_2 \texttt{ else } q_3$

A consulta q é tipada por T-If.

$$\frac{K \vdash q_1 : Bool \qquad K \vdash q_2 : \sigma \qquad K \vdash q_3 : \sigma}{K \vdash \texttt{if } q_1 \texttt{ then } q_2 \texttt{ else } q_3 : \sigma}$$

- caso q progrida por S-If1, $q_1 \equiv \texttt{true}$ e $q' \equiv q_2$

$$ee; oe \vdash \texttt{if true then } q_2 \texttt{ else } q_3 \rightarrow q_2$$

Pelas premissas de T-If, $K \vdash q_2 : \sigma$.

- caso q progrida por S-If2, $q_1 \equiv \texttt{false}$ e $q' \equiv q_3$.

$$ee; oe \vdash \texttt{if false then } q_2 \texttt{ else } q_3 \rightarrow q_3$$

Pelas premissas de T-If, $K \vdash q_3 : \sigma$.

- caso q progrida por S-If3, $q_1 \equiv \texttt{nill}$ e $q' \equiv \texttt{nill}$.

$$ee; oe \vdash \texttt{if nill then } q_1 \texttt{ else } q_2 \rightarrow \texttt{nill}$$

Por T-Nill, $K \vdash \texttt{nill} : \sigma$

- caso q progrida por S-Ctx, $\mathcal{E} \equiv \texttt{if } \bullet \texttt{ then } q_2 \texttt{ else } q_3$ e $q' \equiv (\texttt{if } \bullet \texttt{ then } q_2 \texttt{ else } q_3)[q_1]$.

$$\frac{ee; oe \vdash q_1 \rightarrow q_1'}{ee; oe \vdash (\texttt{if } \bullet \texttt{ then } q_2 \texttt{ else } q_3)[q_1] \rightarrow (\texttt{if } \bullet \texttt{ then } q_2 \texttt{ else } q_3)[q_1']}$$

Pelas premissas de T-If, $K \vdash q_1 : Bool$. Como q_1 é bem-tipado e $ee; oe \vdash q_1 \rightarrow q_1'$, por hipótese indutiva sobre a preservação, $K \vdash q_1' : Bool$. Desta forma, $K \vdash q' : \sigma$ por T-If.

$$\frac{K \vdash q_1' : Bool \qquad K \vdash q_2 : \sigma \qquad K \vdash q_3 : \sigma}{K \vdash \texttt{if } q_1' \texttt{ then } q_2 \texttt{ else } q_3 : \sigma}$$

Caso $q \equiv \texttt{select}\ q_a\ \texttt{from}\ q_1\ x_1\ , \ldots, q_n\ x_n\ \texttt{where}\ q_b$

A consulta q é tipada por T-SELECT.

$$\frac{\begin{array}{rcl} sch; ee; oe; \Gamma & \vdash & q_1 : Bag(\sigma_1) \\ sch; ee; oe; \Gamma, x_1 : \sigma_1 & \vdash & q_2 : Bag(\sigma_2) \\ & \vdots & \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_{n-1} : \sigma_{n-1} & \vdash & q_n : Bag(\sigma_n) \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n & \vdash & q_a : \sigma_a \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n & \vdash & q_b : Bool \\ x_1 \text{ a } x_n \text{ diferentes} & & x_1, \ldots, x_n \notin Dom(\Gamma) \end{array}}{sch; ee; oe; \Gamma \vdash \texttt{select}\ q_a\ \texttt{from}\ q_1\ x_1\ , \ldots, q_n\ x_n\ \texttt{where}\ q_b : Bag(\sigma_a)}$$

- caso q progrida por S-SELECT1,

$$\begin{array}{c} ee; oe \vdash \texttt{select}\ q_a\ \texttt{from}\ \{v_1, \ldots, v_m\}\ x_1, \ldots, q_n\ x_n\ \texttt{where}\ q_b \rightarrow \\ (\texttt{select}\ q_a\ \texttt{from}\ q_2\ x_2, \ldots, q_n\ x_n\ \texttt{where}\ q_b)[x_1 ::= v_1] \\ \texttt{union} \\ \texttt{select}\ q_a\ \texttt{from}\ \{v_2, \ldots, v_m\}\ x_1, \ldots, q_n\ x_n\ \texttt{where}\ q_b \end{array}$$

Neste caso, q' resulta em uma operação binária (`union`) sobre duas consultas. Pela regra T-BINOP, basta provar que ambas possuem tipo $Bag(\sigma_a)$.

– primeira consulta.

Pelas premissas de T-SELECT, $q_1 \equiv \{v_1, \ldots, v_m\}$ e $K \vdash q_1 : Bag(\sigma_1)$. Pelas premissas de T-BAG, $K \vdash v_1 : \sigma_1$.
Por T-SELECT, utilizando-se as premissas da tipagem de q, tem-se:

$$\frac{\begin{array}{rcl} sch; ee; oe; \Gamma, x_1 : \sigma_1 & \vdash & q_2 : Bag(\sigma_2) \\ & \vdots & \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_{n-1} : \sigma_{n-1} & \vdash & q_n : Bag(\sigma_n) \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n & \vdash & q_a : \sigma_a \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n & \vdash & q_b : Bool \\ x_2 \text{ a } x_n \text{ diferentes} & & x_2, \ldots, x_n \notin Dom(\Gamma, x_1 : \sigma_1) \end{array}}{sch; ee; oe; \Gamma, x_1 : \sigma_1 \vdash \texttt{select}\ q_a\ \texttt{from}\ q_2\ x_2\ , \ldots, q_n\ x_n\ \texttt{where}\ q_b : Bag(\sigma_a)}$$

Como $\vdash v_1 : \sigma_1$ e
$sch; ee; oe; \Gamma, x_1 : \sigma_1 \vdash \texttt{select}\ q_a\ \texttt{from}\ q_2\ x_2 \ldots, q_n\ x_n\ \texttt{where}\ q_b : Bag(\sigma_a)$, pelo lema 5 (Substituição) tem-se que $sch; ee; oe; \Gamma \vdash (\texttt{select}\ q_a\ \texttt{from}\ q_2\ x_2\ , \ldots, q_n\ x_n\ \texttt{where}\ q_b)[x_1 ::= v_1] : Bag(\sigma_a)$

– segunda consulta.

Pelas premissas de T-SELECT, $sch; ee; oe; \Gamma \vdash \{v_1, \ldots, v_n\} : Bag(\sigma_1)$ por T-BAG.

$$\frac{K \vdash v_1 : \sigma_1 \quad \ldots \quad K \vdash v_n : \sigma_1}{K \vdash \{v_1, \ldots, v_n\} : Bag(\sigma_1)}$$

Pelas premissas da regra, tem-se

$$\frac{K \vdash v_2 : \sigma_1 \quad \ldots \quad K \vdash v_n : \sigma_1}{K \vdash \{v_2, \ldots, v_n\} : Bag(\sigma_1)}$$

Por T-SELECT,

$$\frac{\begin{array}{rcl} sch; ee; oe; \Gamma & \vdash & \{v_2, \ldots, v_n\} : Bag(\sigma_1) \\ sch; ee; oe; \Gamma, x_1 : \sigma_1 & \vdash & q_2 : Bag(\sigma_2) \\ & \vdots & \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_{n-1} : \sigma_{n-1} & \vdash & q_n : Bag(\sigma_n) \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n & \vdash & q_a : \sigma_a \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n & \vdash & q_b : Bool \\ x_1 \text{ a } x_n \text{ diferentes} & & x_1, \ldots, x_n \notin Dom(\Gamma) \end{array}}{sch; ee; oe; \Gamma \vdash \texttt{select}\ q_a\ \texttt{from}\ \{v2, \ldots, v_n\}\ x_1\ , \ldots, q_n\ x_n\ \texttt{where}\ q_b : Bag(\sigma_a)}$$

- caso q progrida por S-SELECT2, $q' \equiv \{\}$.

$$ee; oe \vdash \texttt{select}\ q_a\ \texttt{from}\ \{\}\ x_1, \ldots, q_m\ x_n\ \texttt{where}\ q_b \rightarrow \{\}$$

  Neste caso, o tipo de q' é dado por T-BAG.

$$K \vdash \{\} : Bag(\sigma_a)$$

- caso q progrida por S-SELECT3, $q' \equiv \texttt{nill}$.

$$ee; oe \vdash \texttt{select}\ q_a\ \texttt{from nill}\ x_1, \ldots, q_m\ x_n\ \texttt{where}\ q_b \rightarrow \{\}$$

  Neste caso, o tipo de q' é dado por T-NILL.

$$K \vdash \texttt{nill} : \sigma$$

- caso q progrida por S-SELECT4, $q' \equiv \{q_a\}$.

$$ee; oe \vdash \texttt{select}\ q_a\ \texttt{from where}\ true \rightarrow \{q_a\}$$

  Neste caso, como não há variáveis sendo declaradas, $sch; ee; oe; \Gamma \vdash q_a : \sigma_a$. Desta forma, por T-BAG:

$$\frac{K \vdash q_a : \sigma_a}{K \vdash \{q_a\} : Bag(\sigma_a)}$$

- caso q progrida por S-SELECT5, $q' \equiv \{\}$.

$$ee; oe \vdash \texttt{select}\ q_a\ \texttt{from where}\ false \rightarrow \{\}$$

  Neste caso, o tipo de q' é dado por T-BAG.

$$K \vdash \{\} : Bag(\sigma_a)$$

- caso q progrida por S-SELECT6, $q' \equiv \{\}$.

$$ee; oe \vdash \texttt{select}\ q_a\ \texttt{from where nill} \rightarrow \{\}$$

  Neste caso, o tipo de q' é dado por T-BAG.

$$K \vdash \{\} : Bag(\sigma_a)$$

Lema 2.3 *(Operações Binárias) Se* $K \vdash v_1 : \sigma_1$, $K \vdash v_2 : \sigma_2$ *e* $\delta_b(bop, \sigma_1, \sigma_2) = \sigma_3$*, então* $\eta_b(bop, v_1, v_2) = v_3$ *e* $K \vdash v_3 : \sigma_3$*.*

Prova. Pela definição de *bop*.

Casos $bop \equiv$ `+`, $bop \equiv$ `-`, $bop \equiv$ `*`

Pela definição de δ_b, $\delta_b(bop, Int, Int) = Int$. Os valores v_1 e v_2, pelo sistema de tipos, podem ser inteiros ou `nill`. Se um dos argumentos for `nill`, $\eta_b(bop, v_1, v_2) =$ `nill`, sendo `nill` do tipo Int por T-Nill. Se ambos forem inteiros, $\eta_b(bop, v_1, v_2) = i$, já que estas operações são *totais*, sendo i do tipo `Int` por T-Int.

Portanto, $\eta_b(bop, v_1, v_2) = v_3$ e $K \vdash v_3 : Int$.

Casos $bop \equiv$ `<`, $bop \equiv$ `<=`, $bop \equiv$ `>`, $bop \equiv$ `>=`

Pela definição de δ_b, $\delta_b(bop, Int, Int) = Bool$. Os valores v_1 e v_2, pelo sistema de tipos, podem ser inteiros ou `nill`. Se um dos argumentos for `nill`, $\eta_b(bop, v_1, v_2) =$ `nill`, sendo `nill` do tipo $Bool$ por T-Nill. Se ambos forem inteiros, $\eta_b(bop, v_1, v_2) = b$, já que estas operações de comparação são *totais*, sendo b do tipo `Bool` por T-Bool.

Portanto, $\eta_b(bop, v_1, v_2) = v_3$ e $K \vdash v_3 : Bool$.

Casos $bop \equiv$ `and`, $bop \equiv$ `or`

Pela definição de δ_b, $\delta_b(bop, Bool, Bool) = Bool$. Os valores v_1 e v_2, pelo sistema de tipos, podem ser booleanos ou `nill`. Se um dos argumentos for `nill`, $\eta_b(bop, v_1, v_2) =$ `nill`, sendo `nill` do tipo $Bool$ por T-Nill. Se ambos forem booleanos, $\eta_b(bop, v_1, v_2) = b$, já que estas operações booleanas são *totais*, sendo b do tipo $Bool$ por T-Bool.

Portanto, $\eta_b(bop, v_1, v_2) = v_3$ e $K \vdash v_3 : Bool$.

Casos $bop \equiv$ `union`, $bop \equiv$ `intersection`, $bop \equiv$ `difference`

Pela definição de δ_b, $\delta_b(bop, Bag(\sigma), Bag(\sigma)) = Bag(\sigma)$. Os valores v_1 e v_2, pelo sistema de tipos, podem ser coleções ou `nill`. Se um dos argumentos for `nill`, $\eta_b(bop, v_1, v_2) =$ `nill`, sendo `nill` do tipo $Bag(\sigma)$ por T-Nill.

Se ambos operandos forem coleções de elementos dos tipo σ, $\eta_b(bop, \{\vec{v_a}\}, \{\vec{v_b}\}) = \{\vec{v_c}\}$. Como as coleções $\{\vec{v_a}\}$ e $\{\vec{v_b}\}$ são bem-tipadas, seus elementos possuem tipo σ. Como as operações sobre multiconjuntos são totais, e o resultado é uma coleção na qual ocorrem somente elementos que aparecem nos operandos, pode-se garantir que o resultado é do tipo $Bag(\sigma)$ por T-Bag.

Portanto, $\eta_b(bop, v_1, v_2) = v_3$ e $K \vdash v_3 : Bag(\sigma)$.

Casos $bop \equiv$ `=`, $bop \equiv$ `<>`

Pela definição de δ_b, $\delta_b(bop, \sigma, \sigma) = Bool$. Os valores v_1 e v_2 podem ser valores quaisquer do mesmo tipo. Se um dos argumentos for `nill`, $\eta_b(bop, v_1, v_2) =$ `nill`, sendo `nill` do tipo $Bool$ por T-Nill. Se ambos argumentos estiverem definidos, pode-se avaliar se são iguais ou não pela sua estrutura, levando a $\eta(bop, v_1, v_2) = b$, sendo b do tipo $Bool$ por T-Bool.

Portanto, $\eta_b(bop, v_1, v_2) = v_3$ e $K \vdash v_3 : Bool$.

Lema 2.4 *(Operações Unárias) Se* $K \vdash v_1 : \sigma_1$ *e* $\delta_u(uop, \sigma_1) = \sigma_2$*, então* $\eta_u(uop, v_1) = v_2$ *e* $K \vdash v_2 : \sigma_2$*.*

Prova. Pela definição de *uop*.

Caso $uop \equiv$ `-`

Pela definição de δ_u, $\delta_b(-, Int) = Int$. Pelo sistema de tipos, v_1 pode ser um inteiro ou `nill`. Se v_1 for `nill`, $\eta_u(uop, v_1) =$ `nill`, sendo `nill` do tipo Int por T-Nill. Se v_1 for inteiro, $\eta_u(-, v_1)$, retornando o inverso de v_1, que é um inteiro i do tipo Int por T-Int.

Portanto, $\eta_u(-, v_1) = v_2$ e $K \vdash v_2 : Int$.

Caso $uop \equiv$ `not`

Pela definição de δ_u, $\delta_u(\texttt{not}, Bool) = Bool$. Pelo sistema de tipos, v_1 pode ser um booleano ou `nill`. Se v_1 for `nill`, $\eta_u(uop, v_1) =$ `nill`, sendo `nill` do tipo $Bool$ por T-Nill. Se v_1 for booleano, $\eta_u(\texttt{not}, v_1)$ retorna a negação de v_1, que é um booleano b do tipo $Bool$ por T-Bool.

Portanto, $\eta_u(\texttt{not}, v_1) = v_2$ e $K \vdash v_2 : Int$.

Caso $uop \equiv$ `set`

Pela definição de δ_u, $\delta_u(\texttt{not}, Bag(\sigma)) = Bag(\sigma)$. Pelo sistema de tipos, v_1 pode ser uma coleção de elementos do tipo σ ou então o valor `nill`. Se v_1 for `nill`, $\eta_u(\texttt{set}, v_1) =$ `nill`, sendo `nill` do tipo $Bag(\sigma)$ por T-Nill. Se v_1 for uma coleção de elementos do tipo σ, $\eta_u(\texttt{set}, v_1)$ retorna uma coleção com os mesmos elementos de v_1, porém sem repetição de elementos. Por T-Bag, tal coleção possui tipo $Bag(\sigma)$.

Portanto, $\eta_u(\texttt{set}, v_1) = v_2$ e $K \vdash v_2 : Bag(\sigma)$.

Caso $uop \equiv$ `size`

Pela definição de δ_u, $\delta_u(\texttt{size}, Bag(\sigma)) = Int$. Pelo sistema de tipos, v_1 pode ser uma coleção de elementos do tipo σ ou então o valor `nill`. Se v_1 for `nill`, $\eta_u(\texttt{size}, v_1) =$ `nill`, sendo `nill` do tipo $Bag(\sigma)$ por T-Nill. Se v_1 for uma coleção de elementos do tipo σ, $\eta_u(\texttt{size}, v_1)$ retorna o inteiro i representando o número de elementos da coleção, sendo i do tipo Int por T-Int.

Portanto, $\eta_u(\texttt{size}, v_1) = v_2$ e $K \vdash v_2 : Int$.

Lema 2.5 *(Substituição) Se* $sch; ee; oe; \Gamma \vdash v : \sigma_v$, $sch; ee; oe; \Gamma, x : \sigma_v \vdash q : \sigma$ *e* $x \notin \Gamma$, *então* $sch; ee; oe; \Gamma \vdash q[x ::= v] : \sigma$.

Prova. Por indução sobre a estrutura de q.

Casos $q \equiv b$, $q \equiv i$, $q \equiv s$, $q \equiv oid$, $q \equiv nill$, $q \equiv x'$, $\quad x' \neq x$

Triviais, já que $sch; ee; oe; \Gamma, x : \sigma_v \vdash q : \sigma$ e $q[x ::= v] = q$.

Caso $q \equiv x$

A consulta q é tipada por T-Var.

$$\frac{(\Gamma, x : \sigma_v)(x) = \sigma_v}{sch; ee; oe; \Gamma \vdash x : \sigma_v}$$

Pela definição da operação de substituição, tem-se

$$x[x ::= v] = v$$

Pelo enunciado da substituição, $sch; ee; oe; \Gamma \vdash v : \sigma_v$. Como $x = q$, tem-se que $\sigma = \sigma_v$.

Caso $q \equiv \{q_1, \ \ldots, \ q_n\}$

A consulta q é tipada por T-Bag.

$$\frac{sch; ee; oe; \Gamma, x : \sigma_v \vdash q_1 : \sigma_m \quad \ldots \quad sch; ee; oe; \Gamma, x : \sigma_v \vdash q_n : \sigma_m}{sch; ee; oe; \Gamma, x : \sigma_v \vdash \{q_1, \ldots, q_n\} : Bag(\sigma_m)}$$

Pela definição da operação de substituição,

$$\{q_1, \ldots, q_n\}[x ::= v] = \{q_1[x ::= v], \ldots, q_n[x ::= v]\}$$

Pelas premissas de T-Bag, $sch; ee; oe; \Gamma, x : \sigma_v \vdash q_k : \sigma_m$ para todo os elementos de q_k de q. Pela hipótese indutiva, como $sch; ee; oe; \Gamma \vdash v : \sigma_v$ e $sch; ee; oe; \Gamma, x : \sigma_v \vdash q_k : \sigma_m$, então $sch; ee; oe; \Gamma \vdash q_k[x ::= v] : \sigma_m$. Desta forma, $sch; ee; oe; \Gamma \vdash q[x ::= v] : Bag(\sigma_m)$ por T-Bag.

$$\frac{sch; ee; oe; \Gamma \vdash q_1[x ::= v] : \sigma_m \quad \ldots \quad sch; ee; oe; \Gamma \vdash q_n[x ::= v] : \sigma_m}{sch; ee; oe; \Gamma \vdash \{q_1[x ::= v], \ldots, q_n[x ::= v]\} : Bag(\sigma_m)}$$

Caso $q \equiv q_1 \ bop \ q_2$

A consulta q é tipada por T-Binop.

$$\frac{sch; ee; oe; \Gamma, x : \sigma_v \vdash q_1 : \sigma_1 \qquad sch; ee; oe; \Gamma, x : \sigma_v \vdash q_2 : \sigma_2 \qquad \delta_b(bop, \sigma_1, \sigma_2) = \sigma}{sch; ee; oe; \Gamma, x : \sigma_v \vdash q_1 \ bop \ q_2 : \sigma}$$

Pela definição da operação de substituição,

$$(q_1 \ bop \ q_2)[x ::= v] = q_1[x ::= v] \ bop \ q_2[x ::= v]$$

Pelas premissas de T-Binop, tanto q_1 quanto q_2 são bem-tipados. Pela hipótese indutiva, $sch; ee; oe; \Gamma \vdash q_1[x ::= v] : \sigma_1$ e $sch; ee; oe; \Gamma \vdash q_2[x ::= v] : \sigma_2$. Desta forma, $sch; ee; oe; \Gamma \vdash q[x ::= v] : \sigma$ por T-Binop.

$$\frac{sch; ee; oe; \Gamma \vdash q_1[x ::= v] : \sigma_1 \qquad sch; ee; oe; \Gamma \vdash q_2[x ::= v] : \sigma_2 \qquad \delta_b(bop, \sigma_1, \sigma_2) = \sigma}{sch; ee; oe; \Gamma \vdash q_1[x ::= v] \ bop \ q_2[x ::= v] : \sigma}$$

Caso $q \equiv uop\ q_1$

A consulta q é tipada por T-Unop.

$$\frac{sch; ee; oe; \Gamma, x : \sigma_v \vdash q_1 : \sigma_1 \qquad \delta_u(uop, \sigma_1) = \sigma}{sch; ee; oe; \Gamma, x : \sigma_v \vdash uop\ q_1 : \sigma}$$

Pela definição da operação de substituição,

$$(uop\ q_1)[x ::= v] = uop\ (q_1[x ::= v])$$

Pelas premissas de T-Unop, $sch; ee; oe; \Gamma, x : \sigma_v \vdash q_1 : \sigma_1$. Pela hipótese indutiva, $sch; ee; oe; \Gamma \vdash q_1[x ::= v] : \sigma_1$. Desta forma, $sch; ee; oe; \Gamma \vdash q[x ::= v] : \sigma$ por T-Unop.

$$\frac{sch; ee; oe; \Gamma \vdash q_1[x ::= v] : \sigma_1 \qquad \delta_u(uop, \sigma_1) = \sigma}{sch; ee; oe; \Gamma \vdash uop\ q_1[x ::= v] : \sigma}$$

Caso $q \equiv q_1.a$

A consulta q é tipada por T-Dot.

$$\frac{sch; ee; oe; \Gamma, x : \sigma_v \vdash q_1 : c \qquad sch(c) = (p, e, ty) \qquad ty(a) = \sigma}{sch; ee; oe; \Gamma, x : \sigma_v \vdash q_1.a : \sigma}$$

Pela definição da operação de substituição,

$$(q_1.a)[x ::= v] = q_1[x ::= v].a$$

Pelas premissas de T-Dot, $sch; ee; oe; \Gamma, x : \sigma_v \vdash q_1 : c$. Pela hipótese indutiva, $sch; ee; oe; \Gamma \vdash q_1[x ::= v] : c$. Desta forma, $sch; ee; oe; \Gamma \vdash q[x ::= v] : \sigma$ por T-Dot.

$$\frac{sch; ee; oe; \Gamma \vdash q_1[x ::= v] : c \qquad sch(c) = (p, e, ty) \qquad ty(a) = \sigma}{sch; ee; oe; \Gamma \vdash q_1[x ::= v].a : \sigma}$$

Caso $q \equiv q_1\texttt{->}a$

A consulta q é tipada por T-Arrow.

$$\frac{sch; ee; oe; \Gamma, x : \sigma_v \vdash q : Bag(c) \qquad sch(c) = (p, e, ty) \qquad ty(a) = Bag(\sigma)}{sch; ee; oe; \Gamma, x : \sigma_v \vdash q\texttt{->}a : Bag(\sigma)}$$

Pela definição da operação de substituição,

$$(q_1\texttt{->}a)[x ::= v] = q_1[x ::= v]\texttt{->}a$$

Pelas premissas de T-Arrow, $sch; ee; oe; \Gamma, x : \sigma_v \vdash q_1 : Bag(c)$. Pela hipótese indutiva, $sch; ee; oe; \Gamma \vdash q_1[x ::= v] : Bag(c)$. Desta forma, $sch; ee; oe; \Gamma \vdash q[x ::= v] : Bag(\sigma_m)$ por T-Arrow.

$$\frac{sch; ee; oe; \Gamma \vdash q[x ::= v] : Bag(c) \qquad sch(c) = (p, e, ty) \qquad ty(a) = Bag(\sigma)}{sch; ee; oe; \Gamma \vdash q[x ::= v]\texttt{->}a : Bag(\sigma)}$$

Caso $q \equiv \texttt{if}\ q_1\ \texttt{then}\ q_2\ \texttt{else}\ q_3$

A consulta q é tipada por T-If.

$$\frac{sch; ee; oe; \Gamma, x : \sigma_v \vdash q_1 : Bool \qquad sch; ee; oe; \Gamma, x : \sigma_v \vdash q_2 : \sigma \qquad sch; ee; oe; \Gamma, x : \sigma_v \vdash q_3 : \sigma}{sch; ee; oe; \Gamma, x : \sigma_v \vdash if\ q_1\ then\ q_2\ else\ q_3 : \sigma}$$

Pela definição da operação de substituição,

$$(\texttt{if}\ q\ \texttt{then}\ q_1\ \texttt{else}\ q_2)[x ::= v] = \texttt{if}\ q[x ::= v]\ \texttt{then}\ q_1[x ::= v]\ \texttt{else}\ q_2[x ::= v]$$

Pelas premissas de T-If, $sch; ee; oe; \Gamma, x : \sigma_v \vdash q_1 : Bool$, $sch; ee; oe; \Gamma, x : \sigma_v \vdash q_2 : \sigma$ e $sch; ee; oe; \Gamma, x : \sigma_v \vdash q_3 : \sigma$. Pela hipótese indutiva, $sch; ee; oe; \Gamma \vdash q_1[x ::= v] : Bool$, $sch; ee; oe; \Gamma \vdash q_2[x ::= v] : \sigma$ e $sch; ee; oe; \Gamma \vdash q_3[x ::= v] : \sigma$. Desta forma, $sch; ee; oe; \Gamma \vdash q[x ::= v] : \sigma$ por T-If.

$$\frac{sch; ee; oe; \Gamma \vdash q_1[x ::= v] : Bool \qquad sch; ee; oe; \Gamma \vdash q_2[x ::= v] : \sigma \qquad sch; ee; oe; \Gamma \vdash q_3[x ::= v] : \sigma}{sch; ee; oe; \Gamma \vdash \texttt{if}\ q_1[x ::= v]\ \texttt{then}\ q_2[x ::= v]\ \texttt{else}\ q_3[x ::= v] : \sigma}$$

Caso $q \equiv \texttt{select}\ q_a\ \texttt{from}\ q_1\ x_1,\ \ldots,\ q_n\ x_n\ \texttt{where}\ q_b$

A consulta q é tipada por T-SELECT.

$$\frac{\begin{array}{rcl} sch; ee; oe; \Gamma, x : \sigma_v, & \vdash & q_1 : Bag(\sigma_1) \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, x : \sigma_v & \vdash & q_2 : Bag(\sigma_2) \\ & \vdots & \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_{n-1} : \sigma_{n-1}, x : \sigma_v & \vdash & q_n : Bag(\sigma_n) \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_n, : \sigma_n x : \sigma_v & \vdash & q_a : \sigma_a \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n, x : \sigma_v & \vdash & q_b : Bool \\ x_1 \text{ a } x_n \text{ diferentes} \qquad x_1, \ldots, x_n \notin Dom(\Gamma, x : \sigma_v) & & \end{array}}{sch; ee; oe; \Gamma, x : \sigma_v, \vdash \texttt{select}\ q_a\ \texttt{from}\ q_1\ x_1\ , \ldots, q_n\ x_n\ \texttt{where}\ q_b : Bag(\sigma_a)}$$

Pela definição da operação de substituição,

$$\begin{array}{lcl} (\texttt{select}\ q_a & & \texttt{select}\ q_a[x ::= v] \\ \texttt{from}\ q_1\ x_1, \ldots, q_n\ x_n & = & \texttt{from}\ q_1[x ::= v]\ x_1, \ldots, q_n[x ::= v]\ x_n \\ \texttt{where}\ q_b)[x ::= v] & & \texttt{where}\ q_b[x ::= v] \end{array}$$

Pelas premissas de T-SELECT, $sch; ee; oe; \Gamma, x : \sigma_v, \vdash q_1 : Bag(\sigma_1)$. Pela hipótese indutiva, então $sch; ee; oe; \Gamma \vdash q_1[x ::= v] : Bag(\sigma_1)$. O mesmo ocorre com todas as premissas de T-SELECT, levando-se em consideração cada ambiente de tipos. Desta forma:

$$\frac{\begin{array}{rcl} sch; ee; oe; \Gamma & \vdash & q_1[x ::= v] : Bag(\sigma_1) \\ sch; ee; oe; \Gamma, x_1 : \sigma_1 & \vdash & q_2[x ::= v] : Bag(\sigma_2) \\ & \vdots & \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_{n-1} : \sigma_{n-1} & \vdash & q_n[x ::= v] : Bag(\sigma_n) \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n & \vdash & q_a[x ::= v] : \sigma_a \\ sch; ee; oe; \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n & \vdash & q_b[x ::= v] : Bool \\ x_1 \text{ a } x_n \text{ diferentes} \qquad x_1, \ldots, x_n \notin Dom(\Gamma, x : \sigma_v) & & \end{array}}{sch; ee; oe; \Gamma, x : \sigma_v, \vdash \texttt{select}\ q_a[x ::= v]\ \texttt{from}\ q_1[x ::= v]\ x_1\ , \ldots, q_n[x ::= v]\ x_n\ \texttt{where}\ q_b[x ::= v] : Bag(\sigma_a)}$$

ANEXO B DEFINIÇÕES E PROVAS REFERENTES A CVOQL

Esquema bem-formado

$wf(vsch)$ se e somente se

1. cada classe está definida uma só vez;
2. duas classes não podem estar associadas à mesma extensão;
3. a classe *Object* está no topo da hierarquia de classes;
4. todas as classes (com exceção de *Object*) possuem superclasse definida no esquema;
5. não há ciclos na hierarquia de classes;
6. nenhuma classe redefine um atributo definido em uma classe que esteja acima na hierarquia;
7. todas as classes referenciadas nos tipos dos atributos pertencem ao esquema.

Relação de validade

$vsch \vdash_{vsch} (vee, voe, ve)$ se e somente se

1. o esquema *vsch* é bem-formado.
2. $\bigcup_{c \in Dom(vsch)} \{e \mid vsch(c) = (p, e, ty)\} = Dom(vee)$
3. $\bigcup_{e \in Dom(vee)} vee(e) = Dom(voe)$
4. se $sch(c) = (p, e, ty)$ e $sch(p) = (p', e', ty')$ então $vee(e) \subseteq vee(e')$
5. $\bigcup_{void \in Dom(voe)} (ver$ onde $voe(void) = (c, (ver, curr, deriv))\ ) = Dom(ve)$
6. se $void_1 \neq void_2$, $voe(void_1) = (c_1, (ver_1, curr_1, deriv_1))$ e $voe(void_2) = (c_2, (ver_2, curr_2, deriv_2))$ então $ver_1 \cap ver_2 = \emptyset$
7. se $voe(void) = (c, (ver, curr, deriv))$ então $curr \in ver$, $deriv \subseteq ver \times ver$ e $deriv$ é uma árvore
8. se $vsch(c) = (p, e, ty)$ então $vsch; vee; voe; ve; \Gamma \vdash vee(e) : Bag(c)$
9. se $voe(void) = (c, (ver, curr, deriv))$ então $vsch; vee; voe; ve; \Gamma \vdash ver : Bag(c)$ e $vsch; vee; voe; ve; \Gamma \vdash curr : c$
10. se $ve(vid) = (c, vstate)$, $vsch(c) = (p, e, ty)$ e $ty(a) = \sigma$ então $vsch; vee; voe; ve; \Gamma_{vsch} \vdash vstate(a) : \sigma$

Lema 3.1 *(Progresso) Se* $vsch; vee; voe; ve; \Gamma_{vsch} \vdash q : \sigma$ *e* $vsch \vdash_{vsch} (vee, voe, ve)$*, então ou* $q \in Values$ *ou existe* q' *tal que* $vee; voe; ve \vdash q \rightarrow q'$*.*

Prova. Por indução sobre a estrutura da consulta q.

Casos $q \equiv b$, $q \equiv i$, $q \equiv s$, $q \equiv vid$, $q \equiv \texttt{nill}$, $q \equiv \{\vec{v}\}$, $q \equiv v_1, v_2$

Triviais, já que q é valor.

Caso $q \equiv void$

A consulta q é tipada por VT-Void.

$$\frac{voe(void) = (c, (versions, current, deriv))}{K \vdash void : c}$$

Progride por VS-Void.

$$\frac{voe(void) = (c, (versions, current, deriv))}{vee; voe; ve \vdash void \rightarrow current}$$

Caso $q \equiv \{q_1, \ldots, q_n\}, \quad q \notin Values$

Prova similar à do mesmo caso no lema 2.1 (Progresso em cOQL).

Caso $q \equiv q_1, q_2 \quad q \notin Values$

A consulta q é tipada pela regra VT-Pair.

$$\frac{K \vdash q_1 : \sigma \qquad K \vdash q_2 : \sigma_2}{K \vdash q_1, q_2 : \sigma_1 * \sigma_2}$$

- caso $q_1 \notin Values$, como q_1 é bem-tipada, pela hipótese indutiva tem-se que $vee; voe; ve \vdash q_1 \rightarrow q_1'$. Logo, q progride por VS-Ctx.

$$\frac{vee; voe; ve \vdash q_1 \rightarrow q_1'}{vee; voe; ve \vdash (\bullet, q_2)[q_1] \rightarrow (\bullet, q_2)[q_1']}$$

- caso $q_1 \in Values$ e $q_2 \notin Values$, como q_2 é bem-tipada, pela hipótese indutiva tem-se que $vee; voe; ve \vdash q_2 \rightarrow q_2'$. Logo, q progride por VS-Ctx.

$$\frac{vee; voe; ve \vdash q_2 \rightarrow q_2'}{vee; voe; ve \vdash (q_1, \bullet)[q_2] \rightarrow (q_1, \bullet)[q_2']}$$

Caso $q \equiv x$

A consulta q é tipada por VT-Var.

$$\frac{\Gamma_{vsch}(x) = \sigma}{vsch; vee; voe; ve; \Gamma_{vsch} \vdash x : \sigma}$$

Pela restrição 2 da relação de validade e pela definição de Γ_{vsch} (Seção 3.4), $Dom(vee) = Dom(\Gamma_{vsch})$. Assim, $vee(x) = \{\overrightarrow{void}\}$, satisfazendo os pré-requisitos da regra VS-Var.

$$\frac{vee(x) = \{\overrightarrow{void}\}}{vee; voe; ve \vdash x \rightarrow \{\overrightarrow{void}\}}$$

Caso $q \equiv q_1\ bop\ q_2$

Prova similar à do mesmo caso no lema 2.1 (Progresso em cOQL). Faz referência ao lema 3.3 (Operações Binárias em cVOQL).

Caso $q \equiv uop\ q_1$

Prova similar à do mesmo caso no lema 2.1 (Progresso em cOQL). Faz referência ao lema 3.4 (Operações Unárias em cVOQL).

Caso $q \equiv q_1.a$

A consulta q é tipada pela regra VT-Dot.

$$\frac{K \vdash q_1 : c \qquad vsch(c) = (p, e, ty) \qquad ty(a) = \sigma}{K \vdash q_1.a : \sigma}$$

- caso $q_1 \in Values$, pelas premissas de VT-Dot, q_1 é do tipo c. Pelo sistema de tipos, valores do tipo c podem ser `nill` ou vid.

  Caso $q_1 \equiv vid$, q_1 é tipado por VT-Vid.

  $$\frac{ve(vid) = (c, vstate)}{K \vdash vid : c}$$

  Pelas premissas de VT-Dot e de VT-Vid e pela restrição 10 da relação de validade tem-se

  $$\frac{ve(vid) = (c, vstate) \qquad vsch(c) = (p, e, ty) \qquad ty(a) = \sigma}{vsch; vee; voe; ve; \Gamma_{vsch} \vdash vstate(a) : \sigma}$$

  Logo existe q_2 tal que $vstate(a) = q_2$, possibilitando que q progrida por VS-Dot1.

  $$\frac{ve(vid) = (c, vstate) \qquad vstate(a) = q_2}{vee; voe; ve \vdash vid.a \rightarrow q_2}$$

  Caso $q_1 \equiv nill$, q progride por S-Dot2.

  $$vee; voe; ve \vdash \texttt{nill}.a \rightarrow \texttt{nill}$$

- se $q_1 \notin Values$, como a consulta q_1 é bem tipada, pela hipótese indutiva tem-se que $vee; voe; ve \vdash q_1 \rightarrow q_1'$. Logo, q progride por VS-Ctx.

  $$\frac{vee; voe; ve \vdash q_1 \rightarrow q_1'}{vee; voe; ve \vdash (\bullet.a)[q_1] \rightarrow (\bullet.a)[q_1']}$$

Caso $q \equiv q_1\texttt{->}m$

A consulta q é tipada pela regra VT-Arrow.

$$\frac{K \vdash q_1 : Bag(c) \qquad vsch(c) = (p, e, ty) \qquad ty(a) = Bag(\sigma)}{K \vdash q_1\texttt{->}a : Bag(\sigma)}$$

- se $q_1 \in Values$, q_1 pode ser ou `nill` ou uma coleção de valores do tipo c, que podem ser tanto `nill` ou vid.

  Caso $q_1 \equiv \{vid, \vec{v}\}$, q progride através de VS-Arrow1.

  $$vee; voe; ve \vdash \{vid, \vec{v}\}\texttt{->}a \rightarrow vid.a \texttt{ union } \{\vec{v}\}\texttt{->}a$$

  Caso $q_1 \equiv \{\texttt{nill}, \vec{v}\}$, q progride através de VS-Arrow2.

  $$vee; voe; ve \vdash \{\texttt{nill}, \vec{v}\}\texttt{->}a \rightarrow \{\vec{v}\}\texttt{->}a$$

  Caso $q_1 \equiv \{\}$, q progride através de S-Arrow3.

  $$vee; voe; ve \vdash \{\}\texttt{->}a \rightarrow \{\}$$

  Caso $q_1 \equiv \texttt{nill}$, q progride através de S-Arrow4.

  $$vee; voe; ve \vdash \texttt{nill->}a \rightarrow \texttt{nill}$$

- se $q_1 \notin Values$, como a consulta q_1 é bem tipada, pela hipótese indutiva tem-se que $vee; voe; vee \vdash q_1 \rightarrow q_1'$. Logo, q progride por VS-Ctx.

  $$\frac{vee; voe; ve \vdash q_1 \rightarrow q_1'}{vee; voe; ve \vdash (\bullet\texttt{->}a)[q_1] \rightarrow (\bullet\texttt{->}a)[q_1']}$$

Caso $q \equiv q_1$`->versions`

A consulta q é tipada por VT-Versions.

$$\frac{K \vdash q : Bag(c)}{K \vdash q\texttt{->versions} : Bag(c)}$$

- se $q_1 \in Values$, q_1 pode ser ou `nill` ou uma coleção de valores do tipo c, que podem ser tanto `nill` ou vid.

  Caso $q_1 \equiv \{vid, \vec{v}\}$, pelas restrições 5 e 6 da relação de validade, todo $vid \in Dom(ve)$ pertence ao conjunto de versões de um e somente um $void$. Desta forma, q progride por VS-Versions1.

  $$\frac{voe(void) = (c, (versions, current, deriv)) \qquad vid \in versions}{vee; voe; ve \vdash \{vid, \vec{v}\}\texttt{->versions} \rightarrow\ versions\ \texttt{union}\ \{\vec{v}\}\texttt{->versions}}$$

  Caso $q_1 \equiv \{\texttt{nill}, \vec{v}\}$, q progride através de VS-Versions2.

  $$vee; voe; ve \vdash \{\texttt{nill}, \vec{v}\}\texttt{->versions} \rightarrow \{\vec{v}\}\texttt{->versions}$$

  Caso $q_1 \equiv \{\}$, q progride através de VS-Versions3.

  $$vee; voe; ve \vdash \{\}\texttt{->versions} \rightarrow \{\}$$

  Caso $q_1 \equiv$ `nill`, q progride através de VS-Versions4.

  $$vee; voe; ve \vdash \texttt{nill->versions} \rightarrow \texttt{nill}$$

- se $q_1 \notin Values$, como a consulta q_1 é bem tipada, pela hipótese indutiva tem-se que $vee; voe; ve \vdash q_1 \rightarrow q_1'$. Logo, q progride por VS-Ctx.

  $$\frac{vee; voe; ve \vdash q_1 \rightarrow q_1'}{vee; voe; ve \vdash (\bullet\texttt{->versions})[q_1] \rightarrow (\bullet\texttt{->versions})[q_1']}$$

Caso $q \equiv$ `if` q_1 `then` q_2 `else` q_3

A estrutura da prova é similar à estrutura da prova do mesmo caso no lema 2.1 (Progresso em cOQL).

Caso $q \equiv$ `select` q_a `from` $q_1\ x_1,\ \ldots,\ q_n\ x_n$ `where` q_b

A estrutura da prova é similar à estrutura da prova do mesmo caso no lema 2.1 (Progresso em cOQL).

Caso a consulta q seja tipada pela regra VT-Sub, q precisa ser bem-tipada sob o mesmo contexto K nas premissas.

$$\frac{K \vdash q : \sigma_1 \qquad \sigma_1 <: \sigma_2}{K \vdash q : \sigma_2}$$

O progresso da avaliação é assegurado pela hipótese indutiva.

Lema 3.2 *(Preservação) Se* $vsch; vee; voe; ve; \Gamma_{vsch} \vdash q : \sigma$, $vee; voe; ve \vdash q \rightarrow q'$ *e* $vsch \vdash_{vsch} (vee, voe, ve)$, *então* $vsch; vee; voe; ve; \Gamma_{vsch} \vdash q' : \sigma$.

Prova. Por indução sobre a estrutura de q.

Casos $q \equiv b$, $q \equiv i$, $q \equiv s$, $q \equiv oid$, $q \equiv \texttt{nill}$, $q \equiv \{\vec{v}\}$, $q \equiv q_1, q_2$

Triviais, já que não existe q' tal que $ee; oe \vdash q \rightarrow q'$.

Caso $q \equiv void$

A consulta q é tipada por VT-Void.

$$\frac{voe(void) = (c, (versions, current, deriv))}{K \vdash void : c}$$

- caso a consulta q progrida por VS-Void, $q' \equiv current$

$$\frac{voe(void) = (c, (versions, current, deriv))}{vee; voe; ve \vdash void \rightarrow current}$$

Pela restrição 9 da relação de validade ,

$$\frac{\begin{array}{c} voe(void) = (c, (versions, current, deriv)) \\ vsch; vee; voe; ve; \Gamma_{vsch} \vdash versions : Bag(c) \\ vsch; vee; voe; ve; \Gamma_{vsch} \vdash current : c \end{array}}{vsch; vee; voe; ve; \Gamma_{vsch} \vdash deriv : Bag(c \times c)}$$

Portanto, $vsch; vee; voe; ve; \Gamma_{vsch} \vdash current : c$

Caso $q \equiv \{q_1, \ldots, q_n\}, \quad q \notin Values$

A estrutura da prova é similar à estrutura da prova do mesmo caso no lema 2.2 (Preservação em cOQL).

Caso $q \equiv q_1, q_2, \quad q \notin Values$

A consulta q é tipada por VT-Pair.

$$\frac{K \vdash q_1 : \sigma_1 \qquad K \vdash q_2 : \sigma_2}{K \vdash q_1, q_2 : \sigma_1 * \sigma_2}$$

- caso a consulta q progrida por S-Ctx sendo $\mathcal{E} \equiv (\bullet, q_2)$, $q' \equiv (q_1', q_2)$.

$$\frac{vee; voe; ve \vdash q_1 \rightarrow q_1'}{vee; voe; ve \vdash (\bullet, q_2)[q_1] \rightarrow (\bullet, q_2)[q_1']}$$

Como $K \vdash q_1 : \sigma_1$ e $vee; voe; ve \vdash q_1 \rightarrow q_1'$, por hipótese indutiva sobre a preservação, $K \vdash q_1' : \sigma_1$. Desta forma, $K \vdash q' : \sigma_1 * \sigma_2$ por VT-Pair.

$$\frac{K \vdash q_1' : \sigma_1 \qquad K \vdash q_2 : \sigma_2}{K \vdash q_1', q_2 : \sigma_1 * \sigma_2}$$

- caso a consulta q progrida por S-Ctx sendo $\mathcal{E} \equiv (q_1, \bullet)$, $q_1 \in Values$ e $q' \equiv (q_1, q_2')$.

$$\frac{vee; voe; ve \vdash q_2 \rightarrow q_2'}{vee; voe; ve \vdash (q_1, \bullet)[q_2] \rightarrow (q_1, \bullet)[q_2']}$$

Como $K \vdash q_2 : \sigma_2$ e $vee; voe; ve \vdash q_2 \rightarrow q_2'$, por hipótese indutiva sobre a preservação, $K \vdash q_2' : \sigma_2$. Desta forma, $K \vdash q' : \sigma_1 * \sigma_2$ por VT-Pair.

$$\frac{K \vdash q_1 : \sigma_1 \qquad K \vdash q_2' : \sigma_2}{K \vdash q_1, q_2' : \sigma_1 * \sigma_2}$$

Caso $q \equiv x$

A consulta q é tipada por VT-Var.

$$\frac{\Gamma(x) = \sigma}{vsch; vee; voe; \Gamma \vdash x : \sigma}$$

Pela formação de Γ_{sch} tem-se $\Gamma(x) = Bag(c)$.

- caso a consulta q progrida por VS-Var, $q' \equiv \{oid_1, \ldots, oid_n\}$

$$\frac{vee(x) = \{\overrightarrow{void}\}}{vee; voe; ve \vdash x \rightarrow \{\overrightarrow{void}\}}$$

Pela restrição 2 da relação de validade, toda extensão definida por vee é referenciada por no mínimo uma classe no esquema $vsch$. Pela boa-formação do esquema, duas classes diferentes não referenciam uma mesma extensão. Logo, existe uma classe c tal que $vsch(c) = (p, e, ty)$. Pela restrição 8 da relação de validade, se $vsch(c) = (p, e, ty)$ então $vsch; vee; voe; ve; \Gamma_v sch \vdash vee(e) : Bag(c)$. Portanto, $vsch; vee; voe; ve; \Gamma_v sch \vdash \{\overrightarrow{void} : Bag(c)$.

Caso $q \equiv q_1\ bop\ q_2$

A estrutura da prova é similar à estrutura da prova do mesmo caso no lema 2.2 (Preservação em cOQL). Faz referência ao lema 3.3 (Operações Binárias em cVOQL).

Caso $q \equiv uop\ q_1$

A estrutura da prova é similar à estrutura da prova do mesmo caso no lema 2.2 (Preservação em cOQL). Faz referência ao lema 3.4 (Operações Unárias em cVOQL).

Caso $q \equiv q_1.a$

A consulta q é tipada por VT-Dot.

$$\frac{K \vdash q_1 : c \qquad vsch(c) = (p, e, ty) \qquad ty(a) = \sigma}{K \vdash q_1.a : \sigma}$$

- caso a consulta q progrida por VS-Dot1,$q_1 \equiv vid$ e $q' \equiv q_2$

$$\frac{ve(vid) = (c, vstate) \qquad vstate(a) = q_2}{vee; voe; ve \vdash vid.a \rightarrow q_2}$$

Pela restrição 10 da relação de validade, e pelas premissas de VT-Dot e VS-Dot1, tem-se

$$\frac{ve(vid) = (c, vstate) \qquad vsch(c) = (p, e, ty) \qquad ty(a) = \sigma}{vsch; vee; voe; ve; \Gamma_{vsch} \vdash vstate(a) : \sigma}$$

- caso a consulta q progrida por VS-Dot2,$q_1 \equiv \texttt{nill}$ e $q' \equiv \texttt{nill}$

$$vee; voe; ve \vdash \texttt{nill}.a \rightarrow \texttt{nill}$$

Por VT-Nill,

$$K \vdash \texttt{nill} : \sigma$$

- caso a consulta q progrida por S-Ctx sendo $\mathcal{E} \equiv (q_1.a)$, $q' \equiv (q_1'.a)$.

$$\frac{vee; voe; ve \vdash q_1 \rightarrow q_1'}{vee; voe; ve \vdash (\bullet.a)[q_1] \rightarrow (\bullet.a)[q_1']}$$

Como $K \vdash q_1 : c$ e $vee; voe; ve \vdash q_1 \rightarrow q_1'$, por hipótese indutiva sobre a preservação, $K \vdash q_1' : c$. Desta forma, $K \vdash q' : \sigma$ por VT-Dot.

$$\frac{K \vdash q_1' : c \qquad vsch(c) = (p, e, ty) \qquad ty(a) = \sigma}{K \vdash q_1'.a : \sigma}$$

Caso $q \equiv q_1\texttt{->}a$

A consulta q é tipada por VT-Arrow.

$$\frac{K \vdash q_1 : Bag(c) \qquad vsch(c) = (p, e, ty) \qquad ty(a) = Bag(\sigma)}{K \vdash q_1\texttt{->}a : Bag(\sigma)}$$

- caso q progrida por VS-Arrow1, $q' \equiv vid.a$ `union` $\{\vec{v}\}\texttt{->}a$.

  $$vee; voe; ve \vdash \{vid, \vec{v}\}\texttt{->}a \rightarrow vid.a \texttt{ union } \{\vec{v}\}\texttt{->}a$$

  A consulta q é avaliada para uma operação sobre duas consultas. Por VT-Binop, basta provar que ambas possuem tipo $Bag(\sigma)$.

  – primeira consulta, $vid\texttt{->}a$

  A consulta $q_1 \equiv \{\texttt{nill}, \vec{v}\}$ é tipada por VT-Bag.

  $$\frac{K \vdash vid : c \quad K \vdash v_1 : c \ldots \quad K \vdash v_n : c}{K \vdash \{vid, v_1, \ldots, v_n\} : Bag(c)}$$

  Pelas premissas de VT-Bag e de VT-Arrow, tem-se por VT-Dot.

  $$\frac{K \vdash vid : c \qquad vsch(c) = (p, e, ty) \qquad ty(a) = Bag(\sigma)}{K \vdash q.a : Bag(\sigma)}$$

  – segunda consulta, $\{\vec{v}\}\texttt{->}a$

  A consulta $q_1 \equiv \{\texttt{nill}, \vec{v}\}$ é tipada por VT-Bag.

  $$\frac{K \vdash \texttt{nill} : \sigma \quad K \vdash v_1 : \sigma \ldots \quad K \vdash v_n : \sigma}{K \vdash \{\texttt{nill}, v_1, \ldots, v_n\} : Bag(\sigma)}$$

  Pelas premissas de VT-Bag, $K \vdash \{v_1, \ldots, v_n\} : Bag(c)$. Desta forma, $K \vdash q' : Bag(\sigma)$ por VT-Arrow.

$$\frac{K \vdash \{v_1, \ldots, v_n\} : Bag(c) \qquad vsch(c) = (p, e, ty) \qquad ty(a) = Bag(\sigma)}{K \vdash \{v_1, \ldots, v_n\}\texttt{->}a : Bag(\sigma)}$$

- caso q progrida por VS-Arrow2.

  $$vee; voe; ve \vdash \{\texttt{nill}, \vec{v}\}\texttt{->}a \rightarrow \{\vec{v}\}\texttt{->}a$$

  A consulta $q_1 \equiv \{\texttt{nill}, \vec{v}\}$ é tipada por VT-Bag.

  $$\frac{K \vdash \texttt{nill} : \sigma \quad K \vdash v_1 : \sigma \ldots \quad K \vdash v_n : \sigma}{K \vdash \{\texttt{nill}, v_1, \ldots, v_n\} : Bag(\sigma)}$$

  Pelas premissas de VT-Bag, $K \vdash \{v_1, \ldots, v_n\} : Bag(c)$. Desta forma, $K \vdash q' : Bag(\sigma)$ por VT-Arrow.

  $$\frac{K \vdash \{v_1, \ldots, v_n\} : Bag(c) \qquad vsch(c) = (p, e, ty) \qquad ty(a) = Bag(\sigma)}{K \vdash \{v_1, \ldots, v_n\}\texttt{->}a : Bag(\sigma)}$$

- caso q progrida por VS-Arrow3, $q' \equiv \{\}$.

  $$vee; voe; ve \vdash \{\}\texttt{->}a \rightarrow \{\}$$

  Por VT-Bag,

  $$K \vdash \{\} : Bag(\sigma)$$

- caso q progrida por VS-Arrow4, $q' \equiv \texttt{nill}$.

$$vee; voe; ve \vdash \texttt{nill->}a \rightarrow \texttt{nill}$$

Por VT-Nill,

$$K \vdash \texttt{nill} : Bag(\sigma)$$

- caso q progrida por VS-Ctx sendo $\mathcal{E} \equiv \bullet\texttt{->}a$, $q' \equiv q_1'\texttt{->}a$.

$$\frac{vee; voe; ve \vdash q_1 \rightarrow q_1'}{vee; voe; ve \vdash (\bullet\texttt{->}a)[q_1] \rightarrow (\bullet\texttt{->}a)[q_1']}$$

Como $K \vdash q_1 : Bag(c)$ e $vee; voe; ve \vdash q_1 \rightarrow q_1'$ progride, por hipótese indutiva sobre a preservação, $K \vdash q_1' : Bag(c)$. Desta forma,

$$\frac{K \vdash q_1' : Bag(c) \qquad vsch(c) = (p, e, ty) \qquad ty(a) = Bag(\sigma)}{K \vdash q_1'\texttt{->}a : Bag(\sigma)}$$

Caso $q \equiv \texttt{if}\ q_1\ \texttt{then}\ q_2\ \texttt{else}\ q_3$

A estrutura da prova é similar à estrutura da prova do mesmo caso no lema 2.2 (Preservação em cOQL).

Caso $q \equiv \texttt{select}\ q_a\ \texttt{from}\ q_1\ x_1\ , \ldots , q_n\ x_n\ \texttt{where}\ q_b$

A estrutura da prova é similar à estrutura da prova do mesmo caso no lema 2.2 (Preservação em cOQL).

A diferença entre os lemas auxiliares de operações binárias e unárias em cOQL e cVOQL reside no acréscimo de casos e na inclusão de $vsch \vdash_{vsch} (vee, voe, ve)$ nas premissas. Isto se deve ao acréscimo de operações binárias e unárias sobre versões cujo resultado depende da configuração do estado do banco de dados.

Lema 3.3 *(Operações Binárias) Se* $vsch \vdash_{vsch} (vee, voe, ve)$, $K \vdash v_1 : \sigma_1$, $K \vdash v_2 : \sigma_2$ *e* $\delta_b(bop, \sigma_1, \sigma_2) = \sigma_3$, *então* $\eta_b(bop, v_1, v_2) = v_3$ *e* $K \vdash v_3 : \sigma_3$.

Prova. Pela definição de *bop*.

Casos $bop \equiv$ `+`, $bop \equiv$ `-`, $bop \equiv$ `*`

Prova similar ao mesmo caso no lema 2.3 (Operações Binárias em cOQL).

Casos $bop \equiv$ `<`, $bop \equiv$ `<=`, $bop \equiv$ `>`, $bop \equiv$ `>=`

Prova similar ao mesmo caso no lema 2.3 (Operações Binárias em cOQL).

Casos $bop \equiv$ `and`, $bop \equiv$ `or`

Prova similar ao mesmo caso no lema 2.3 (Operações Binárias em cOQL).

Casos $bop \equiv$ `union`, $bop \equiv$ `intersection`, $bop \equiv$ `difference`

Prova similar ao mesmo caso no lema 2.3 (Operações Binárias em cOQL).

Casos $bop \equiv$ `=`, $bop \equiv$ `<>`

Prova similar ao mesmo caso no lema 2.3 (Operações Binárias em cOQL).

Casos $bop \equiv$ `is_succ`, $bop \equiv$ `is_pred`

Pela definição de δ_b, $\delta_b(bop, c, c) = Bool$. Portanto, os valores v_1 e v_2 são do tipo c, podendo ser ou `nill` ou *vid*. Se um dos argumentos for `nill`, $\eta_b(bop, v_1, v_2) =$ `nill`, sendo `nill` do tipo *Bool* por VT-Nill. Se ambos argumentos forem identificadores de versão de um mesmo objeto versionado, $\eta_b(bop, v_1, v_2)$ pode ser `true` ou `false` dependendo da sua posição na árvore de derivação. Se não pertencerem ao mesmo objeto, $\eta_b(bop, v_1, v_2)$ resulta sempre em `false`. Todos esses resultados são do tipo *Bool* por VT-Bool.

Lema 3.4 *(Operações Unárias) Se* $vsch \vdash_{vsch} (vee, voe, ve)$, $K \vdash v_1 : \sigma_1$ *e* $\delta_u(uop, \sigma_1) = \sigma_2$, *então* $\eta_u(uop, v_1) = v_2$ *e* $K \vdash v_2 : \sigma_2$.

Prova. Pela definição de *uop*.

Caso $uop \equiv$ `-`

Prova similar ao mesmo caso no lema 2.4 (Operações Unárias em cOQL).

Caso $uop \equiv$ `not`

Prova similar ao mesmo caso no lema 2.4 (Operações Unárias em cOQL).

Caso $uop \equiv$ `set`

Prova similar ao mesmo caso no lema 2.4 (Operações Unárias em cOQL).

Caso $uop \equiv$ `size`

Prova similar ao mesmo caso no lema 2.4 (Operações Unárias em cOQL).

Caso $uop \equiv$ `is_leaf`, $uop \equiv$ `is_root`, $uop \equiv$ `is_current`

Pela definição de δ_u, $\delta_u(uop, c) = Bool$. Portanto, o valor v_1 é do tipo c, podendo ser ou `nill` ou *vid*. Se for `nill`, $\eta_u(uop, v_1) =$ `nill`, sendo `nill` do tipo *Bool* por VT-Nill. Se for *vid*, $\eta_u(uop, v_1)$ pode ser `true` ou `false` dependendo da sua posição na árvore de derivação, sendo o resultado do tipo *Bool* por VT-Bool.

Caso $uop \equiv$ `first`

Pela definição de δ_u, $\delta_u(uop, \sigma_a * \sigma_b) = \sigma_a$. Pelo sistema de tipos, ou v_1 é `nill` ou então é um par (v_a, v_b). Se v_1 for `nill`, $\eta_u(uop,$ `nill`$) =$ `nill`, sendo o resultado do tipo σ_a por VT-Nill. Caso $v_1 \equiv (v_a, v_b)$, então v_1 é tipado por VT-Pair. Sendo assim, $\eta_u(uop, (v_a, v_b)) = v_a$, sendo v_a do tipo σ_a pelas premissas da regra VT-Pair.

Caso $uop \equiv$ `second`

Pela definição de δ_u, $\delta_u(uop, \sigma_a * \sigma_b) = \sigma_b$. Pelo sistema de tipos, ou v_1 é `nill` ou então é um par (v_a, v_b). Se v_1 for `nill`, $\eta_u(uop,$ `nill`$) =$ `nill`, sendo o resultado do tipo σ_b por VT-Nill. Caso $v_1 \equiv (v_a, v_b)$, então v_1 é tipado por VT-Pair. Sendo assim, $\eta_u(uop, (v_a, v_b)) = v_b$, sendo v_b do tipo σ_b pelas premissas da regra VT-Pair.

Lema 3.5 *(Substituição) Se* $vsch; vee; voe; ve; \Gamma \vdash v : \sigma_v$, $vsch; vee; voe; ve; \Gamma, x : \sigma_v \vdash q : \sigma$ *e* $x \notin \Gamma$, *então* $vsch; vee; voe; ve; \Gamma \vdash q[x ::= v] : \sigma$.

Prova. Por indução sobre a estrutura de q.

Casos $q \equiv b$, $q \equiv i$, $q \equiv s$, $q \equiv void$, $q \equiv vid$, $q \equiv nill$, $q \equiv x'$, $\quad x' \neq x$

Triviais, já que $vsch; vee; voe; ve; \Gamma, x : \sigma_v \vdash q : \sigma$ e $q[x ::= v] = q$.

Caso $q \equiv x$

A estrutura da prova é similar à estrutura da prova do mesmo caso no lema 2.5 (Substituição em cOQL).

Caso $q \equiv \{q_1, \ \ldots, \ q_n\}$

A estrutura da prova é similar à estrutura da prova do mesmo caso no lema 2.5 (Substituição em cOQL).

Caso $q \equiv q_1, q_2$

A consulta q é tipada por VT-Pair

$$\frac{vsch; vee; voe; ve; \Gamma, x : \sigma_v \vdash q_1 : \sigma_1 \qquad vsch; vee; voe; ve; \Gamma, x : \sigma_v \vdash q_2 : \sigma_2}{vsch; vee; voe; ve; \Gamma, x : \sigma_v \vdash q_1, q_2 : \sigma_1 * \sigma_2}$$

Pela definição da operação de substituição,

$$(q_1, q_2)[x ::= v] = q_1[x ::= v], q_2[x ::= v]$$

Pelas premissas de VT-Pair, tanto q_1 quanto q_2 são bem-tipados. Pela hipótese indutiva, $vsch; vee; voe; ve; \Gamma \vdash q_1[x ::= v] : \sigma_1$ e $vsch; vee; voe; ve; \Gamma \vdash q_2[x ::= v] : \sigma_2$. Desta forma, $vsch; voe; ve; ve; \Gamma \vdash q_1, q_2 : \sigma_1 * \sigma 2$ por VT-Pair.

$$\frac{vsch; vee; voe; ve; \Gamma \vdash q_1[x ::= v] : \sigma_1 \qquad vsch; vee; voe; ve; \Gamma \vdash q_2[x ::= v] : \sigma_2}{vsch; vee; voe; ve; \Gamma \vdash q_1[x ::= v], q_2[x ::= v] : \sigma_1 * \sigma_2}$$

Caso $q \equiv q_1 \ bop \ q_n$

A estrutura da prova é similar à estrutura da prova do mesmo caso no lema 2.5 (Substituição em cOQL).

Caso $q \equiv uop \ q_1$

A estrutura da prova é similar à estrutura da prova do mesmo caso no lema 2.5 (Substituição em cOQL).

Caso $q \equiv q_1.a$

A estrutura da prova é similar à estrutura da prova do mesmo caso no lema 2.5 (Substituição em cOQL).

Caso $q \equiv q_1$`->`a

A estrutura da prova é similar à estrutura da prova do mesmo caso no lema 2.5 (Substituição em cOQL).

Caso $q \equiv q_1$`->versions`

A consulta q é tipada por VT-Versions.

$$\frac{vsch; vee; voe; ve; \Gamma, x : \sigma_v \vdash q : Bag(c)}{vsch; vee; voe; ve; \Gamma, x : \sigma_v \vdash q\texttt{->versions} : Bag(c)}$$

Pela definição da operação de substituição,

$$(q\texttt{->versions})[x ::= v] = (q[x ::= v])\texttt{->versions}$$

Pelas premissas de VT-Pair, q_1 é bem-tipado. Pela hipótese indutiva, $vsch; vee; voe; ve; \Gamma \vdash q_1[x ::= v] : Bag(c)$. Desta forma, $vsch; voe; ve; ve; \Gamma \vdash q_1$`->versions` $: Bag(c)$ por VT-Versions.

$$\frac{vsch; vee; voe; ve; \Gamma \vdash q[x ::= v] : Bag(c)}{vsch; vee; voe; ve; \Gamma \vdash q[x ::= v]\texttt{->versions} : Bag(c)}$$

Caso $q \equiv$ `if` q_1 `then` q_2 `else` q_3

A estrutura da prova é similar à estrutura da prova do mesmo caso no lema 2.5 (Substituição em cOQL).

Caso $q \equiv$ `select` q_a `from` $q_1 \ x_1, \ \ldots, \ q_n \ x_n$ `where` q_b

A estrutura da prova é similar à estrutura da prova do mesmo caso no lema 2.5 (Substituição em cOQL).