GEMS - UM GErenciador de Módulos de Software

Augusto César Sampaio, Hermano Perrelli de Moura,
Max Cavalcanti de Albuquerque, Silvio Lemos Meira

Departamento de Informática
Universidade Federal de Pernambuco
50739, Recife, PE, Brasil

RESUMO

Utilizamos o método *me too* na especificação e prototipagem do GEMS, um GErenciador de Módulos de Software, que, em particular, foi desenvolvido como um modelo semântico para o gerenciamento de módulos da notação Zc. *me too* engloba uma notação abstrata, para a fase de especificação, e uma notação concreta (executável), que permite o desenvolvimento rápido de protótipos.

1 INTRODUÇÃO

O método *me too* [HenMin, Martin 86] possibilita o uso de uma notação abstrata (*me too* abstrato) na fase de especificação e de uma mais concreta (*me too* concreto) no desenvolvimento de protótipos. *me too* abstrato e concreto têm uma base matemática sólida (semântica simples e bem definida) e são suficientemente próximos, de tal forma que o processo de transformação das especificações formais em protótipos é relativamente simples e pode ser formalizado. Na realidade, o termo *me too* engloba as notações abstrata e concreta e um método de especificação/prototipagem, que é apresentado na Seção 2.

Este trabalho apresenta uma especificação e protótipo do GEMS - um GErenciador de Módulos de Software, que foi desenvolvido como um possível modelo para administrar a importação/exportação de objetos entre os módulos da notação Zc [SamMei 88]. Apesar de ser um sistema relativamente complexo, tanto a especificação como o protótipo do GEMS em *me too* são simples e concisos, devido à expressividade da notação.

2 ME TOO

me too é uma linguagem de especificação baseada numa combinação de programação funcional, em particular Miranda [Turner 87], e uma parte construtiva da teoria dos conjuntos. Como em VDM [Jones 86], *me too* usa a noção de que software pode ser especificado através da construção de

um modelo matemático para o mesmo.

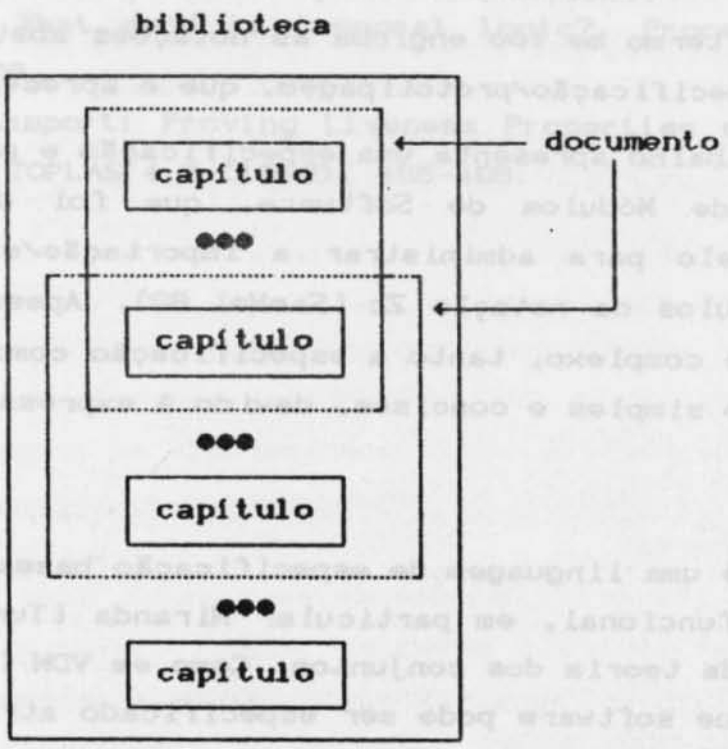
Para caracterizar os objetos do modelo abstrato, dispomos dos seguintes objetos matemáticos pré-definidos em *me too*: conjuntos, seqüências, tuplas, relações binárias e funções finitas, que possuem as características usuais. Além destes, há uniões, que são objetos que podem assumir valores de tipos distintos, embora num determinado instante assumam um e um só valor de um dos tipos possíveis. No Apêndice A apresentamos algumas das principais operações sobre os objetos pré-definidos.

3 ESPECIFICAÇÃO INFORMAL

Nesta Seção, descrevemos a idéia de módulos em Zc e apresentamos o GEMS informalmente, através de uma descrição dos objetos e das operações que o constituem.

3.1 MÓDULOS EM Zc

A notação Zc é parte de um projeto do Grupo de Especificação e Programação Funcional do Departamento de Informática da UFPE, que visa construir um ambiente para desenvolvimento de *software* a partir de especificações em Zc. Zc oferece mecanismos de suporte a uma abordagem modular de desenvolvimento de especificações de grande porte, como mostrado a seguir.



Um capítulo é a unidade sintática que agrupa definições de constantes, tipos, operadores e esquemas.[1] Um capítulo pode importar definições de (e exportar para) outros capítulos. Um documento é formado pelos nomes (referências) dos diversos capítulos que constituem a especificação global do sistema e, portanto, representa a especificação do sistema como um todo. Uma biblioteca é constituída de capítulos e documentos. O ambiente de desenvolvimento de especificações em Zc pode conter diversas bibliotecas.

É importante ressaltar que, fisicamente, um capítulo está em uma biblioteca. O documento contém apenas referências a capítulos. Neste sentido, há uma independência entre capítulos e documentos, ou seja, tanto é possível haver capítulos que fazem parte de mais de um documento, como pode haver capítulos que não fazem parte de nenhum documento, simplesmente contêm definições que, eventualmente, são utilizadas por outros capítulos. Apesar dos documentos serem arquivos armazenados em bibliotecas junto com os capítulos, consideramos, por simplicidade de projeto, que há bibliotecas distintas para capítulos e documentos.

O compartilhamento de elementos (constantes, tipos, operadores e esquemas) entre os capítulos é realizado explicitamente por declarações de importação/exportação. Os capítulos que constituem um dado documento podem importar elementos de capítulos que, efetivamente, não são referenciados pelo documento. Estes capítulos são chamados de capítulos secundários, pois fazem parte da especificação de forma indireta. Os capítulos secundários são inferidos pelo GEMS. Voltamos a este assunto na Seção 4.

As informações relevantes de um capítulo para o desenvolvimento do GEMS são a interface e os identificadores dos elementos definidos no capítulo. Portanto, é irrelevante a definição destes elementos e não há necessidade de separá-los em tipos, constantes, operadores e esquemas.

A interface de um capílulo oferece uma flexibilidade considerável quanto à importação/exportação de elementos. Podemos resumir a importação em dois tipos: alguns elementos (explicitamente especificados) ou todos os elementos. Desta forma, cada declaração de importação contém o nome do capítulo do qual se deseja importar e uma

---

[1] Esquemas agrupam declarações de variáveis e um predicado que as relaciona.

opção: os elementos a serem importados (no primeiro tipo de importação) ou a palavra ALL, significando que todos os elementos devem ser importados. No segundo tipo de importação, todos os elementos visíveis pelo capítulo são importados, ou seja, os elementos definidos no capítulo e os que porventura tenham sido importados de outros capítulos (importação transitiva).

3.2 O GEMS

O GEMS é composto de dois subsistemas: o gerenciador de compilação e o gerenciador de visibilidade de elementos. O gerenciador de compilação é responsável pela "automação" do processo de compilação. Os capítulos são compilados com referência à biblioteca na qual estão definidos. Contudo, pode-se restringir o contexto a um documento. As operações disponíveis no gerenciador de compilação incluem

- Geração de uma estrutura que representa o relacionamento de importação entre os capítulos de uma biblioteca. Esta operação cria uma estrutura mais adequada às outras operações do gerenciador de compilação.
- Geração de uma estrutura idêntica à anterior, restrita a capítulos de um documento.
- Indicação dos capítulos secundários de um documento.

As operações que se seguem são aplicáveis tanto a uma biblioteca como a um documento.

- Indicação do conjunto de capítulos a serem recompilados por alteração de um dado capítulo.
- Indicação de uma possível seqüência de compilação para os capítulos gerados pela operação anterior.
- Relação de dependência (quanto à compilação), por nível, de um dado capítulo.
- Indicação dos capítulos cuja modificação pode provocar a compilação de um dado capítulo.

O gerenciador de visibilidade de elementos oferece várias operações que facilitam identificar o fluxo de elementos entre capítulos. As operações que se seguem são definidas a partir de uma biblioteca de capítulos, independente dos documentos que referenciam

estes capítulos. As operações disponíveis são:

- Geração de uma biblioteca de capítulos expandida, a partir de uma biblioteca. Esta operação substitui as opções ALL em todas as declarações de importação dos capítulos da biblioteca pelos respectivos conjuntos de elementos. A biblioteca expandida facilita a definição das outras operações.
- Indicação dos elementos importados por um capítulo, com as respectivas origens.
- Indicação dos elementos definidos em um capítulo.
- Indicação dos elementos visíveis (definidos e importados) por um dado capítulo.
- Indicação dos elementos importados de um dado capítulo para outro.
- Verificar se um dado elemento é ou não visível por um capítulo.
- Indicação da origem de um elemento visível por um capítulo.

4 Especificação Formal

Passamos agora à especificação do GEMS em *me too*. Por questão de espaço, não a apresentamos na íntegra . Uma versão completa pode ser encontrada em [SMAM 88], que inclui o protótipo do GEMS.

4.1 Objetos

Dentro da biblioteca de capítulos, conforme explicado anteriormente, cada capítulo é uma entidade à qual podemos fazer referência univocamente através de seu nome. Daí podemos estruturar esta biblioteca através de uma função finita que mapeia o nome do capítulo em seu conteúdo:

BibCap = ff (RefCap, InfCap)

Cada capítulo é uma tupla de dois componentes: um conjunto (set) de declarações de importação e a definição dos elementos do capítulo. Cada declaração de importação, por sua vez, possui uma referência a um capítulo (de onde ele importa) e, ou um conjunto de elementos que se deseja importar ou a cláusula ALL. A definição dos elementos de um

capítulo é, simplesmente, um conjunto de referências a estes elementos. Podemos então definir:

InfCap = tup (Importacao, Definicao)
Importacao = set (DecImp)
DecImp = tup (RefCap, Opcao)
Opcao = set (RefElem) U ALL
Definicao = set (RefElem)

Na biblioteca de documentos, cada documento pode ser referenciado de uma forma única, razão pela qual optamos por representá-la através de uma função finita, que associa a cada documento um conjunto de capítulos que o constitui.

BibDoc = ff (RefDoc, set (RefCap))

Embora **BibCap** e **BibDoc** sejam os principais objetos do sistema, duas estruturas mais adequadas são geradas, e servem de base para as operações dos subsistemas de compilação e de visibilidade de elementos. Essas estruturas são as representações do fluxo de importação (**Grafo**) e da expansão da biblioteca de capítulos (**BibCapExp**).

Em **BibCap**, estão os capítulos que se relacionam (ou não) entre si através de cláusulas de importação. Esse relacionamento equivale a uma estrutura de importação ligando os diversos capítulos, a qual possui as características de um grafo direcionado, onde, para cada aresta de um nó n_1 para um nó n_2, significa que n_2 importa algo de n_1 direta ou indiretamente, ou seja, o sucessor da aresta é o capítulo que importa e o antecessor é o que exporta.

É importante salientar que o grafo de uma biblioteca pode não ser conexo, pois a relação de importação entre os capítulos é totalmente arbitrária (com a exceção de que não pode formar grafos cíclicos) e, portanto, pode haver capítulos (nós) sem ligação alguma. Por exemplo, um capítulo que não importa de (e nem exporta para) outro capítulo. Assim, a estrutura que modela o grafo de importação é uma relação de referências de capítulo:

Grafo = rel (RefCap, RefCap)

A outra estrutura principal é a biblioteca de capítulos expandida, que é uma estrutura igual a **BibCap** com a cláusula ALL substituída pela relação textual dos elementos que são visíveis no capítulo importado, pelo acréscimo de tuplas de importação do capítulo importado, e assim

recursivamente, pela transitividade da cláusula ALL.

BibCapExp = ff (RefCap, InfCapExp)
InfCapExp = tup (ImpExp, Definicao)
ImpExp = set (ListaImp)
ListaImp = tup (RefCap, set (RefElem)

Portanto, ao invés do segundo componente de **ListaImp** ser uma opção por um conjunto de elementos ou pela cláusula ALL, temos apenas um **set (RefElem)**.

Alguns objetos que representam estruturas elementares do GEMS não foram, propositadamente, definidos, pois, neste nível de especificação, é interessante manter uma certa abstração. A definição de **RefCap**, por exemplo, como **Nat** ou **seq (Char)** é um detalhe de implementação.

4.2 INVARIANTES

Modelados os objetos, devemos estabelecer algumas propriedades para manter a consistência do sistema, através de invariantes. Por exemplo:

- Todos os capítulos referenciados pelos documentos (da biblioteca de documentos) devem estar definidos na biblioteca de capítulos.

  union (ran BibDoc) ⊆ dom BibCap

O conjunto completo dos invariantes pode ser encontrado em [SMAM 88].

4.3 OPERAÇÕES

A seguir, daremos as especificações das operações através da notação abstrata do método *me too*.

4.3.1 GERENCIADOR DE COMPILAÇÃO

Dada uma biblioteca de capítulos, devemos gerar uma estrutura que contenha o relacionamento de importação entre estes capítulos. Essa estrutura, já discutida anteriormente, é o ponto de partida para todas as outras operações deste subsistema, já que ela modela a biblioteca de um modo adequado o suficiente para tornar as operações posteriores mais compreensíveis. Portanto, temos a operação:

geragrafo : BibCap ⟶ Grafo

```
geragrafo(bcap) ≡ {(a, b)| b ← dom bcap ; a ← conjexp(bcap, b)}
conjexp(bcap, rcap) ≡ {first t| t ← first bcap[rcap]} ∪
                      union {conjexp(bcap, c)| c ← conjALL(bcap, rcap)}
conjALL(bcap, rcap) ≡ {first t| t ← first bcap[rcap] ;
                                   second t = ALL}
```

que tem como parâmetro uma biblioteca de capítulos e produz o grafo de importação equivalente. De um modo geral, essa operação gera todos os pares (a,b) onde a exporta para b direta ou indiretamente. Daí, pode-se tirar b do domínio da biblioteca de capítulos e emparelhá-lo com todos os elementos que fazem parte do conjunto de capítulos da parte de importação de b (conjexp). Esse conjunto é formado pela simples aplicação da biblioteca ao capítulo, obtendo-se uma tupla, e retirando-se o primeiro elemento que é o conjunto Importacao. Daí, extrai-se, de novo, de cada elemento deste conjunto o primeiro elemento, formando assim o conjunto dos capítulos dos quais o capítulo em questão importa.

Podemos querer apenas o grafo da relação de importação de um documento específico. Pode-se observar que o grafo resultante é um subgrafo do grafo da biblioteca de capítulos, que será composto pelos nós referentes aos capítulos descritos no documento, e mais os capítulos por eles referenciados que não fazem parte explicitamente do documento.

```
grafodoc : Grafo x BibDoc x RefDoc → Grafo

grafodoc(gr, bdoc, rdoc) ≡ grafodocaux(gr, bdoc[rdoc])
grafodocaux(gr, crc) ≡ if crc = {} then {}
                       else
                        let grd = {t| t ← gr ; second t ∈ crc}
                        in grd ∪ grafodocaux(gr, (dom grd)-crc)
```

grafodoc é definida em termos de uma função auxiliar (grafodocaux), que tem como parâmetros o grafo e o conjunto de capítulos pertencentes ao documento dado. Essa função auxiliar reúne todos os pares retirados do grafo da biblioteca de capítulos, tal que o segundo elemento destes pares pertença ao conjunto dos capítulos que formam o documento, e mais os pares que envolvam o conjunto de capítulos que são apenas referenciados no documento, ou seja, que exportam direta ou indiretamente para os capítulos do documento (chamada recursiva de grafodocaux). Foi usada a construção let.. in.. que, como usual, permite definições locais.

A operação que se segue tem como resultado o conjunto dos capítulos que são referenciados (secundários) dentro de um documento dado, mas que, efetivamente, não fazem parte dele:

```
secundarios : Grafo x BibDoc x RefDoc → set (RefCap)

secundarios(gr, bdoc, rdoc) ≡ let grd = grafodoc(gr, bdoc, rdoc)
                              in (dom grd)-bdoc[rdoc]
```

secundarios utiliza o grafo da biblioteca de capítulos para aplicar **grafodoc** e obter o grafo do documento. Os capítulos secundários são a diferença entre o domínio da relação (grafo) e os capítulos componentes do documento.

Uma operação fundamental deste subsistema é a que, dados o grafo da relação de importação e um capítulo, produz o conjunto de capítulos a serem recompilados por alteração deste capítulo.

```
recomp : Grafo x RefCap → set (RefCap)

recomp(gr, rcap) ≡ projr(rcap, gr)
```

Observe que o grafo argumento para a operação acima pode tanto ser o grafo da biblioteca de capítulos como um grafo específico para um documento. O mesmo vale para as operações seguintes.

Uma outra operação importante é uma extensão da anterior, pois agora não se quer o conjunto de capítulos a serem recompilados, mas uma possível seqüência de compilação destes capítulos.

```
seqcom : Grafo x RefCap → seq (RefCap)

seqcom(gr, rcap) ≡ seqcor(gr, rcap, recomp(gr, rcap))
seqcor(gr, rcap, crcap) ≡ if crcap = {} then {}
                          else
                           let sx = <x| x ← crcap ;
                                (crcap-{x}) ∩ nos(gr, rcap, x) = {}>
                           in
                           sx ↑ seqcor(gr, rcap, crcap-elems sx)
```

Dado o conjunto de recompilação, **recomp(gr, rcap)**, calcula-se, primeiramente, a seqüência de capítulos, retirados do conjunto de capítulos que devem ser recompilados, tais que não existam nós intermediários no caminho entre eles e o nó alterado, pois, se existirem, estes nós devem ser compilados primeiro. Calculada essa seqüência, ela deve ser concatenada (através do operador ↑) com uma nova aplicação dessa função auxiliar, porém com o parâmetro referente ao conjunto de capítulos a serem recompilados sem os elementos da

seqüência formada pela primeira aplicação. Foi usada, na definição, uma outra função auxiliar, nos, que retorna os nós no caminho entre dois capítulos.

Além das operações já descritas, o gerenciador de compilação dispõe ainda das operações reldep e importa. reldep retorna uma função finita, onde a cada natural está relacionado um conjunto de capítulos. Isso significa que, se alterarmos e recompilarmos um dado capítulo, a operação reldep fornece o conjunto de capítulos que devem ser recompilados em primeiro lugar (em qualquer ordem), em segundo lugar, etc. importa retorna o conjunto dos capítulos que podem fazer com que um dado capítulo precise ser recompilado. Em outras palavras, devolve os capítulos dos quais um dado capítulo importa elementos direta ou indiretamente. A seguir mostramos a funcionalidade dessas operações:

```
reldep : Grafo x RefCap → ff (Nat, set (RefCap))

importa : Grafo x RefCap → set (RefCap)
```

4.3.2 O Gerenciador de Visibilidade de Elementos

O gerenciador de visibilidade de elementos tem como objetivo prover o usuário do GEMS de um conjunto de informações acerca da origem e da visibilidade dos elementos de um dado capítulo. Esse subsistema está baseado num objeto do tipo BibCapExp. Tal objeto é gerado a partir de um objeto do tipo BibCap através da função expandedoc especificada a seguir. BibCapExp é uma estrutura semelhante a BibCap, com a diferença de que todas as opções ALL foram substituídas pelos devidos elementos. A definição de expandedoc segue um raciocínio semelhante ao usado na definição da operação geragrafo.

```
expandedoc : BibCap → BibCapExp

expandedoc(bcap) ≡
  [rcap: dom bcap → (conjimp(bcap, rcap), second bcap[rcap])]
conjimp(bcap, rcap) ≡
        {t | t ← first bcap[rcap]; not (second t = ALL)}
        ∪
        union {conjimp(bcap, c) ∪ {(c, second bcap[c])}
                | c ← conjALL(bcap, rcap)}
```

Dado um capítulo, é importante saber que elementos este capítulo importa com suas respectivas origens. elemimp fornece estes resultados:

```
elemimp : BibCapExp x RefCap → ImpExp
```

elemimp(bcapexp, rcap) ≡ first bcapexp[rcap]

Os elementos definidos em um capítulo são dados pela operação elemdef:

elemdef : BibCapExp x RefCap → set (RefElem)

elemdef(bcapexp, rcap) ≡ second bcapexp[rcap]

Um dado importante é quais elementos são visíveis -definidos e importados- por um capítulo. Tais elementos são gerados por elemvistos.

elemvistos : BibCapExp x RefCap → set (RefElem)

elemvistos(bcapexp, rcap) ≡ second bcapexp[rcap] ∪
union {second x | x ← first bcapexp[rcap]}

Dado um capítulo, é relevante saber quais elementos este capítulo importa de um capítulo dado. elemimpcap retorna tais elementos:

elemimpcap : BibCapExp x RefCap x RefCap → set (RefElem)

elemimpcap(bcapexp, rcapd, rcapo) ≡
≡ the {second x | x ← first bcapexp[rcapd] ;
first x = rcapo}

onde rcapd (capítulo destino) importa ou não elementos de rcapo (capítulo origem).

O gerenciador de visibilidade de elementos dispõe ainda das operações visivel e origem. visivel indica se um dado elemento é visível por um capítulo. origem nos diz onde está definido um elemento de um capítulo. A seguir, damos a funcionalidade dessas operações:

visivel : BibCapExp x RefCap x RefElem → Bool

origem : BibCapExp x RefCap x RefElem → seq (Char)

5 CONCLUSÃO

Uma importante característica de *me too* é a uniformidade das notações abstrata e concreta, o que torna o processo de transformação de especificações formais em protótipos uma tarefa relativamente simples, pois há um mapeamento direto entre os tipos nas duas notações. Portanto os protótipos mantêm a abstração da especificação, o que os torna bastante flexíveis. O GEMS, em particular, pode ser rapidamente modificado tanto para incluir novas operações específicas para Zc, como para o gerenciamento de módulos de software com uma semântica diferente da apresentada aqui.

Este trabalho está inserido em um contexto maior - a definição formal da semântica de Zc, que será descrita usando semântica denotacional. Devido à complexidade de usar semântica denotacional para o gerenciamento de módulos, optamos pela definição de um modelo a partir do qual pudéssemos construir uma especificação e um protótipo do GEMS. Apesar de não possuir a formalidade de uma definição que usa semântica denotacional, o desenvolvimento do GEMS em *me too* possibilitou uma análise mais concreta do compartilhamento de elementos em Zc, através de vários testes realizados no protótipo, que foi concluído e está funcionando.

Além disso, a substituição dos invariantes definidos sobre o modelo do GEMS por operações para os casos de erro, resulta um protótipo de parte de um compilador para Zc, em particular a que trata capítulos e documentos. Observe ainda que o gerenciador de visibilidade de elementos constitui um módulo de um editor que será desenvolvido para Zc, pois as operações lá definidas são de grande utilidade durante o processo de edição de uma especificação.

REFERÊNCIAS

[HenMin] Henderson P. and Minkowitz C.: "The *me too* Method of Software Design". Dep. Comput. Sci., Univ. Stirling, Stirling, Scotland, Rep. FPN/10 (available from the author).

[Jones 86] Jones C. B.: "Systematic Software Development Using VDM". Prentice-Hall Intl., UK, 1986.

[Martin 86] Martins F. M.: "Especificação Formal e Prototipificação de *Software* - A Metodologia *me too*". Notas de Aula, Universidade do Minho, Portugal, 1986.

[SamMei 88] Sampaio A. C. A. e Meira S. L.: "Zc: Uma Notação para Especificação de Sistemas Complexos". Anais do XV SEMISH, Congresso da SBC, Rio de Janeiro, Julho/88.

[SMAM 88] Sampaio, A. C., Moura, H. P., Albuquerque, M. C. e Meira, S. L.: "Um Modelo Semântico para os Módulos de Zc". RT-DI/UFPE 012/88, Depto. de Informática, UFPE, Recife, PE, 1988.

[Turner 87] Turner D.: "Miranda System Manual". Research Software Limited. Canterbury, England, 1987.

APÊNDICE A. TABELAS DAS OPERAÇÕES PRÉ-DEFINIDAS EM ME TOO

Conjuntos. Na tabela abaixo A e B são conjuntos; x, y e z são elementos de conjuntos e α é uma variável de tipo.

set (α)	- todos os subconjuntos finitos cujos elementos são do tipo α
{x, y,...,z}	- enumeração de um conjunto
{} ou ∅	- conjunto vazio
x ∈ A	- true, se x é membro de A
A ↑ B	- intersecção; {x | x ∈ A ∧ x ∈ B}
A ∪ B	- união; {x | x ∈ A ∨ x ∈ B}
A - B	- diferença; {x | x ∈ A ∧ ¬(x ∈ B}
A ≤ B	- true, se ∀ x ∈ A . x ∈ B
A = B	- true, se A ≤ B ∧ B ≤ A
A × B	- {(x,y) | x ∈ A ∧ y ∈ B}
card A	- número de elementos de A
union X...Y	- união distribuída; X ∪ ...∪ Y
inter X...Y	- intersecção distribuída; X ↑ ...↑ Y
{e(x) | x ← A}	- {e(x) | x ∈ A}
{e(x) | x ← A; p(x)}	- {e(x) | x ∈ A ∧ p(x) ↔ true}
the A	- fornece um x qualquer pertencente a A

Seqüências. Na tabela abaixo, l, l1 e l2 são seqüências; e, e1,...,en são elementos de uma seqüência e k é do tipo Nat.

seq (α)	- todas as seqüências finitas de elementos do tipo α
⟨e1,...,en⟩	- enumeração de seqüência
NIL ou ⟨⟩	- seqüência vazia
l1 ↑ l2	- concatenação de duas seqüências
hd l ou head (l)	- seleciona o primeiro elemento de uma seqüência
tl l ou tail (l)	- retorna uma seqüência com todos os elementos de l, exceto o primeiro.
inv l	- dá a seqüência invertida
len l	- número de elementos da seqüência l
elems l	- conjunto dos elementos de l
l [k]	- k-ésimo elemento de uma lista
⟨e(x) | x ← l⟩	- ⟨e(l1),...,e(l len l)⟩
⟨e(x) | x ← l; p(x)⟩	- ⟨e(lk) | lk ← l e p(lk) = true⟩

Tuplas. Na tabela seguinte, t é uma tupla, e1,...,en são componentes de uma tupla e α1, α2, αn são variáveis de tipo.

tup (α1,α2,...,αn)	- todas as tuplas cujos componentes têm tipos α1, α2, , αn, nesta ordem.
[e1,...,en]	- construção de uma tupla por enumeração
first t	- seleciona o primeiro componente da tupla t
second t	- seleciona o segundo componente da tupla t

Relações Binárias. Na tabela abaixo mostramos algumas operações sobre relações. Sejam α e β variáveis de tipo.

rel (α, β)	- todas as relações cujos elementos do domínio são do tipo α e da imagem são do tipo β.
{⟨a1,b1⟩,...,⟨an,bn⟩}	- construção por enumeração
{} ou ∅	- relação vazia
join (r,s)	- {⟨a,c⟩ | ⟨a,b⟩ ∈ r; ⟨b',c⟩ ∈ s; b = b'}
dom r	- {a | ⟨a,b⟩ ∈ r}
ran r	- {b | ⟨a,b⟩ ∈ r}
projl (r,y)	- projeção a esquerda; {x | ⟨x,y⟩ ∈ r}
projr (x,r)	- projeção a direita; {y | ⟨x,y⟩ ∈ r}
r dr A	- restrição do domínio; {⟨a,b⟩ ∈ r | a ∈ A}
r ds A	- subtração do domínio; {⟨a,b⟩∈ r|a ∈(dom r- A)}

Funções Finitas.

ff (α, β)	- funções finitas de α para β.
[a → b]	- enumeração; {⟨a,b⟩}
[]	- função finita vazia
dom f	- domínio de f
ran f	- contradomínio de f
f[a] ou f[a]d	- aplicação; se a ∈ dom f então f(a) senão d
f ⊛ g	- sobreposição; {a → g(a) | a ∈ dom g} ∪ {a → f(a) | a ∈ (dom f - dom g)}
f dr A	- restrição do domínio;{a → f(a) | a ∈ (dom f ↑ A)}
f ds A	- subtração do domínio;{a → f(a)|a ∈ (dom f - A)}
[a:A → e]	- {a → e(a) | a ∈ A}
[a:A → e; p]	- {a → e(a) | a ∈ A; p(a)}