1. INTRODUÇÃO

A idéia de implementação é crucial no desenvolvimento de programas por refinamentos sucessivos. Uma implementação envolve descrever os conceitos de um nível, mais abstrato, em termos dos conceitos do nível seguinte, mais concreto. Linguagens de programação costumam fornecer mecanismos para esta etapa: módulos consistindo de procedimentos baseados em uma representação. A representação é feita através de construtores de sortes, como registro, sub-intervalo, etc., que descrevem como novos sortes são construídos a partir de sortes disponíveis. O propósito deste trabalho é apresentar especificações formais para alguns construtores de sortes similares àqueles encontrados nas linguagens de programação.

O enfoque lógico para especificações formais trata-as como apresentações de teorias. Uma especificação é uma teoria em uma linguagem, usualmente poli-sortida, de primeira ordem apresentada por um conjunto de axiomas. Neste caso, o processo de implementar uma especificação formal em outra pode ser descrito em termos de conceitos simples de lógica relacionando as especificações formais envolvidas [Vel87].

O resultado de implementar uma especificação abstrata ***A*** em termos de outra especificação concreta ***C***, é um módulo que:

representa os objetos de ***A*** em termos dos objetos de ***C*** ;

implementa as operações e predicados de ***A*** por meio de procedimentos usando as operações e predicados de ***C***.

A representação dos objetos de ***A*** em termos dos de ***C*** é geralmente feita descrevendo os sortes da especificação ***A*** através de sortes construídos usando os de ***C***. Linguagens de programação usualmente provêm alguns construtores de sortes, como enumeração, registro, subsorte, "variant record" e união discriminada.

Mas, a representação dos objetos também pode ser pensada como uma especificação de como os sortes de ***A*** estão conectados aos sortes concretos de ***C***. Isto é análogo à maneira pela qual os textos dos procedimentos, que implementam as operações e predicados de ***A***, são especificados por seu comportamento de entrada-saída. Neste último caso são elaboradas definições (talvez incompletas) das operações e predicados de ***A*** em termos de aqueles de ***C***.

A abstração descrita acima nos leva ao conceito de implementação de especificações formais. Uma especificação formal é uma apresentação de uma teoria em lógica poli-sortida de primeira ordem. Uma implementação de uma especificação formal ***A*** em outra especificação formal ***C*** é uma interpretação de ***A*** em uma extensão conservativa ***B*** de ***C*** [Vel87]. A idéia aqui é que a especificação concreta ***C*** seja estendida até que possa incorporar versões concretas nas quais os símbolos abstratos de ***A*** sejam representados, mas sem acrescentar qualquer propriedade aos símbolos de ***C*** (já que eles estavam pré-definidos).

Este processo envolve examinar como os novos símbolos, junto com suas propriedades, podem ser acrescentados a uma especificação formal já existente. Uma extensão consiste do acréscimo de novos símbolos e axiomas, um caso simples e especial, mas muito importante, é o das chamadas extensões por definição [Sho67]. Os novos símbolos a serem introduzidos correspondem tanto a símbolos de função e predicado quanto a símbolos de sortes. Extensões por acréscimo de símbolos de função e de predicado estão bem estudados na literatura (por exemplo, [Sho67]). Em particular, extensões por definição de símbolos de função e de predicado são caracterizadas pelas propriedades de Conservatividade e Eliminabilidade.

Neste trabalho especificaremos construtores que introduzem novos sortes. A novidade aqui aparece em dois aspectos, um no que se refere a encarar a extensão por acréscimo de sorte como um mecanismo construtor (que introduz um sorte novo) e o outro em que é

apresentado como uma especificação parametrizada. No trabalho serão examinados alguns construtores similares aos encontrados em linguagens de programação: produto cartesiano (correspondente a registro), união discriminada, subsorte (que corresponde ao sub-intervalo) e quociente ([Hoa74]). Este último construtor, quociente, mesmo não sendo costumeiro nas linguagens de programação, corresponde à representação da igualdade no sorte abstrato, que é necessário para implementações corretas [TM87,Vel87]. Analisar este gênero de construção é de importância porque descreve como sortes novos são construídos a partir de certos sortes dados, o que ocorre na implementação de especificações formais, quando sortes novos são representados em termos de sortes concretos.

Na próxima seção apresentaremos, de maneira informal, cada um dos quatro construtores de sortes, que fornecem novos sortes construídos a partir de certos sortes dados. Já na seção seguinte, serão apresentadas para eles especificação formais que visam a captar a visão intuitiva que se tem deles. Na seção 4 apresentamos várias propriedades que estas especificações possuem, o que mostra que elas realmente têm o comportamento desejado. Na seção 5 tratamos do emprego das especificações apresentadas na seção anterior como especificações parametrizadas, tanto por instanciação a casos particulares, quanto em construção e verificação de programas. Finalmente, na seção 6 apresentamos algumas conclusões e perspectivas de trabalhos em andamento.

2. CONSTRUTORES DE SORTES

Nesta seção descreveremos, de uma maneira um tanto informal, os quatro construtores de sortes em que estamos interessados. Estas descrições, na meta-linguagem, seguem de perto as apresentadas por Hoare [Hoa74], sendo baseadas em construções usuais de teoria de conjuntos. Discutiremos também certos aspectos da sintaxe e da semântica desses construtores.

Na seção seguinte trataremos de formalizar estas descrições em linguagens de primeira ordem apropriadas.

2.1 DESCRIÇÃO DOS CONSTRUTORES

O construtor de sorte *produto* introduz um novo sorte cujos objetos são os registros (pares ordenados) de objetos dos dois sortes dados junto com as correspondentes funções de seleção. Assim, temos um novo conjunto $C = S_1 \times S_2$, junto com suas projeções $p_k : C \rightarrow S_k$, que selecionam as componentes dos pares ordenados: $p_k(<a_1, a_2>) = a_k$.

O construtor de sorte *união discriminada* introduz um novo sorte formado pelos objetos de cada um dos dois sortes dados. Assim, temos um novo conjunto $U = S_1 + S_2$ (união disjunta). Mas, já que existe a possibilidade de termos diferentes representações dos objetos, introduzimos duas funções injetivas de inserção $i_k : S_k \rightarrow U$ para efetuarem a conversão dos elementos.

O construtor de sorte *subsorte* introduz um novo sorte que é um subconjunto do sorte dado. Este subconjunto é formado pelos objetos do sorte original que satisfazem uma certa propriedade expressa pelo *predicado de relativização*. Assim, temos um novo conjunto $N = \{ a \in S ; r(a) \}$. Já que os elementos de este conjunto novo N não precisam ser representados da mesma forma que os elementos do conjunto S, devemos introduzir uma função injetiva de inserção $j : N \rightarrow S$ para realizar a conversão dos elementos.

O construtor de sorte chamado *quociente* introduz um novo sorte formado pelas classes de equivalência dos objetos do sorte dado sob uma relação de equivalência definida nele. Assim, temos um conjunto novo $M = \{ [a/q] ; a \in S \}$. Neste caso, é natural introduzir uma função sobrejetora de projeção no conjunto quociente $p : S \rightarrow M$ associando a cada elemento a de S sua classe de equivalência $[a/q]$ segundo a relação q.

2.2 SINTAXE E SEMANTICA DOS CONSTRUTORES

A nível da sintaxe, podemos dizer que cada construtor de sorte gera, a partir de uma linguagem *L*', uma nova linguagem estendida *L*" com o sorte introduzido e alguns símbolos de funções de conversão que conectam o sorte novo com os antigos. Além disto, como supomos que cada sorte *S* tem associado a ele seu símbolo de igualdade $\approx_S$, esta extensão também acrescenta o símbolo de igualdade que corresponde ao novo sorte. Por exemplo, o construtor do sorte *produto* parte de uma linguagem *L*' com sortes S_1 e S_2, com seus respectivos símbolos de igualdade, e gera uma nova linguagem *L*" com novo sorte *C* e as projeções $p_1 : C \rightarrow S_1$ e $p_2 : C \rightarrow S_2$, junto com o símbolo de igualdade $\approx_C$ sobre o novo sorte *C*.

Já a nível da semântica, é bastante natural ver (a semântica de) cada construtor de sorte *constr* como uma operação **CONSTR** sobre estruturas: ela expande uma estrutura da linguagem antiga *L*', acrescentando um conjunto novo e funções de conversão, gerando uma estrutura para a linguagem nova *L*".

Mas, estas operações podem ter certas pré-condições para sua aplicação correta. Por exemplo, uma das operações de construção que temos é **SUBSRT**, correspondente ao construtor *subsorte*. Esta operação produz um conjunto novo N formado pelos elementos do conjunto dado S que satisfazem certo predicado de relativização r. Mas já que, os sortes devem corresponder a conjuntos não vazios (como é usualmente exigido em lógica), só poderemos aplicar esta operação **SUBSRT** quando o predicado de relativização r seja realizado por um conjunto não vazio.

Assim, parece natural dizer que a semântica de cada construtor de sorte é uma operação parcial **CONSTR** sobre estruturas. Desta maneira, atingimos uma noção muito similar à de *tipos parametrizados de dados.* Por exemplo, a operação **SUBSRT** teria como parâmetro um único tipo de dados, formado por uma estrutura da linguagem antiga *L*' que satisfaz a condição de aplicabilidade (ou seja, que a realização do predicado de relativização seja um conjunto não vazio).

Consideramos, contudo, preferível uma abordagem alternativa, que apresenta a vantagem de tratar todo o procedimento num só nível, o das propriedades expressas na linguagem, internalizando as construções.

Essa abordagem descreve cada construtor de sorte como uma *especificação formal parametrizada* ***CONSTR***, isto é, como a apresentação de uma teoria na linguagem estendida *L*", que é uma extensão conservativa de seu parâmetro: uma apresentação ***PARAM*** de uma teoria na linguagem antiga *L*' que descreve a condição de aplicabilidade. Para nós, uma especificação ***A*** é *parametrizada* por uma sub-especificação ***P*** (chamada de *parâmetro*) se e só se ***A*** é uma extensão conservativa de ***P***.

Por exemplo, para o construtor de sorte *produto*, a linguagem antiga *L*' tem só dois símbolos de sorte S_1 e S_2 (junto com seus respectivos símbolos de igualdades) e a linguagem estendida *L*" terá, além dos símbolos de *L*', um símbolo para o novo sorte *C* e os símbolos para as projeções p_1 e p_2 (além do símbolo para a igualdade $\approx_C$ sobre o novo sorte *C*). Já para o construtor *subsorte*, onde existe uma pré-condição a ser satisfeita, a apresentação parâmetro deverá ter a condição de aplicabilidade expressa na

linguagem L', além de símbolos para o sorte S (e sua igualdade) e o predicado de relativização r.

Desta maneira, a aplicação de um construtor de sorte para introduzir um novo sorte em uma especificação dada processa-se pela identificação, na linguagem dada, de seus argumentos de modo a que seja garantida sua condição de aplicabilidade. Em outras palavras, a aplicação de um construtor de sorte *constr* a uma especificação dada ***SPEC*** vem a ser uma instanciação de sua apresentação parâmetro ***PARAM*** a ***SPEC***.

3. ESPECIFICAÇÃO DOS CONSTRUTORES

Nesta seção daremos, para cada construtor de sorte informalmente apresentado na seção anterior, uma especificação formal como uma apresentação parametrizada. Nos dois primeiros casos, os construtores de produto e união discriminada, a apresentação do parâmetro consistirá de sua linguagem junto com um conjunto vazio de axiomas, já que não há pré-condições a serem satisfeitas. Por outro lado, nos dois últimos construtores, subsorte e quociente, a apresentação parâmetro terá axiomas que expressam a pré-condição para aplicá-las.

Em cada caso, apresentaremos primeiro a especificação correspondente ao parâmetro ***PARAM***, expressando (quando necessárias) as condições de aplicabilidade, para logo depois apresentar a especificação parametrizada ***CONSTR*** correspondente ao construtor. Lembremos, que para nós, uma *especificação* é uma apresentação de uma teoria, isto é, uma linguagem poli-sortida de primeira ordem L' junto com um conjunto G de axiomas que são sentenças de L'.

3.1 O CONSTRUTOR PRODUTO

Para o construtor produto, a especificação parâmetro ***PRM_P***. tem na sua linguagem L'***_P*** dois símbolos de sortes S_1 e S_2, bem como de dois símbolos de predicados binários para as igualdades $\approx_{S_1}$, sobre sorte S_1, e $\approx_{S_2}$, sobre sorte S_2. Seu conjunto G'***_P*** de axiomas é vazio, já que não há pré-condições para construir o produto cartesiano de dois conjuntos.

Agora vamos descrever a apresentação da extensão ***PROD***. Sua linguagem L''***_P*** consiste de L'***_P*** mais o novo símbolo de sorte C, junto com um símbolo de predicado binário $\approx_C$ sobre sorte C, bem como dois símbolos de operações unárias $p_1 : C \to S_1$ e $p_2 : C \to S_2$. Seu conjunto G''***_P*** de axiomas consiste das duas sentenças seguintes da linguagem L''***_P***:

(PJS) $\quad (\forall x_1{:}S_1)\,(\forall\ x_2{:}S_2)\,(\exists\ y{:}C)\,[\,p_1(y) \approx_{S_1} x_1 \wedge p_2(y) \approx_{S_2} x_2\,]$

(PJI) $\quad (\forall y,y'{:}C)\,[\,p_1(y) \approx_{S_1} p_1(y') \wedge p_2(y) \approx_{S_2} p_2(y') \to y \approx_C y']$

O axioma (PJS) afirma que ambas as projeções são sobrejetivas e o axioma (PJI) que as projeções são conjuntamente injetivas.

3.2 O CONSTRUTOR UNIÃO DISCRIMINADA

A especificação parâmetro ***PRM_U*** para o construtor união discriminada é como a do produto. A especificação parâmetro ***PRM_U*** tem na sua linguagem L'***_U*** dois símbolos de sortes S_1 e S_2 (com seus símbolos de predicados binários para igualdades $\approx_{S_1}$, sobre sorte S_1, e $\approx_{S_2}$, sobre sorte S_2). Seu conjunto G'***_U*** de axiomas é vazio, já que não há pré-condições para construir a união disjunta de dois conjuntos.

Passemos agora à apresentação da extensão ***UNIÃO_DSCR***. Sua linguagem L''***_U*** consiste de L'***_U*** mais o novo símbolo de sorte U, junto com um símbolo de predicado

binário $\approx_U$ sobre sorte U, e dois símbolos de operações unárias $i_1 : S_1 \to U$ e $i_2 : S_2 \to U$. Seu conjunto G"$_\boldsymbol{U}$ de axiomas consiste das quatro sentenças seguintes da linguagem L"$_\boldsymbol{U}$:

(IJS) $(\forall y{:}U)\ [\ (\exists x_1{:}S_1)\ y \approx_U i_1(x_1) \vee (\exists\ x_2{:}S_2)\ y \approx_U i_2(x_2)\ ]$

(IDI) $(\forall x_1{:}S_1)\ (\forall\ x_2{:}S_2) \neg\ i_1(x_1) \approx_U i_2(x_2)$

(I1I) $(\forall x_1,x'_1{:}S_1)\ [\ i_1(x_1) \approx_U i_1(x'_1) \to\ x_1 \approx_{S_1} x'_1\ ]$

(I2I) $(\forall x_2,x'_2{:}S_2)\ [\ i_2(x_2) \approx_U i_2(x'_2) \to\ x_2 \approx_{S_2} x'_2\ ]$

O axioma (IJS) afirma que as imagens das inserções cobrem o novo sorte, enquanto que o axioma (IDI) estabelece que estas imagens são disjuntas. Os axiomas (I1I) e (I2I) expressam que cada inserção é injetiva.

3.3 O CONSTRUTOR SUBSORTE

Tratemos primeiro da apresentação parâmetro ***PRM_S*** . Sua linguagem L'$_\boldsymbol{S}$ tem um símbolo de sorte S, junto com um símbolo de predicado binário $\approx_S$ sobre sorte S, e um símbolo de predicado unário (de relativização) r sobre sorte S. Seu conjunto G'$_\boldsymbol{S}$ de axiomas consiste da seguinte sentença da linguagem L'$_\boldsymbol{S}$:

(NVR) $(\exists x{:}S\ )\ r(x)$

O axioma (NVR) expressa que o predicado de relativização (mais precisamente, sua realização) deve ser um conjunto não vazio.

Passemos agora à extensão ***SUBSRT***. Sua linguagem L"$_\boldsymbol{S}$ consiste de L'$_\boldsymbol{S}$ mais um novo símbolo de sorte N, junto com um símbolo de predicado binário $\approx_N$ correspondente à igualdade no sorte N, e um símbolo de operação unária $j : N \to S$. O conjunto G"$_\boldsymbol{S}$ de axiomas consiste das duas sentenças seguintes da linguagem L"$_\boldsymbol{S}$:

(IRP) $(\forall x{:}S)\ [\ r(x) \leftrightarrow (\exists\ y{:}N)\ x \approx_S j(y)\ ]$

(II) $(\forall y,y'{:}C)\ [\ j(y) \approx_S j(y') \to\ y \approx_N y'\ ]$

O axioma (IRP) estabelece que a imagem da inserção é idêntica à extensão do predicado de relativização, enquanto que o axioma (II) expressa que a inserção é injetiva.

Mais adiante, em 5.2, discutiremos como acrescentar novas conversões a esta e às outras especificações.

3.4 O CONSTRUTOR QUOCIENTE

Comecemos com a apresentação parâmetro ***PRM_Q***. Sua linguagem L'$_\boldsymbol{Q}$ tem um símbolo de sorte S, junto com um símbolo de predicado binário $\approx_S$ sobre sorte S, e um símbolo de predicado binário q sobre sorte S. Seu conjunto G'$_\boldsymbol{Q}$ de axiomas consists consiste das três sentenças seguintes de L'$_\boldsymbol{Q}$:

(R) $(\forall x{:}S)\ q(x,x)$

(S) $(\forall x,x'{:}S)\ [\ q(x,x') \to q(x',x)\ ]$

(T) $(\forall x,x',x''{:}S)\ \{\ [\ q(x,x') \wedge q(x',x'')\ ] \to\ q(x,x'')\ \}$

Estes três axiomas estabelecem que o símbolo de predicado q deve ser realizado por uma relação de equivalência, pois (R), (S) e (T) expressam, respectivamente, reflexividade, simetria e transitividade.

Vejamos agora a apresentação da extensão ***QUOT***. Sua linguagem L"***_Q*** é a linguagem L'***_Q*** mais um novo símbolo de sorte M, junto com um símbolo de predicado binário $\approx_M$ correspondendo à igualdade no sorte M, e um símbolo de operação unária $p : S \to M$. O conjunto G"***_Q*** de axiomas consiste das duas sentenças seguintes de L"***_Q***:

(SP) $(\forall y{:}M)\ (\exists x{:}S)\ p(x) \approx_M y$

(PQ) $(\forall x,x'{:}S)\ [\ p(x) \approx_M p(x') \leftrightarrow q(x,x')\ ]$

O axioma (SP) expressa a sobrejetividade da projeção sobre o quociente, enquanto que o axioma (PQ) afirma que a projeção de um elemento de S consiste exatamente dos elementos a ele relacionados pela relação q.

4. ADEQUAÇÃO DAS ESPECIFICAÇÕES

Nesta seção discutiremos propriedades das especificações parametrizadas apresentadas na seção anterior. Algumas destas propriedades indicam que as especificações apresentadas são adequadas, no sentido de que elas exibem o comportamento intuitivo desejado. Outras propriedades mostram que estas extensões têm um bom comportamento do ponto de vista lógico; em particular que elas exibem comportamento similar ao das extensões por definição explícita de novos símbolos de predicado e de função.

Aqui a ênfase será no significado das propriedades. As propriedades serão apresentadas apenas com referências para as respectivas demonstrações.

Além disto, as especificações apresentadas para cada um dos quatro construtores satisfazem todas as propriedades que enunciaremos. Por isso não faremos distinções especiais para cada caso específico, a não ser a título de exemplo.

4.1 CARACTERIZAÇÃO E ADEQUAÇÃO DOS CONSTRUTORES

Trataremos primeiro de algumas propriedades que caracterizam os modelos das especificações ***CONSTR*** em relação aos modelos das especificações parâmetros ***PARAM.*** Assim, estas propriedades estabelecem o comportamento das especificações.

Uma primeira propriedade caracteriza os modelos de ***CONSTR*** como expansões de modelos de ***PARAM***. Como exemplo consideremos o caso do construtor *produto*. Cada modelo **P** de ***PROD*** tem um conjunto $C^{\mathbf{P}}$ realizando o novo sorte C. Não se pode garantir que este novo conjunto seja realmente o produto cartesiano dos conjuntos realizando os sortes S_1 and S_2 da linguagem L'***_P***. Pode-se, contudo, garantir que, com as funções $p_1^{\mathbf{P}}$ e $p_2^{\mathbf{P}}$ realizando as projeções, o novo conjunto apresenta o comportamento deste conjunto de pares ordenados (registros), conforme desejado. De maneira análoga, temos a seguinte caracterização, formulada mais precisamente, para o caso de quociente.

Proposição: *Caracterização dos modelos de* ***QUOT***

Uma estrutura **Q** da linguagem L"***_Q***, é modelo de ***QUOT*** sse

seu reduto à linguagem L'***_Q*** é modelo de ***PRM_Q***, e

ela é isomorfa à espansão canônica deste reduto através do quociente com sua projeção.

Para cada um dos nossos quatro construtores - *produto*, *união discriminada*, *subsorte* e *quociente* - especificados na seção 3, temos uma *caracterização* similar que estabelece que uma estrutura da linguagem estendida L" é um modelo de ***CONSTR*** se e só se seu reduto à linguagem menor L' é um modelo de ***PARAM*** e o sorte novo é isomorfo a um

conjunto, obtido dos conjuntos do reduto, que tem o comportamento desejado [MV92]. Isto mostra sua *adequação* em relação ao seu significado pretendido, descrito em termos de teoria de conjuntos na seção 2.

Em consequência, podemos afirmar que as extensões tratadas são *expansivas*, logo *conservativas*, i.e. preservam todas as propriedades da especificação ***PARAM***.

Teorema: *Expasividade única e Conservatividade*

Para cada um construtor ***CONSTR*** especificado na seção 3, tem-se:

cada modelo de ***PARAM*** tem uma expansão única (salvo isomorfismos) a um modelo de ***CONSTR***; e

a extensão de ***PARAM*** a ***CONSTR*** é conservativa.

4.2 PROPRIEDADES DE INVARIANCIA DOS CONSTRUTORES

Outra propriedade interessante, também a nível de modelos, é que eles são *invariantes sob isomorfismo*, i.e., dadas duas estruturas da linguagem estendida *L*" que são modelos de ***CONSTR***, sempre que seus redutos à linguagem menor *L*' sejam isomorfos, podemos obter um (único) isomorfismo entre eles. O análogo desta propriedade para o caso, mais simples, das extensões por definição de novos símbolos de predicado e de função é o conhecido Teorema de Beth [Sho67]. Na verdade, esta nossa propriedade decorre de outra, um pouco mais geral, que se refere ao c*aráter funtorial* dos modelos: dadas duas estruturas da linguagem estendida *L*" que são modelos de ***CONSTR***, cada homomorfismo entre seus redutos à linguagem menor *L*' induz um (único) homomorfismo entre as estruturas originais ([MV92]).

No que se refere às especificações vistas como extensões lógicas podemos citar as propriedades de definibilidade, de invariância sob equivalência elementar e eliminabilidade. Estas propriedades juntas, estabelecem o fato de que, mesmo não sendo extensões por definição de novos símbolos, as extensões por acréscimo de símbolos de sortes comportam-se como elas, o que corrobora o nosso sentimento intuitivo.

A propriedade da *definibilidade* garante que qualquer sentença da linguagem estendida *L*", sem variáveis livres sobre o novo sorte, é equivalente a uma sentença na linguagem menor *L*'.

Além disto, estabelecemos uma noção geral de eliminabilidade, chamada *eliminabilidade sob conexão*, mais adequada para o caso de teorias poli-sortidas. Neste caso, qualquer sentença da linguagem estendida *L*" é equivalente a uma sentença na linguagem menor *L*' módulo uma conexão estabelecida entre suas variáveis do novo sorte com variáveis dos sortes originais. Na verdade, esta conexão entre as variáveis, codifica a tradução das variáveis do sorte novo à linguagem original [MV94]. Em [MV94] provamos que as quatro especificações apresentadas satisfazem esta noção generalizada de eliminabilidade. Tomemos como exemplo o caso do construtor ***PROD***. Aqui, cada variável y sobre o novo sorte *C* é conectada a duas variáveis x_1 e x_2, sobre os sortes S_1 e S_2, através da conexão projetiva $p_1(y) \approx_{S_1} x_1 \wedge p_2(y) \approx_{S_2} x_2$. Então, qualquer sentença da linguagem estendida *L*" (onde podem ocorrer variáveis, ligadas, sobre o novo sorte *C* usando as projeções) é equivalente a uma sentença na linguagem menor *L*' (onde aparecem só variáveis sobre os sortes S_1 e S_2). Para fórmulas, tal equivalência é módulo a conexão. A propriedade de eliminabilidade conhecida para o caso mono-sortido ([Sho67]), onde não existem variáveis percorrendo diferentes sortes, resulta ser um caso particular desta.

No que se refere à propriedade de *invariância sob equivalência elementar*, temos que duas estruturas da linguagem estendida *L*" que são modelos de ***CONSTR***, serão elementarmente equivalentes sempre que seus redutos à linguagem menor *L*' o forem [MV94].

5. APLICAÇÃO DOS CONSTRUTORES

Na seção anterior vimos que as especificações dos construtores dadas na seção 3 são adequadas. Vamos agora examinar dois aspectos da aplicação dessas especificações. O primeiro se refere ao seu uso a especificações particulares. O segundo diz respeito mais diretamente a seu emprego em programação: construção e verificação de programas.

5.1 APLICAÇÃO DOS CONSTRUTORES POR INSTANCIAÇÃO

Até agora examinamos os contrutores de novos sortes de maneira isolada. Porém, graças ao Teorema da Modularização [PV93], sua aplicação a cada caso particular não apresenta maiores percalços.

Vamos ilustrar essa idéia com o caso de subsorte. Suponhamos que temos uma especificação ***SPEC*** que já tem um sorte T e um símbolo de predicado unário d sobre este sorte T. Desejamos extendê-la introduzindo um novo sorte que seja o subsorte de T corresponde ao predicado d como relativização. Isto pode ser feito por aplicação do mecanismo geral ***SUBSRT*** como se segue.

Inicialmente, identificamos os argumentos através de uma tradução $t : L'_\boldsymbol{S} \rightarrow L'_\boldsymbol{SPEC}$ entre a linguagem do parâmetro de ***SUBSRT*** e a de ***SPEC***, associando ao sorte S o sorte T, e ao predicado de relativização r o predicado d. Mostrando que em ***SPEC*** vale a condição de aplicabilidade $(\exists v{:}T)\ d(v)$, verificamos que esta tradução de linguagens é uma interpretação de ***PRM_S*** em ***SPEC***. O Teorema da Modularização faz o resto do trabalho.

De maneira geral, tudo o que precisamos fazer para aplicar um construtor de sorte ***CONSTR*** para introduzir um sorte novo numa especificação dada ***SPEC***, é identificar os elementos do parâmetro formal ***PARAM*** com seus correspondentes na linguagem da especificação dada e garantir as condições de aplicabilidade. Assim, uma tradução de linguagens identifica os elementos do parâmetro formal ***PARAM*** com os correspondentes no argumento real ***SPEC***. A verificação das condições de aplicabilidade assegura que a tradução realmente interpreta teorias: ***PARAM*** em ***SPEC***. O Teorema da Modularização garante que a extensão de ***SPEC*** pelo construtor ***CONSTR*** seja conservativa.

5.2 OS CONSTRUTORES EM PROGRAMAÇÃO

O emprego dos construtores de sortes em programação está relacionado à idéia básica de métodos de desenvolvimento de programas com base nas estruturas de dados, como o de Jackson [Luc82].

Vamos ilustrar essas idéias com o caso do produto, que introduz registros. Um programa, ou procedimento, que tenha dentre suas entradas objetos de sorte produto precisa processá-los. Para isso servem as projeções: para decompor um objeto do sorte produto em seus componentes, que serão processados de acordo com seus sortes. Assim, as projeções $p_1 : C \rightarrow S_1$ e $p_2 : C \rightarrow S_2$ permitem que se definam novas funções com argumentos no sorte produto. Examinemos agora o caso em que a saída é de sorte produto. Supondo que os componentes desta saída já tenham sido obtidos, precisamos formar o registro com estes componentes. Ou seja, precisamos de uma função binária de S_1 e S_2 no sorte produto C. Nossa especificação ***PROD*** não provê tal função, porém, graças aos axiomas (PJS) e (PJI), ela pode ser facilmente introduzida por definição: $mk_rec(x_1,x_2) \approx_C y \leftrightarrow p_1(y) \approx_{S_1} x_1 \wedge p_2(y) \approx_{S_2} x_2$.

No caso de subsorte, já temos a inserção $j : N \rightarrow S$ e precisamos de uma conversão $conv : S \rightarrow N$. Podemos introduzi-la pelo axioma $conv\ (j\ (y)) \approx_N y$. Embora não seja uma

definição, trata-se de uma extensão conservativa, que garante que *conv* se comporte como inversa da inserção sobre a extensão do predicado de relativização, o que é suficiente: seu comportamento é irrelevante fora desta região. Os casos de quociente e de união discriminada são similares ao de subsorte.

6. CONCLUSÕES

Construtores de novos sortes usando sortes disponíveis aparecem em linguagens de programação e são importantes para a implementação de abstrações [Vel87]. Quatro construtores usuais são: produto (consistindo de registros), união discriminada (consistindo de alternativas), subsorte (análogo a sub-intervalo) e quociente (consistindo de classes de equivalência). As descrições intuitivas de [Hoa74] foram formalizadas como especificações para estes mecanismos definindo o comportamento pretendido. Estas especificações têm a característica de especificações parametrizadas por outras onde estão os sortes base e que expressam as condições de aplicabilidade porventura necessárias.

Em cada caso, o sorte introduzido pelo mecanismo é único, a menos de isomorfismo. Isto difere do caso clássico de introdução por definição de novos símbolos de função e de predicado, onde temos expansão única. Além disso, as extensões clássicas por definição têm a propriedade da eliminabilidade: qualquer fórmula da linguagem estendida é equivalente a alguma fórmula na linguagem original. No nosso caso de introdução de novos sortes, não podemos esperar uma eliminabilidade total, já que temos novas variáveis sobre o sorte introduzido. Mas, ainda é possível ter uma eliminabilidade generalizada: eliminabilidade sob conexão.

Um mecanismo freqüente em linguagens de programação, e que não foi tratado aqui, é o de enumeração. Este difere dos anteriores por não necessitar de uma base para introduzir o novo sorte: ele é criado diretamente por enumeração exaustiva de seus elementos. Não é difícil especificá-lo e verificar que tem propriedades similares aos quatro aqui tratados.

Temos boas indicações de que qualquer mecanismo de introdução de sorte que seja funtorial, no sentido mencionado em 4.2, pode ser obtido a partir desses cinco. Isto estabeleceria uma completude do conjunto de construtores produto, união discriminada, subsorte, quociente e enumeração. Trabalhos em andamento visam a examinar esta conjectura e suas implicações.

Uma característica comum a esses cinco construtores é a de só produzirem sortes finitos a partir de sortes finitos. Para as necessidades de implementação é conveniente dispor também de outros, similares aos existentes em certas linguagens de programação. Alguns, como conjunto das partes ou mapeamentos (semelhante a "array"), ainda preservam finitude, mas outros, como seqüências, permitem gerar sortes infinitos (enumeráveis) a partir de sortes finitos. A investigação mas detalhadas de tais construtores é objeto de trabalhos que estão sendo iniciados. Estão sendo examinados também aspectos de computabilidade: construtores que preservam decidibilidade.

REFERÊNCIAS

[Hoa74] C. A. R. Hoare. Notes on Data Structuring. In O. J. Dahl, E. W. Dijkstra & C. A. R. Hoare (eds.), *Structured Programming,* Academic Press, 1974. p. 83 - 174.

[Luc82] C. J. P. de Lucena. *Análise e Síntese de Programas de Computador*. Editora Universidade de Brasília, 1982.

[MV92] M. C. Meré & P. A. S. Veloso. On Extensions by Sorts. *Monografias em Ciência da Computação,* No. 38, Departamento de Informática, PUC-Rio, 1992.

[MV94] M. C. Meré & P. A. S. Veloso. Definition-like Extensions by Sorts. A aparecer como *Monografias em Ciência da Computação,* Departamento de Informática, PUC-Rio. Também *Resumo submetido a WoLLIC'94*, Pernambuco, julho de 1994.

[PV93] H. W. Poubel & P. A. S. Veloso. Sobre o Teorema da Modularização: importância e prova por quociente. *VII Simpósio Brasileiro de Engenharia de Software*. Rio de Janeiro, outubro de 1993,p.17-29.

[Sho67] J. R. Shoenfield. *Mathematical Logic*. Addison-Wesley Publishing Company, 1967.

[TM87] W. M. Turski & T. S. E. Maibaum. *The Specification of Computer Programs*. Addison-Wesley Publishing Company, 1987.

[Vel87] P. A. S. Veloso. *Estruturação e Verificação de Programas com Tipos de Dados.* Editora E. Blücher, São Paulo, 1987.