

Sistemas de Tipos para Módulos, Objectos e Classes (I)

Mestrado em Engenharia Informática
2000/2001 (Luís Caires)

Mestrado em Engenharia Informática 2001/2002

Garantias de composição

- Componentes
 - bibliotecas / módulos / classes
- (inter)Ligação de Componentes
 - estática (mais usual)
 - dinâmica (DL, JVM)
- Garantias de Correção da composição
 - A composição produz um componente com certas garantias (por ex., ausência de certos tipos de erros de execução)
- Verificação de Garantias
 - Estática versus Dinâmica

Mestrado em Engenharia Informática 2001/2002

Sistemas de Tipos

- Um programa é seguro quando não executa nenhuma operação inválida
 - Ex: soma de um inteiro com um booleano
 - Ex: chamada de um método não definido
 - Ex: divisão por zero ?
- Sistemas de Tipos
 - asseguram propriedades de correção
 - impedem operações inválidas
- Programa bem tipificado \Rightarrow Programa seguro
- Programa seguro \Rightarrow Programa bem tipificado?

Mestrado em Engenharia Informática 2001/2002

Tipos

- Classificam “valores”
 - valores básicos
 - valores funcionais (código): funções
 - objectos
 - módulos / classes
- Indicam que operações estão bem definidas sobre as entidades que classificam
 - inteiro: operações aritméticas
 - funções : aplicação a certos argumentos
 - módulo: importação / composição
 - objecto: chamada de método

Mestrado em Engenharia Informática 2001/2002

Tipo abstracto de dados (ADT)

- Um conjunto T (de valores)
- Um conjunto OPS de operações envolvendo valores de tipo T (a “interface” do ADT)
- Os valores de tipo T só podem ser manipulados pelas operações definidas em OPS, as quais satisfazem as propriedades impostas por um conjunto de axiomas
- Deste modo
 - É impossível ao “cliente” do ADT conhecer (e por isso alterar ou modificar) a representação interna dos valores do mesmo: garantias de correcção impostas pela abstracção

Mestrado em Engenharia Informática 2001/2002

Tipo abstracto de dados (ADT)

- *ADT STACK*
- *OPS*
 - new : \rightarrow STACK
 - push : $\text{integer} \times \text{STACK} \rightarrow \text{STACK}$
 - pop : $\text{STACK} \rightarrow \text{integer}$
 - empty?: $\text{STACK} \rightarrow \text{bool}$
- *AXIOMS*
 - empty?(new()) = true
 - empty?(push(n,s)) = false
 - pop(push(n,s)) = n

Mestrado em Engenharia Informática 2001/2002

Implementação de ADTs

- Um nome novo T (o nome do ADT, ex: STACK)
- Uma representação interna (estrutura de dados) para os valores de tipo T, construída eventualmente com base noutros ADTs.
- Uma implementação para cada operação declarada em OPS, recorrendo à representação interna
- Uma associação entre as operações em OPS e a sua implementação. Esta associação pode ser implícita (por nome) ou explícita (por ligação).

Mestrado em Engenharia Informática 2001/2002

ADTs e Linguagens de Programação

- Inicialmente, grande parte das linguagens de programação não suportava directamente a implementação de ADTs (exemplo: ANSI C)
 - ADTs: também são uma ferramenta de modelação / metodologia de programação poderosa
- A maior parte das linguagens de programação modernas suporta ADTs através do uso de abstracções específicas (módulos, classes, etc)
 - única maneira “decente” de estruturar software em componentes.

Mestrado em Engenharia Informática 2001/2002

Módulos (Estrutura Geral)

- Parte pública (“interface” exportada)
 - um conjunto de declarações de
 - importação de outros módulos
 - tipos (declaração parcial)
 - valores
 - funções
- Parte privada (“implementação”)
 - um conjunto de declarações de
 - valores
 - tipos (declaração completa)
 - funções
 - inicialização

Mestrado em Engenharia Informática 2001/2002

Interface do tipo STACK em Modula 3

```
INTERFACE Stacks;  
TYPE Stack = ^StackCell;  
TYPE StackCell = RECORD  
    val: Item;  
    next: Stack  
END  
PROCEDURE NewStack():Stack;  
PROCEDURE Push(item:Item,VAR s:Stack)  
PROCEDURE Pop(VAR s:Stack):Item;  
PROCEDURE Empty(s:Stack):boolean  
END Stacks.
```

Implementação do Tipo STACK em Modula 3

```
MODULE Stacks;  
PROCEDURE NewStack():Stack;  
BEGIN  
    RETURN NIL;  
END;  
PROCEDURE Empty(s:Stack):boolean;  
BEGIN  
    RETURN (s=NIL)  
END; ...  
END Stacks.
```

Uso do Tipo STACK em Modula 3

```
MODULE myProgram;  
IMPORT Stacks;  
  
VAR s:Stack;  
  
BEGIN  
    s := NewStack();  
    Push(s,1);  
    ...  
    Write (s.val);  
END Stacks.
```

Uso do Tipo STACK em Modula 3

```
MODULE myProgram;
IMPORT Stacks;

VAR s:Stack;

BEGIN
  s := NewStack();
  Push(s,1);
  ...
  Write (s.val);
END Stacks.
```

TIPO TRANSPARENTE

Interface do ADT STACK em Modula 3

```
INTERFACE Stacks;

TYPE Stack <: REFANY;
PROCEDURE NewStack():Stack;
PROCEDURE Push(item:Item,VAR s:Stack)
PROCEDURE Pop(VAR s:Stack):Item;
PROCEDURE Empty(s:Stack):boolean

END Stacks.
```

Implementação do ADT STACK em Modula 3

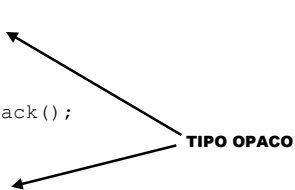
```
MODULE Stacks;
TYPE Stack = ^StackCell;
TYPE StackCell = RECORD
  val: Item;
  next: Stack
END
PROCEDURE Empty(s:Stack):boolean;
BEGIN
  RETURN (s=NIL)
END; ...
END Stacks.
```

Uso do ADT STACK em Modula 3

```
MODULE myProgram;  
IMPORT Stacks;  
  
VAR s:Stack;  
  
BEGIN  
  s := NewStack();  
  Push(s,1);  
  ...  
  (* Write (s.val); *)  
END Stacks.
```

Uso do ADT STACK em Modula 3

```
MODULE myProgram;  
IMPORT Stacks;  
  
VAR s:Stack;  
  
BEGIN  
  s := NewStack();  
  Push(s,1);  
  ...  
  (* Write (s.val); linha incorrecta *)  
END Stacks.
```



Tipos para Objectos, Classes e Componentes

- Tipos básicos
 - Classificam valores de tipo primitivo
 - booleanos, strings
 - semântica de cópia por valor
- Tipos para Objectos
 - Classificam o comportamento dos objectos
 - Tipo de Objecto = Classe (C++)
 - Tipo de Objecto = Interface (Sather)
 - Tipo de Objecto = Classe, Interface (Java)
- Tipos para Módulos, Classes, Componentes?

Sistemas de Tipos OO: Interfaces

- Garantias de correcção e composição pretendidas:
 - Um programa seguro **nunca** invoca um método sobre um objecto a que este não saiba responder
 - `obj.draw(here)`
 - `self.put(2)`
- Interface
 - em geral,
 - especifica um conjunto de métodos a que o objecto ao qual a interface se aplica sabe responder
 - pode apresentar outras informações adicionais (informação semântica, dependências, etc...)

Mestrado em Engenharia Informática 2001/2002

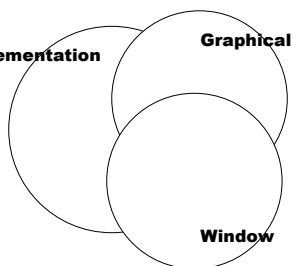
Sistemas de Tipos OO: Interfaces

- Cada método tem um **nome** e uma **assinatura**
 - $AM ::= (T_1, T_2, \dots, T_n)T$
 - Exemplos de assinaturas
 - `(int,int)int`
 - `(IObserver)void`
- Tipos Interface
 - $TI ::= [m_1:AM_1, m_2:AM_2, \dots, m_n:AM_n]$
 - m_i são identificadores (nomes) de método
 - $m_i = m_j \Rightarrow AM_i = AM_j$ (sobrecarga / overloading)
 - Ex: `[attach:(IObserver)void, detach:(IObserver)void]`

Mestrado em Engenharia Informática 2001/2002

Sistemas de Tipos OO: Polimorfismo

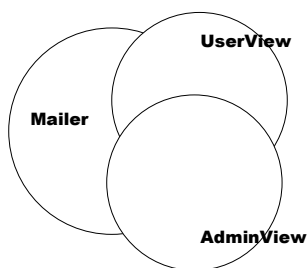
Window_Implementation



Graphical :> Window :> Window_Implementation

Mestrado em Engenharia Informática 2001/2002

Sistemas de Tipos OO: Polimorfismo



Mailer :> **UserView**; **Mailer** :> **AdminView**

Mestrado em Engenharia Informática 2001/2002

Polimorfismo e Ordenação de Interfaces

- Interfaces I e J

Diz-se que $I <: J$ (I subtipo de J)

quando I define **pelo menos** todas as operações que J define:

$$\begin{array}{c} [m_1:A_1, m_2:A_2, \dots, m_n:A_n] \\ <: \\ [m_1:A_1, m_2:A_2, \dots, m_n:A_n, n_1:B_1, n_2:B_2, \dots, n_k:B_k] \end{array}$$

Mestrado em Engenharia Informática 2001/2002

Polimorfismo e Ordenação de Interfaces

- $I <: J$
- Intuição: objectos do subtipo I podem substituir objectos do tipo J em qualquer contexto de uso
- Princípio da Substitutividade:
 - se $P[a]$ com $a:I$ é seguro, $I <: J$ e $b:J$, então $P[b]$ também é seguro
- Esta é a noção de subtipo **estrutural**
- Apesar de $I <: J$ bastar para garantir a segurança, a maior parte das linguagens de programação com objectos require mais ! (discussão)

Mestrado em Engenharia Informática 2001/2002

Polimorfismo e Ordenação de Interfaces

- Container == [add : (Element)void]
- Window == [draw : ()void, add : (Element)void]
 - [draw:()void, add:(Element)void] <: [add:(Element)void]
- Panel <: Element
 - [draw:()void, add:(Panel)void] <: [add:(Element)void] ?
- Como definir a relação de subtipo sobre as assinaturas de métodos (discussão) ?

Mestrado em Engenharia Informática 2001/2002

Sistemas de Tipos OO: Polimorfismo

- Container == [add : (Element)void]
- Window == [draw : ()void, add : (Element)void]
 - [draw:()void, add:(Element)] <: [add:(Element)] ?
- Panel <: Element
 - [draw:()void, add:(Panel)] <: [add:(Element)] ?
- Como definir a relação de subtipo sobre as assinaturas de métodos?
 - $A <: B \Rightarrow [m:A, \dots] <: [m:B, \dots]$ (covariante)
 - $A <: B \Rightarrow [m:B, \dots] <: [m:A, \dots]$ (contravariante)

Mestrado em Engenharia Informática 2001/2002

Covariância e Contravariância

- Panel <: Element
- Panel = [link : ()void, owner : ()Container]
- Element = [owner : ()Container]
- Método da classe Window (Window <: Container)
 - add(**Panel** y) { y.link() }
- b:Container := new Container(), e:Element
 - b.add(e) é seguro,mas
- b:Container := new Window(), e:Element
 - b.add(e) não é seguro!

Mestrado em Engenharia Informática 2001/2002

Covariância e Contravariância

- Contravariância
 - $A <: B \Rightarrow [m:B, \dots] <: [m:A, \dots]$ (Sather, C++)
 - $X \subseteq Y \Rightarrow [m:A, X] <: [m:A, Y]$ (Java)
 - fácil de implementar
 - garante a segurança dos programas de forma modular
 - importante para a programação baseada em componentes
- Covariância (Eiffel, CLOS)
 - Permite representar certas situações de forma + natural
 - `Cow <: Animal == [eat(Food), ...]`
 - `Vegetables <: Food`
 - Impede verificação modular (Eiffel “não serve” para COP)

Mestrado em Engenharia Informática 2001/2002

Equivalência / Ordenação de Tipos

- Equivalência estrutural garante a segurança, mas
- *Personificação*
 - `Shape == [draw: ()void, move: ()void]`
 - `Cowboy == [draw: ()void, move: ()void, shoot: ()void]`
 - `Cowboy <: Shape` (usando ordenação estrutural)
- Um tipo interface é um tipo de dados abstracto?
 - ↑ Encapsulação da implementação
 - ↑ Independência da representação
 - ↓ Não impede a *personificação*

Mestrado em Engenharia Informática 2001/2002

Sistemas de Tipos OO: Personificação

- Solução: equivalência por nome. Alternativas:
- Os tipos são as classes (C++, Eiffel)
 - classes não relacionadas por derivação/herança são tipos incompatíveis, mesmo se respeitarem a mesma interface
 - Por outro lado, nem sempre subclasse \Rightarrow subtipo!
- As interfaces são nomeadas, o programador declara explicitamente $I < J$ (desde que $I <: J!$) (Sather, Java)
 - A declaração de uma classe indica a que tipo pertencem as suas instâncias
 - Separação da hierarquia de tipos (polimorfismo) da hierarquia de reutilização (herança)

Mestrado em Engenharia Informática 2001/2002

Sistemas de Tipos OO: Hierarquias

- Hierarquias
 - de tipos (tipo / subtipo)
 - de classes (classe / subclasse)
 - de módulos (relação de importação)
- Conceitos diferentes!
 - Subtipos: Substitutividade / Polimorfismo
 - Subclasses: Reutilização / Herança
 - Módulos: Cliente / Servidor

Mestrado em Engenharia Informática 2001/2002

Herança não é subtipificação

- Um tipo representa um conjunto de objectos com comportamento comum (não necessariamente da mesma classe)
 - Os tipos hierarquizam-se por uma relação de subtipificação eventualmente usando equivalência por nome nomeada (para atender ao problema da personificação)
- Uma classe é um componente que propões uma certa implementação para um certo tipo
 - As classes hierarquizam-se por uma relação de inclusão (como ocorre usualmente com módulos - mas adoptando “sobreposição” textual do código)

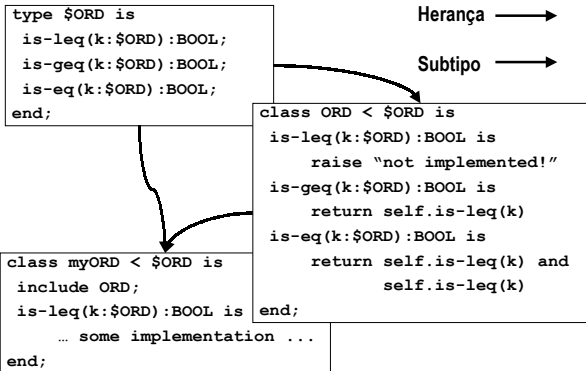
Mestrado em Engenharia Informática 2001/2002

Um caso: a linguagem Sather

- Uma (nova) interface pode ser declarada como subtipo de outra interface
 - `type $Window < $Container is ...`
- ou como subtipo da união de várias interfaces
 - `type $Elem < $Printable, $Ord is ...`
- Uma (nova) classe pode “incluir” várias subclasses
 - `class Window < $Window { ... }`
 - `class Pane < $Window, $Printable include Window { ... }`
 - `class Funny < $Window include Window, TickTac { ... }`
- A subclasse resultante não define um subtipo
- Herança múltipla com renomeação de atributos

Mestrado em Engenharia Informática 2001/2002

Em Sather classes não são tipos



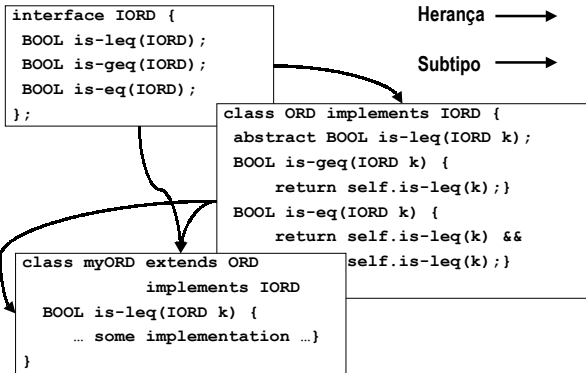
Mestrado em Engenharia Informática 2001/2002

O caso da linguagem Java

- Uma (nova) interface pode ser declarada como subtipo de outra interface
 - `interface IWindow extends IContainer { ... }`
- ou como subtipo da união de várias interfaces
 - `interface IElem extends IPrintable, IOrd { ... }`
- Uma (nova) classe pode ser declarada como subclasse de uma única classe (default = Object)
 - `class Window implements IWindow { ... }`
 - `class Pane extends Window implements Iwindow { ... }`
 - `class IElem extends IPrintable, IOrd { ... }`
- A subclasse resultante define um subtipo

Mestrado em Engenharia Informática 2001/2002

Em Java classes e interfaces são tipos



Mestrado em Engenharia Informática 2001/2002

(Sub)Tipos Comportamentais

- A noção de subtipo baseada em ordenação de assinaturas fornece garantias de correcção importantes, mas relativamente fracas:
 - os objectos do subtipo são capazes de responder a todas as mensagens implementadas pelos objectos do supertipo
 - Garantida a **substitutividade**: todas as operações invocadas estão definidas
 - Mas não necessariamente **bem** definidas.
- Princípio da substitutividade semântica: objectos do subtipo **comportam-se** como objectos do supertipo

Mestrado em Engenharia Informática 2001/2002

(Sub)Tipos Comportamentais

- ADT Bag
 - put: element \rightarrow void
 - get: \rightarrow element
- ADT Stack
 - put: element \rightarrow void
 - get: \rightarrow element
- Qualquer propriedade demonstrável para objectos de tipo T deve ser **também** demonstrável para objectos de um subtipo $S <: T$.

Mestrado em Engenharia Informática 2001/2002

(Sub)Tipos Comportamentais

- ADT Bag Bag <: Stack
 - put: element \rightarrow void
 - get: \rightarrow element
- ADT Stack Stack <: Bag
 - put: element \rightarrow void
 - get: \rightarrow element
- **S é subtipo de T** se qualquer propriedade $\mathcal{A}(x)$ que os objectos de tipo T satisfaçam for **também** satisfeita por qualquer objecto de tipo S.
 - $S <: T$ e $\forall x \in T \mathcal{A}(x) \Rightarrow \forall y \in S \mathcal{A}(y)$

Mestrado em Engenharia Informática 2001/2002

Tipo Comportamental (CADT)

- O nome do tipo T
- O conjunto de valores possíveis associado ao tipo
- Um conjunto OPS de operações sobre valores de tipo T (a “interface” do ADT). Para cada operação (método) define-se
 - O nome do método
 - A sua assinatura
 - O seu comportamento, em termos de pré-condições e pós-condições

Mestrado em Engenharia Informática 2001/2002

Pré- e Pós- condições

- Uma condição sobre o estado de um objecto pode ser descrito por uma fórmula lógica de primeira ordem:
 - Predicados sobre valores: $p(x,y)$, $x = y$, etc
 - Operadores lógicos: $\forall x.A$, $\exists x.A$, $\neg A$, $A \wedge B$, $A \vee B$, $A \Rightarrow B$, etc
- Para cada método (em geral, para uma operação):
 - A **pré-condição** caracteriza os estados nos quais a execução do método **pode** ter lugar (contracto)
 - A **pós-condição** caracteriza o estado resultante após a execução do método, assumindo que a pré-condição era válida no momento em que a execução foi iniciada

Mestrado em Engenharia Informática 2001/2002

O CADT Bag

- Type BAG is
 - domain [elems:multiset(E), bound:integer]
- put: $E \rightarrow \text{void}$
 - get: $\rightarrow E$
 - size: $\rightarrow \text{integer}$
 - equal: $\text{BAG} \times \text{BAG} \rightarrow \text{boolean}$

Mestrado em Engenharia Informática 2001/2002

O CADT Bag (Métodos)

- **type BAG is** for all $b:\text{BAG}$
 - **domain** [$\text{elems}:\text{multiset}(E)$, $\text{bound}:\text{integer}$]

– **put**: ($i:E$) void

- **requires** $\#(b_{\text{pre}}.\text{elems}) < b_{\text{pre}}.\text{bound}$
- **modifies** b
- **ensures** $b_{\text{post}}.\text{elems} = b_{\text{pre}}.\text{elems} \cup \{i\} \wedge$
 $b_{\text{post}}.\text{bound} = b_{\text{pre}}.\text{bound}$

Mestrado em Engenharia Informática 2001/2002

O CADT Bag (Métodos)

- **type BAG is** for all $b:\text{BAG}$
 - **domain** [$\text{elems}:\text{multiset}(E)$, $\text{bound}:\text{integer}$]

– **get**: () integer

- **requires** $\#(b_{\text{pre}}.\text{elems}) > 0$
- **modifies** b
- **ensures** $b_{\text{post}}.\text{elems} = b_{\text{pre}}.\text{elems} - \{\text{result}\} \wedge$
 $\text{result} \in b_{\text{pre}}.\text{elems} \wedge$
 $b_{\text{post}}.\text{bound} = b_{\text{pre}}.\text{bound}$

- Não-determinismo de especificação: a implementação pode escolher o elemento **result** como quiser

Mestrado em Engenharia Informática 2001/2002

O CADT Bag (Métodos)

- **type BAG is** for all $b:\text{BAG}$
 - **domain** [$\text{elems}:\text{multiset}(E)$, $\text{bound}:\text{integer}$]

– **size**: () integer

- **ensures** $\text{result} = \text{size}(b_{\text{pre}}.\text{elems}) \wedge$
 $b_{\text{post}}.\text{elems} = b_{\text{pre}}.\text{elems} \wedge$
 $b_{\text{post}}.\text{bound} = b_{\text{pre}}.\text{bound}$

- Nota: é necessário afirmar explicitamente que os valores não são alterados (na prática podem usar-se abreviaturas: etiquetar métodos como “observadores” ou “modificadores”).

Mestrado em Engenharia Informática 2001/2002

O CADT Bag (Métodos)

- **type BAG is** for all $b:\text{BAG}$
 - **domain** [elems:multiset(E), bound:integer]

– equal: (a:BAG; b:BAG) boolean

- ensures *result* = (a = b)

Mestrado em Engenharia Informática 2001/2002

O CADT Bag (Invariantes)

- Para além destes elementos, a especificação deve listar um conjunto de condições **invariantes**
- **invariante**: condição (que dever ser) sempre satisfeita durante todo o período de vida do objecto
- **type BAG is** for all $b:\text{BAG}$
 - **domain** [elems:multiset(E), bound:integer]

– **invariant** $\#(b.\text{elems}) \leq b.\text{bound}$

Mestrado em Engenharia Informática 2001/2002

O CADT Bag

- O estado inicial de cada objecto é definido através de operações especiais de criação, por exemplo:

– newbag(n:integer)BAG

- requires $n \geq 0$
- ensures new(*result*) and *result*_{post} = ($\{\}$,n)

– newsmallbag()BAG

- ensures new(*result*) and *result*_{post} = ($\{\}$,25)

- As operações de criação têm necessariamente que estabelecer os invariantes do tipo a que pertencem
- Os invariantes têm que ser mantidos pelos métodos

Mestrado em Engenharia Informática 2001/2002

O CADT Bag (Métodos)

- **type BAG is** for all $b:BAG$
 - **domain** [elems:multiset(E), bound:integer]

– **invariant** $\#(b.\text{elems}) \leq b.\text{bound}$
– **put**: $(i:E)$ void

- **requires** $\#(b_{\text{pre}}.\text{elems}) < b_{\text{pre}}.\text{bound}$
- **modifies** b
- **ensures** $b_{\text{post}}.\text{elems} = b_{\text{pre}}.\text{elems} \cup \{i\} \wedge$
 $b_{\text{post}}.\text{bound} = b_{\text{pre}}.\text{bound}$

- **Podemos verificar que**

– $\#(b_{\text{pre}}.\text{elems}) \leq b_{\text{pre}}.\text{bound} \Rightarrow \#(b_{\text{post}}.\text{elems}) \leq b_{\text{post}}.\text{bound}$

Mestrado em Engenharia Informática 2001/2002

O CADT Stack

- **Type STACK is**
 - **domain** [elems:multiset(E), bound:integer]

– **push**: $E \rightarrow \text{void}$
– **pop**: $\rightarrow E$
– **height**: $\rightarrow \text{integer}$
– **settop**: $E \rightarrow \text{void}$
– **equal**: $BAG \times BAG \rightarrow \text{boolean}$

Mestrado em Engenharia Informática 2001/2002

O CADT Stack

- **type STACK is** for all $s:STACK$
 - **domain** [items:list(E), limit:integer]

– **invariant** $\#(s.\text{items}) \leq s.\text{limit}$
– **push**: $(i:E)$ void

- **requires** $\#(s_{\text{pre}}.\text{items}) < s_{\text{pre}}.\text{limit}$
- **modifies** s
- **ensures** $s_{\text{post}}.\text{items} = \text{cons}(i, s_{\text{pre}}.\text{items}) \wedge$
 $s_{\text{post}}.\text{items} = s_{\text{pre}}.\text{limit}$

- **Podemos verificar:**

– $\#(s_{\text{pre}}.\text{items}) \leq s_{\text{pre}}.\text{limit} \Rightarrow \#(s_{\text{post}}.\text{items}) \leq s_{\text{post}}.\text{limit}$

Mestrado em Engenharia Informática 2001/2002

O CADT Stack

- **type STACK is** for all $s:\text{STACK}$
 - **domain** [items:list(E), limit:integer]

– **invariant** $\#(s.\text{items}) \leq s.\text{limit}$

– **pop: () integer**

- **requires** $\#(s_{\text{pre}}.\text{items}) > 0$
- **modifies** s
- **ensures** $s_{\text{post}}.\text{items} = \text{tail}(s_{\text{pre}}.\text{items}) \wedge$
result = $\text{head}(s_{\text{pre}}.\text{items}) \wedge$
 $s_{\text{post}}.\text{limit} = s_{\text{pre}}.\text{limit}$

Mestrado em Engenharia Informática 2001/2002

O CADT Stack (Métodos)

- **type STACK is** for all $s:\text{STACK}$
 - **domain** [items:list(E), limit:integer]

– **height: () integer**

- **ensures result** = $\text{size}(s_{\text{pre}}.\text{items}) \wedge$
 $s_{\text{post}}.\text{items} = s_{\text{pre}}.\text{limit} \quad \wedge$
 $s_{\text{post}}.\text{limit} = s_{\text{pre}}.\text{limit}$

Mestrado em Engenharia Informática 2001/2002

O CADT Stack (Métodos)

- **type STACK is** for all $s:\text{STACK}$
 - **domain** [items:list(E), limit:integer]

– **equal: (a:STACK; b:STACK) boolean**

- **ensures result** = $(a = b)$

Mestrado em Engenharia Informática 2001/2002

O CADT Stack (Métodos)

- **type STACK is** for all $s:STACK$

- **domain** [items:list(E), limit:integer]

- **settop**: (i: E) void

- **requires** $\#(s_{pre}.items) > 0$

- **modifies** s

- **ensures** $s_{post}.items = cons(i, tail(s_{pre}.items)) \wedge$
 $s_{post}.items = s_{pre}.limit$

Mestrado em Engenharia Informática 2001/2002

Ordenação de Tipos Comportamentais

- Na visão comportamental, quando é que um objecto de um subtipo pode ser seguramente substituído por um objecto de um supertipo?
- Quando qualquer propriedade assumida pelo cliente para as instâncias do supertipo é preservada pelas instâncias do subtipo
 - Se um cliente espera um objecto conforme a especificação BAG, pode usar um objecto conforme a especificação STACK, mas não o contrário (contraexemplo?):
- A relação de subtipo exprime refinamento semântico

Mestrado em Engenharia Informática 2001/2002

“Ingredientes” da relação de subtipo comportamental

- Função de abstracção \mathcal{A} converte a representação do subtipo na representação do supertipo:
- Nos exemplos STACK e BAG, temos
 - STACK: **domain** [elems:multiset(E), bound:integer]
 - BAG: **domain** [items:list(E), limit:integer]
- A função \mathcal{A} é definida simplesmente por:
 $\mathcal{A}(\langle list, n \rangle) = \langle elements(list), n \rangle$
(elements(\mathcal{U}) denota o multiconjunto dos elementos da lista \mathcal{U})

Mestrado em Engenharia Informática 2001/2002

“Ingredientes” da relação de subtipo comportamental

- Função de renomeação: função \mathcal{R} que traduz os nomes dos métodos do subtipo nos do supertipo:
- Nos exemplos STACK e BAG, a função \mathcal{R} é definida simplesmente por:

$\mathcal{R}(\text{push}) = \text{put}$
 $\mathcal{R}(\text{pop}) = \text{get}$
 $\mathcal{R}(\text{height}) = \text{size}$
 $\mathcal{R}(\text{equal}) = \text{equal}$

Ordenação de Tipos Comportamentais

- O tipo $\sigma = (S, M)$ diz-se subtipo comportamental do tipo $\tau = (T, N)$ quando:
- Preservação dos invariantes:
 $\forall s \in S \text{ inv}_{\sigma}(s) \Rightarrow \text{inv}_{\tau}(\mathcal{A}(s))$
- Compatibilidade básica dos métodos:
 - Contravariância nos tipos dos argumentos
 - Covariância no tipo do resultado

Ordenação de Tipos Comportamentais

- O tipo $\sigma = (S, M)$ diz-se um **subtipo comportamental** do tipo $\tau = (T, N)$ quando:
- Preservação dos invariantes.
- Compatibilidade básica dos métodos.
- Compatibilidade semântica dos métodos:
Pré- e pós- condições:
 $\forall x \in \sigma \text{ meth}_{\tau}.\text{pre}[x_{\text{pre}} / \mathcal{A}(x_{\text{pre}})] \Rightarrow \text{meth}_{\sigma}.\text{pre}$
 $\forall x \in \sigma \text{ meth}_{\sigma}.\text{post} \Rightarrow \text{meth}_{\tau}.\text{post}[x_{\text{pre}} / \mathcal{A}(x_{\text{pre}}), x_{\text{post}} / \mathcal{A}(x_{\text{post}})]$
(supondo que meth_{τ} é a versão renomeada (por \mathcal{R}) de meth_{σ})

Ordenação de Tipos Comportamentais

- No caso dos tipos STACK e BAG, relativamente aos métodos **put** e **push**, tem que se verificar:

- $\forall s \in S$
 $s_{\text{post}}.\text{items} = \text{cons}(i, s_{\text{pre}}.\text{items}) \wedge s_{\text{post}}.\text{items} = s_{\text{pre}}.\text{limit} \Rightarrow$
 $\mathcal{A}(s_{\text{post}}).\text{elems} = \mathcal{A}(s_{\text{pre}}).\text{elems} \cup \{i\} \wedge$
 $\mathcal{A}(s_{\text{post}}).\text{bound} = \mathcal{A}(s_{\text{pre}}).\text{bound}$
- lembre: pós-condição de **push** em STACK
- $s_{\text{post}}.\text{items} = \text{cons}(i, s_{\text{pre}}.\text{items}) \wedge s_{\text{post}}.\text{items} = s_{\text{pre}}.\text{limit}$
- lembre: pós-condição de **put** em BAG
- $b_{\text{post}}.\text{elems} = b_{\text{pre}}.\text{elems} \cup \{i\} \wedge b_{\text{post}}.\text{bound} = b_{\text{pre}}.\text{bound}$

Mestrado em Engenharia Informática 2001/2002

Subtipos Comportamentais

- Esta definição de subtipo comportamental apresentada (Liskov-Wing, 94) garante: se **S é subtipo de T** então:
- Qualquer propriedade $\mathcal{A}(x)$ que os objectos de tipo T satisfaçam for **também** satisfeita por qualquer objecto de tipo S.

$$S <: T \wedge \forall x \in T \mathcal{A}(x) \Rightarrow \forall y \in S \mathcal{A}(y)$$

- No caso ilustrado, temos
 $\forall x \in \text{BAG } \mathcal{A}(x) \Rightarrow \forall y \in \text{STACK } \mathcal{A}(y)$
- Assim, qualquer implementação de STACK preserva a semântica de qualquer implementação de BAG

Mestrado em Engenharia Informática 2001/2002
