Engenharia de Software
Conceitos Básicos de Verificação Formal

Fernando Vanini

Klais Soluções

IC - UNICAMP

www.klais.com.br
+55 19 3249-2222

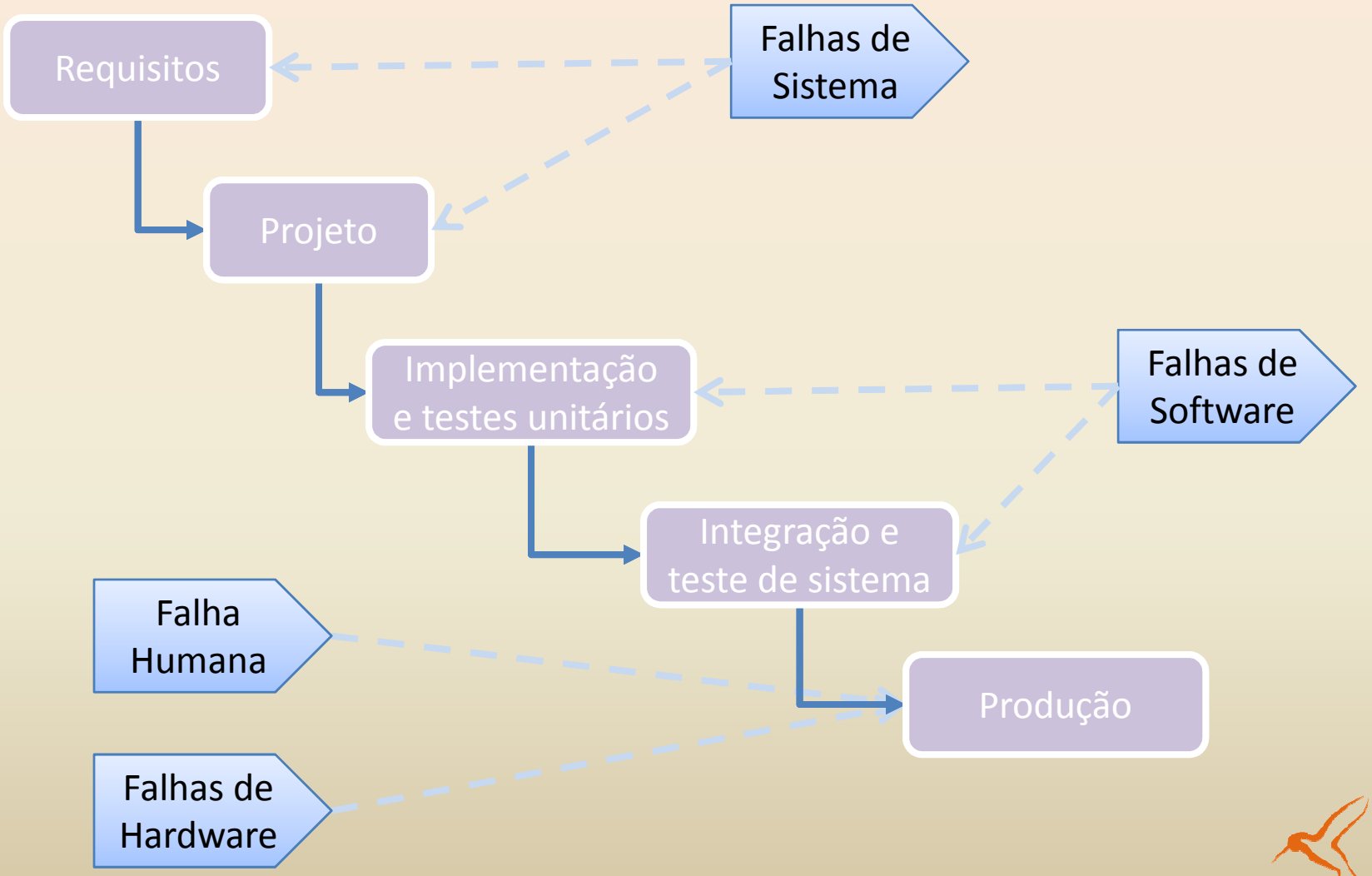
Sistemas Críticos

- Sistemas em que qualquer falha pode resultar em sérias perdas
  - Humana
  - Econômica
  - ...

Falhas

- Sistema
- Hardware
- Software
- Humana

Falhas e o Processo



Klais

Falhas e o ambiente

Klais

Falhas e Linguagens de Programação

- Algumas linguagens de programação foram projetadas no sentido de reduzir a ocorrência de falhas de programação
  - Rigidez de tipos
  - Tratamento de exceções
  - Verificações em tempo de compilação
- Exemplos
  - Pascal, Ada, Euclid, Chill
- Linguagens modernas como Java e C# adotam essas políticas.

Falhas e Testes

Testes Experimentais

- Imprescindíveis
- Principal ferramenta para a localização e correção de falhas.
- Técnica conhecida e madura.
- Suportada por ferramentas.
- No entanto

> *"Testes experimentais podem, quando muito, mostrar a presença, mas nunca provar a inexistência de erros."*
>
> Niklaus Wirth (Programação Sistemática)

Klais

Falhas e Testes

- Testes exaustivos
  - garantem a ausência de falhas
  - são impraticáveis

  Exemplo:

  Para testar de forma exaustiva uma função que tem como parâmetro dois inteiros de 64 bits, o número de casos possíveis é 2^{128}.

Klais

Métodos Formais

- Objetivo
  - Garantir a ausência de falhas durante o processo de desenvolvimento de software
- As técnicas
  - Verificação dos requisitos
  - Verificação dos modelos
  - Verificação dos programas
  - Transformações formais

Métodos Formais

- Técnicas e ferramentas
  - VDL / VDM
  - Z-notation
  - B-method
  - UML/OCL

Verificação Formal de Programas

Idéia geral:

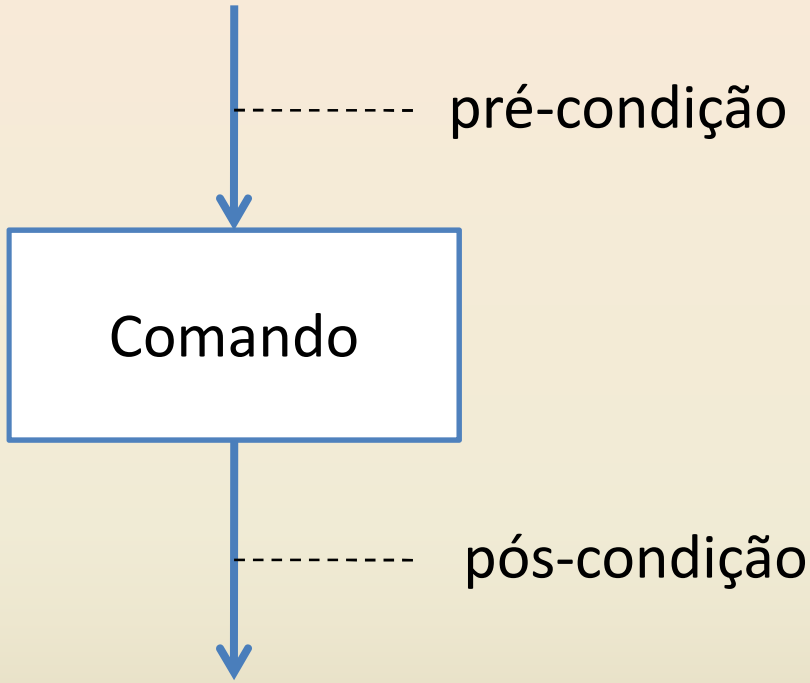
uso de técnicas baseadas em lógica matemática para demonstrar formalmente que um programa implementa corretamente as funcionalidades para as quais foi projetado.

Verificação de Programas

Idéia inicial

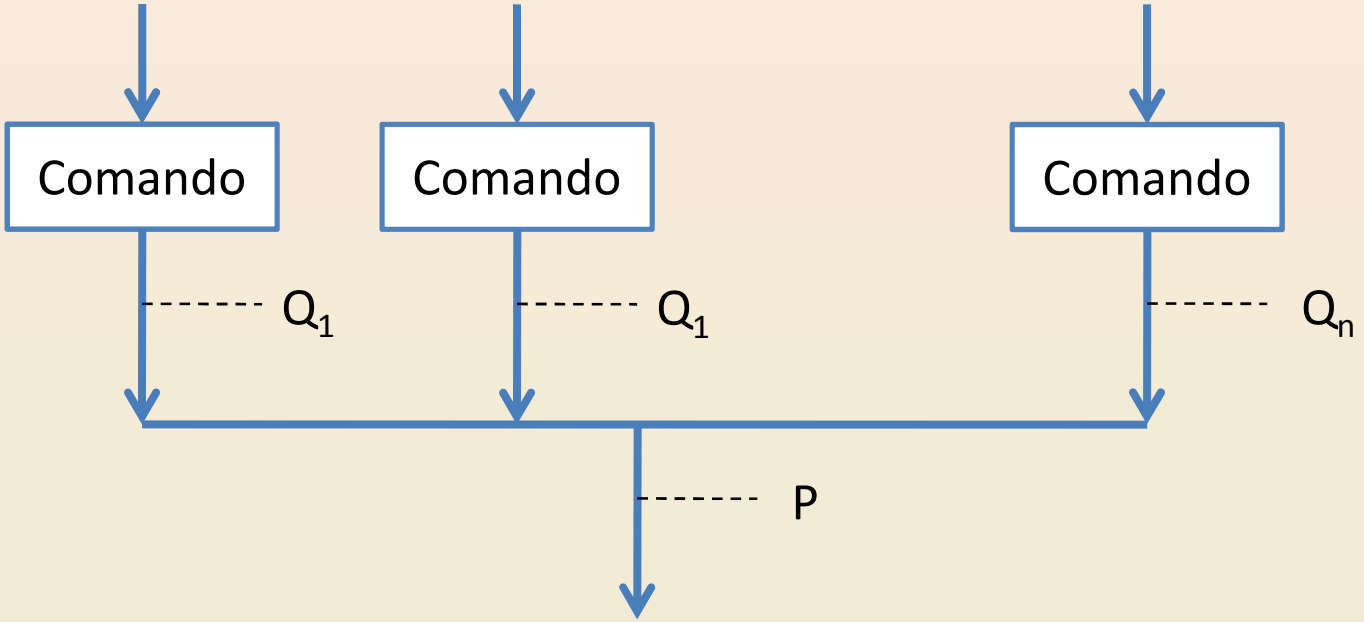
- Estabelecer para cada comando do programa suas pré-condições.
- Determinar as pós-condições e verificar sua validade.

Verificação de Programas



Klais

Verificação de Programas



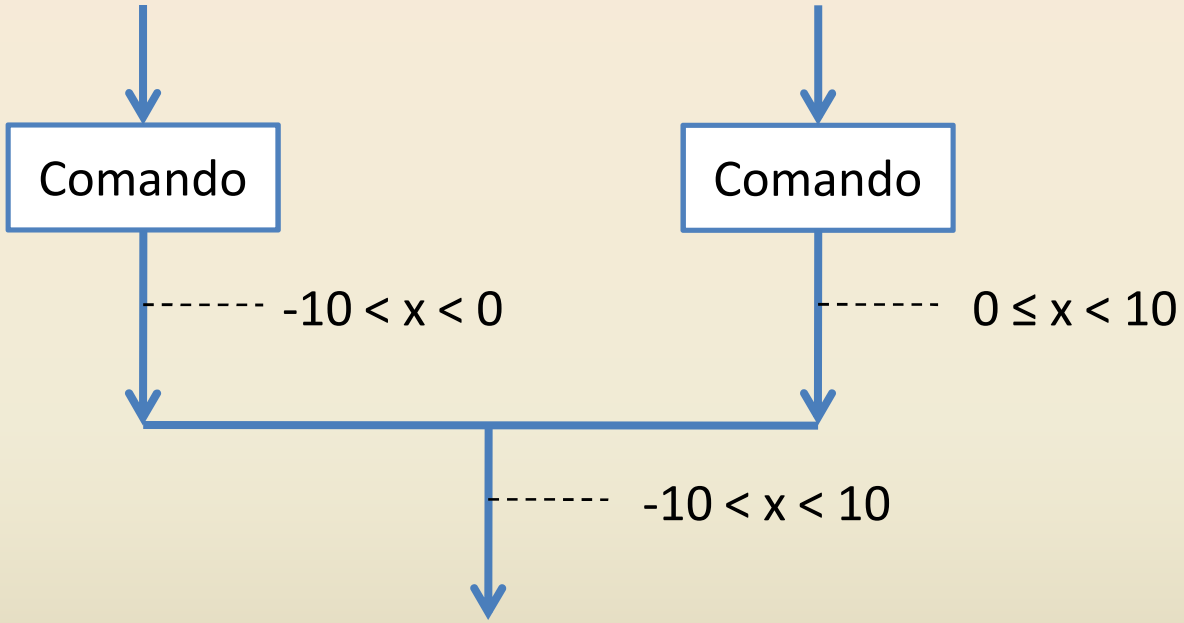
Q_i => P para i = 1,2...n

(para que P seja verdadeiro, Q_i =>P para todo i)

Klais

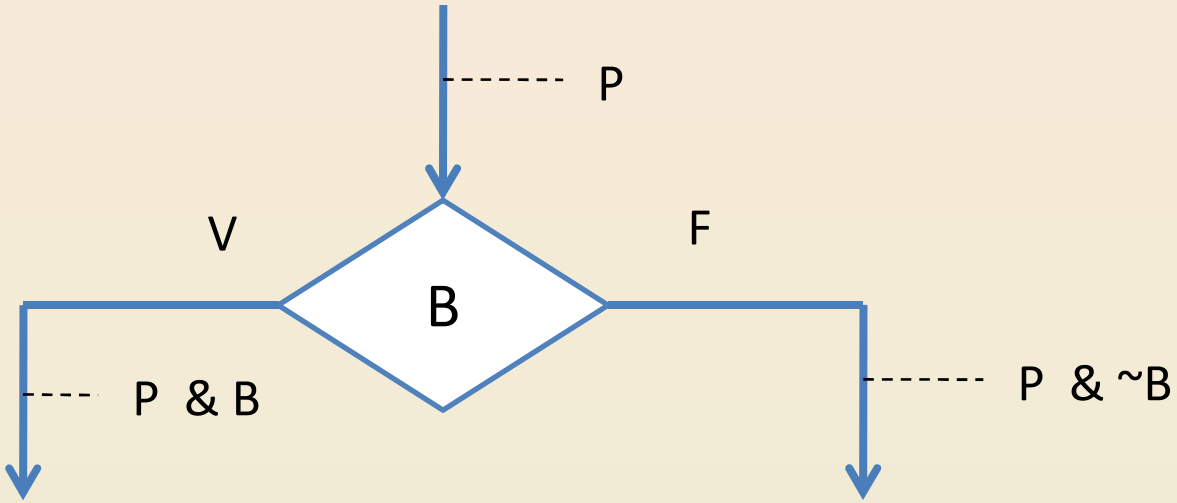
Verificação de Programas

Um exemplo



Klais

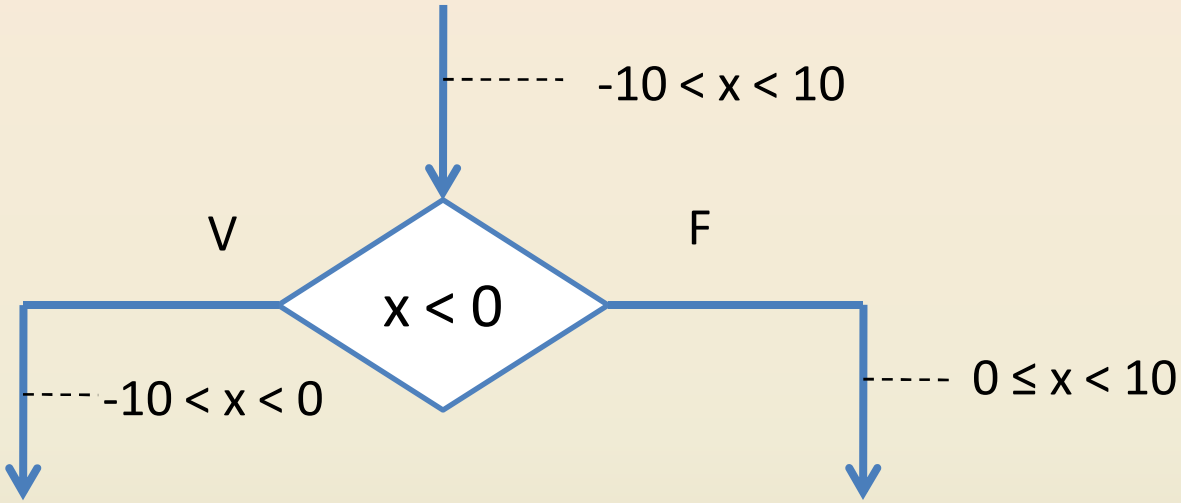
Verificação de Programas



Klais

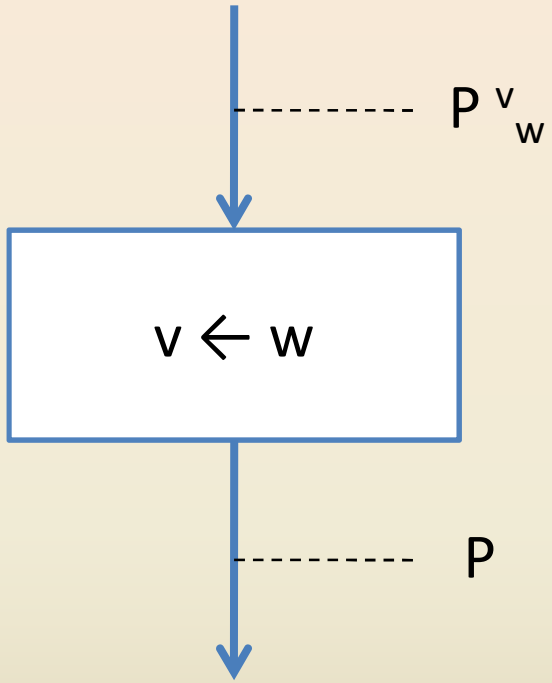
Verificação de Programas

Um exemplo



Klais

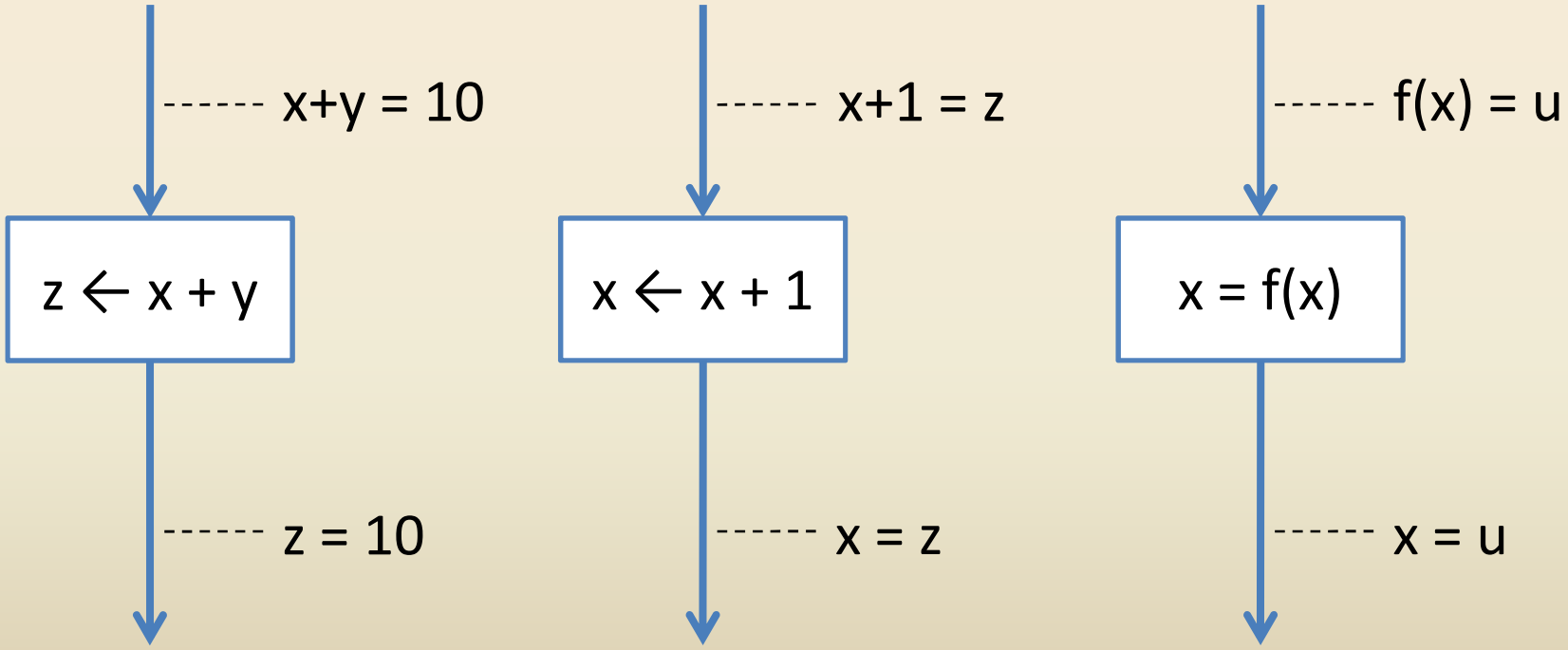
Verificação de Programas



Klais

Verificação de Programas

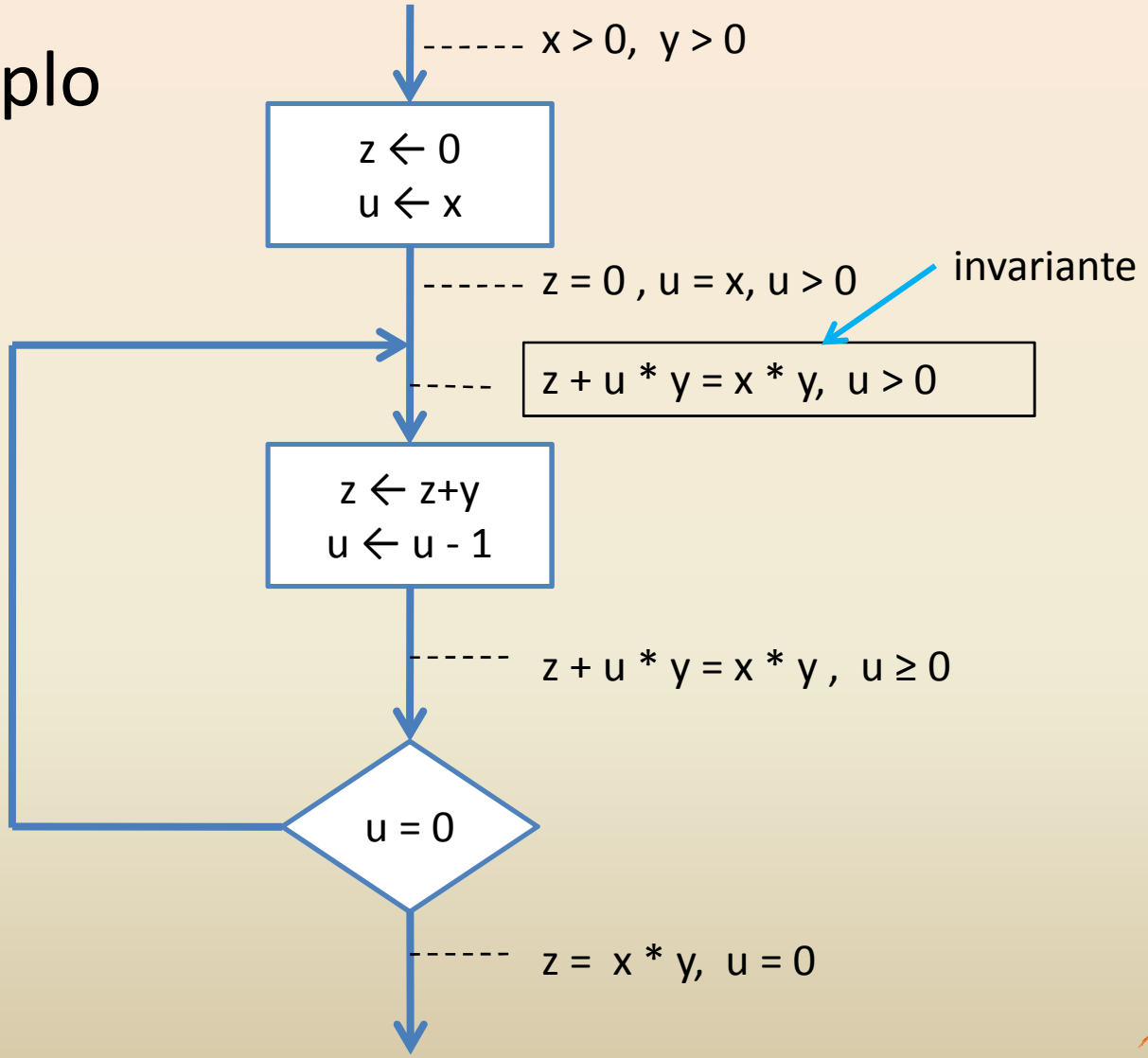
- Exemplos



Klais

Verificação de Programas

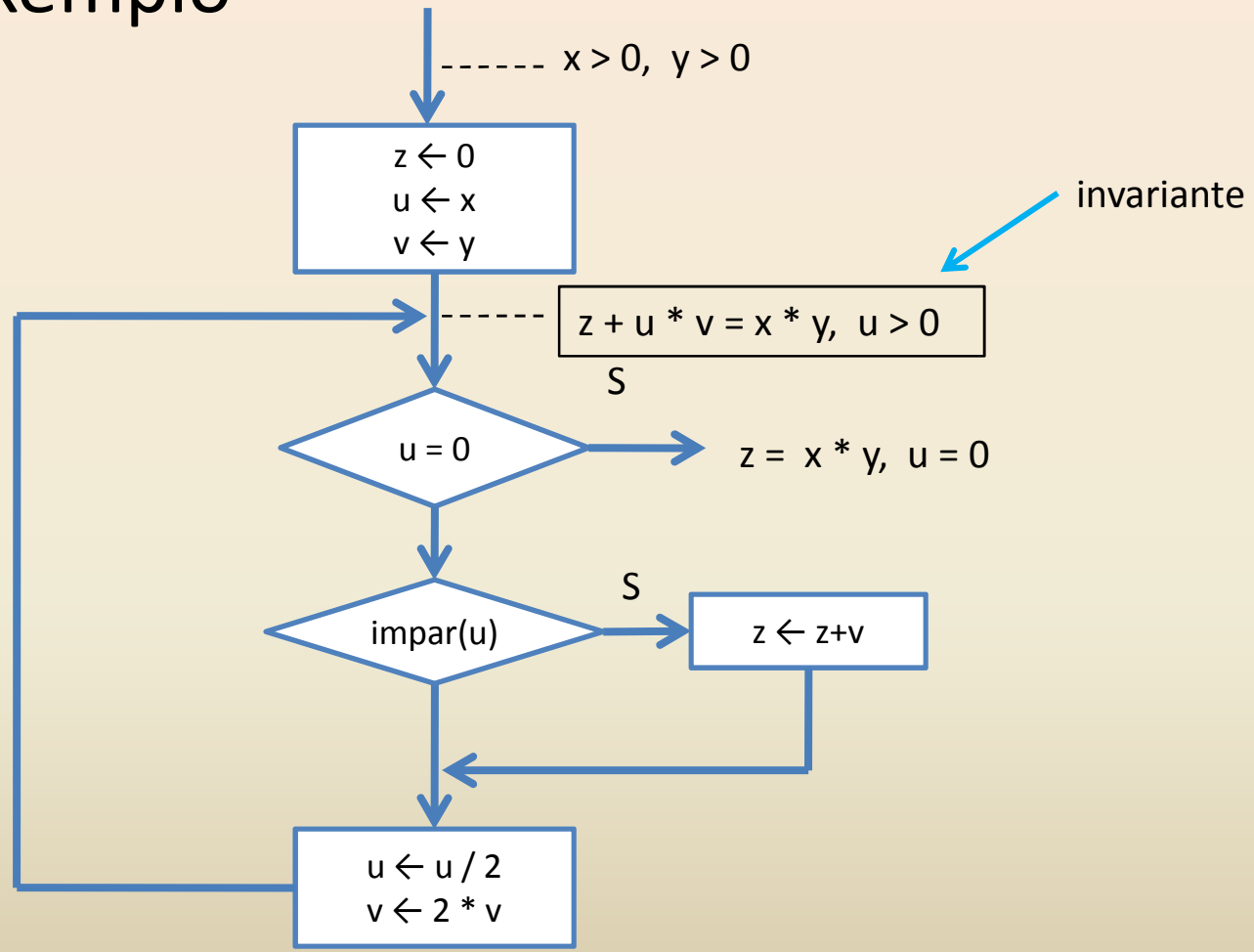
- Um exemplo



Klais

Verificação de Programas

- Outro exemplo



Klais

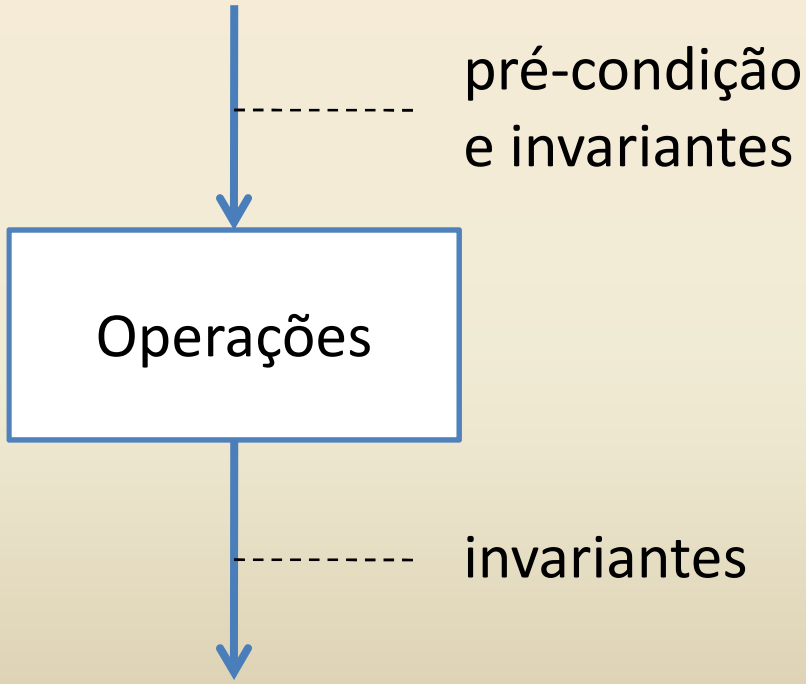
Verificação de Programas

- Na prática
  - Aplicável a pequenos programas
  - Baseada na semântica de uma ‘máquina ideal’
  - É difícil determinar as invariantes

Os conceitos, no entanto são a base dos métodos formais atualmente em uso.

Verificação Formal

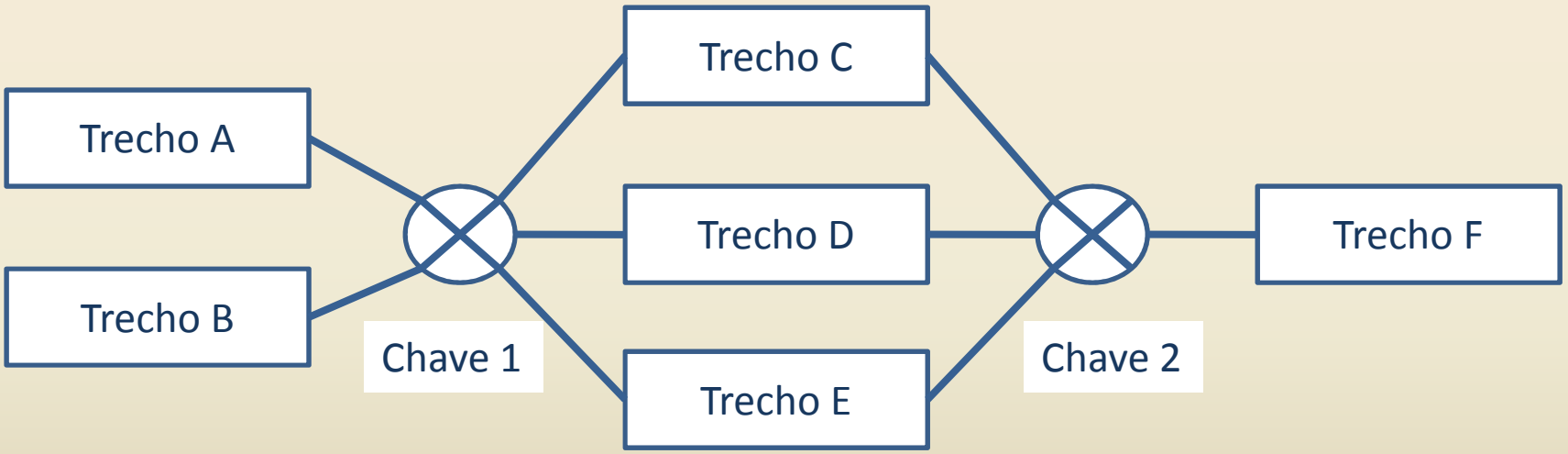
- Idéia Geral



Klais

Um exemplo

- Sistema de Controle Ferroviário



Klais

Um exemplo

- Sistema de Controle de Ferroviário as invariantes poderiam ser
  - Para qualquer par de composições no mesmo trecho de ferrovia, a distância mínima deve ser menor que 2 km.
  - Apenas um trem ocupa uma chave por vez.
  - Se uma chave está mudando de estado, o tempo de chegada de um trem à mesma deve ser maior que o tempo necessário à conclusão da mudança.
  - Se uma chave está ocupada, o tempo de chegada de um trem à mesma deve ser maior que o tempo necessário à liberação da mesma mais o tempo de mudança de estado.

Klais

O Modelo Formal Z

- Criado em Oxford, final da década de ‘70
- Baseado em teoria dos conjuntos e lógica de predicados
  - Teoria dos conjuntos: representa as informações e operações sobre as mesmas
  - Lógica dos predicados: representa as propriedades do sistema.
- Linguagem de especificação
  - Sintaxe e semântica matematicamente precisas
  - Esquemas: permitem a divisão em módulos e refinamentos sucessivos.
- Várias ferramentas baseadas na notação Z

Klais

Z - Lógica de Predicados

- Predicado:
  - Propriedades do sistema
  - Restrições sobre objetos
  - Construídos a partir de predicados mais simples, combinados através de operadores lógicos.
  - Exemplo: x > 10
- Proposições podem ser avaliadas a partir dos predicados
- Quantificadores: expressam propriedades dos objetos
  - Existenciais:
  - Universais

Z - Lógica de Predicados

- Exemplos de predicados:

$$x > 5$$

$$(y \geq 0 \land x > y) \Rightarrow x > 0$$

$$\exists\, x : \mathbb{N} \bullet x > 5$$

$$\forall\, x : \mathbb{N} \bullet x > 5$$

Klais

Z - Lógica de Predicados

- Proposição: forma geral

$$Q\ x{:}\ a \mid p \cdot q$$

  - Q é um quantificador
  - x: a é uma variável do tipo a
  - p é um predicado que define uma restrição sobre x
  - q é uma propriedade (esperada) da variável

  Uma proposição pode ter mais de uma variável.

  Exemplos:

$$\forall\ x{:}\ a \mid p \cdot q$$
$$\exists\ x{:}\ a \mid p \cdot q$$

Klais

Z – Teoria dos Conjuntos

- Conjuntos modelam as informações
  - Coleção bem definida de objetos de mesmo tipo
  - Exemplos de proposições (todas verdadeiras)

$$a == \{1, 2, 3, 4\}$$
$$b == \{2, 4\}$$
$$\forall x : \mathbb{N} \bullet x \in b \Rightarrow x \in a$$
$$\exists x : \mathbb{N} \mid x \in a \bullet x \notin b$$
$$a \cap b \neq \varnothing$$
$$b \subset a$$
$$a \setminus b = \{1, 3\}$$

Klais

Z – Teoria dos Conjuntos

- Produto Cartesiano

$$a \times b = \{(1, 2), (1, 4), (2, 2), (2, 4), (3, 2), (3, 4), (4, 2), (4, 4)\}$$

- Conjunto Potência

$$\mathbb{P}\, b = \{\varnothing, \{2\}, \{4\}, \{2, 4\}\}$$
$$\{2\} \in \mathbb{P}\, b$$
$$\{(1, 2), (1, 4)\} \in \mathbb{P}\, a \times b$$

Klais

O Modelo Formal Z

- Tipos:
  - define o conjunto de valores possíveis para variáveis desse tipo.
  - Tipos básicos: inteiros (Z) e naturais (N)
  - Novos tipos podem ser construídos usando os tipos já definidos.
  - Exemplos:

$$[Bebidas]$$

$$Valores == \mathbb{N}$$

$$Mensagens ::= VendaEfetuada \mid BebidaNaoDisponivel \mid DinheiroInsuficiente \mid TrocoNaoDisponivel$$

$$b : Bebidas$$

$$BebidasDisponiveis : \mathbb{P} Bebidas$$

$$DinheiroDisponivel : Valores$$

Klais

O Modelo Formal Z

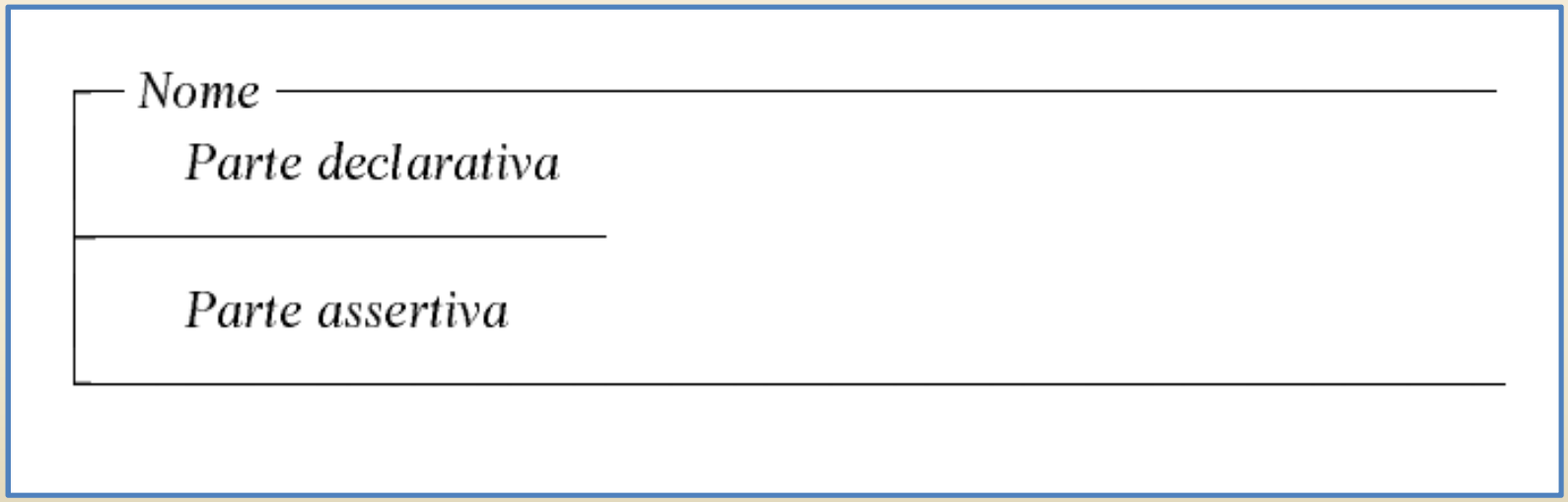
- Relações e Funções:
  - Relação: estabelece ligação entre pares de objetos
  - Operador dom: domínio da relação
  - Exemplo

$$PrecoBebida : Bebidas \pfun Valores$$
$$\forall Beb : Bebidas \mid Beb \in \mathrm{dom}\ PrecoBebida \bullet PrecoBebida(Beb) \geq 0$$

Klais

O Modelo Formal Z

- Esquemas:
  - Combinam as descrições de forma estruturada
  - Agrupam partes da especificação
  - Nomeação e reuso



Klais

O Modelo Formal Z

- Esquemas declarativo:
  - Descreve as variáveis que representam o estado do sistema
  - Exemplo

MaquinaBebidas

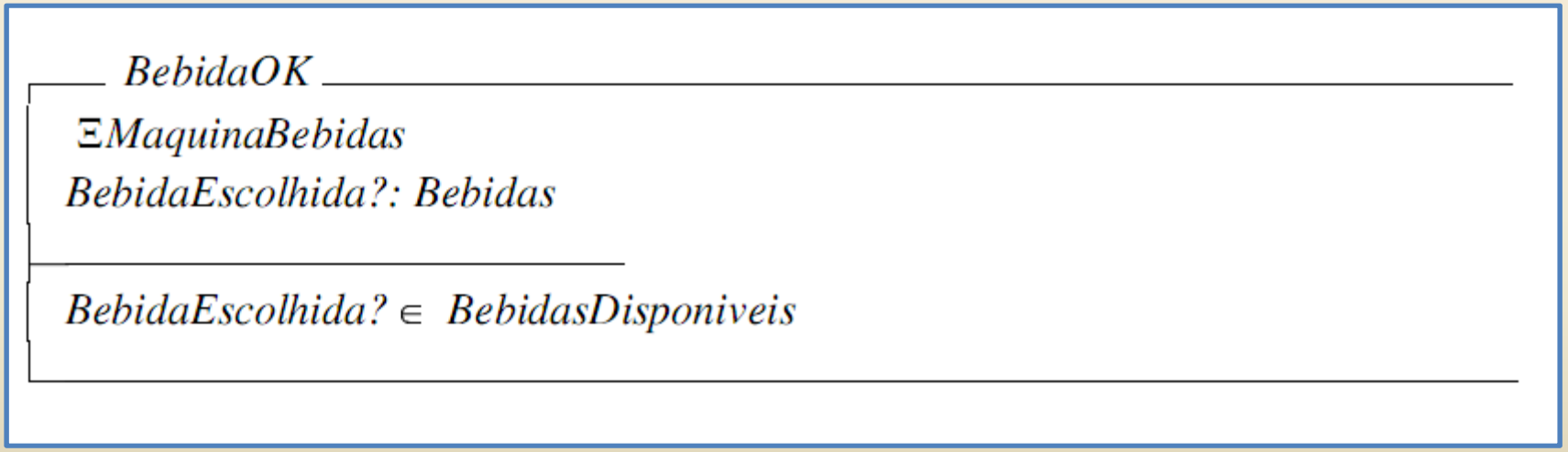
$$BebidasDisponiveis: \mathbb{P} Bebidas$$
$$DinheiroDisponivel: Valores$$
$$PrecoBebida: Bebidas \pfun Valores$$
$$NBebidasVendidas: Bebidas \pfun \mathbb{N}$$

$$DinheiroDisponivel \geq 0$$
$$\forall Beb: Bebidas \mid Beb \in \operatorname{dom} PrecoBebida \bullet PrecoBebida(Beb) \geq 0$$
$$\forall Beb: Bebidas \mid Beb \in \operatorname{dom} PrecoBebida \land Beb \in BebidasDisponiveis \bullet PrecoBebida(Beb) > 0$$

Klais

O Modelo Formal Z

- Variáveis de Entrada e saída:
  - "?" ou "!" são usadas como sufixo no nome para indicar entrada ou saída.
  - Exemplo



Klais

O Modelo Formal Z

- Esquemas e Operações:
  - Operações descrevem mudanças de estado do sistema
  - Parte declarativa: referência ao esquema declarativo
  - Parte assertiva: predicados representam pré e pós-condições das operações especificadas
  - Pode causar (Ξ) ou não (Δ) mudança de estado no esquema declarativo referenciado

Klais

O Modelo Formal Z

DinheiroOK

$\Xi MaquinaBebidas$

$BebidaEscolhida?: Bebidas$

$DinheiroFornecido?: Valores$

$BebidaEscolhida? \in \text{dom } PrecoBebida$

$DinheiroFornecido? \geq PrecoBebida(BebidaEscolhida?)$

DinheiroDisponivelOK

$\Delta MaquinaBebidas$

$BebidaEscolhida?: Bebidas$

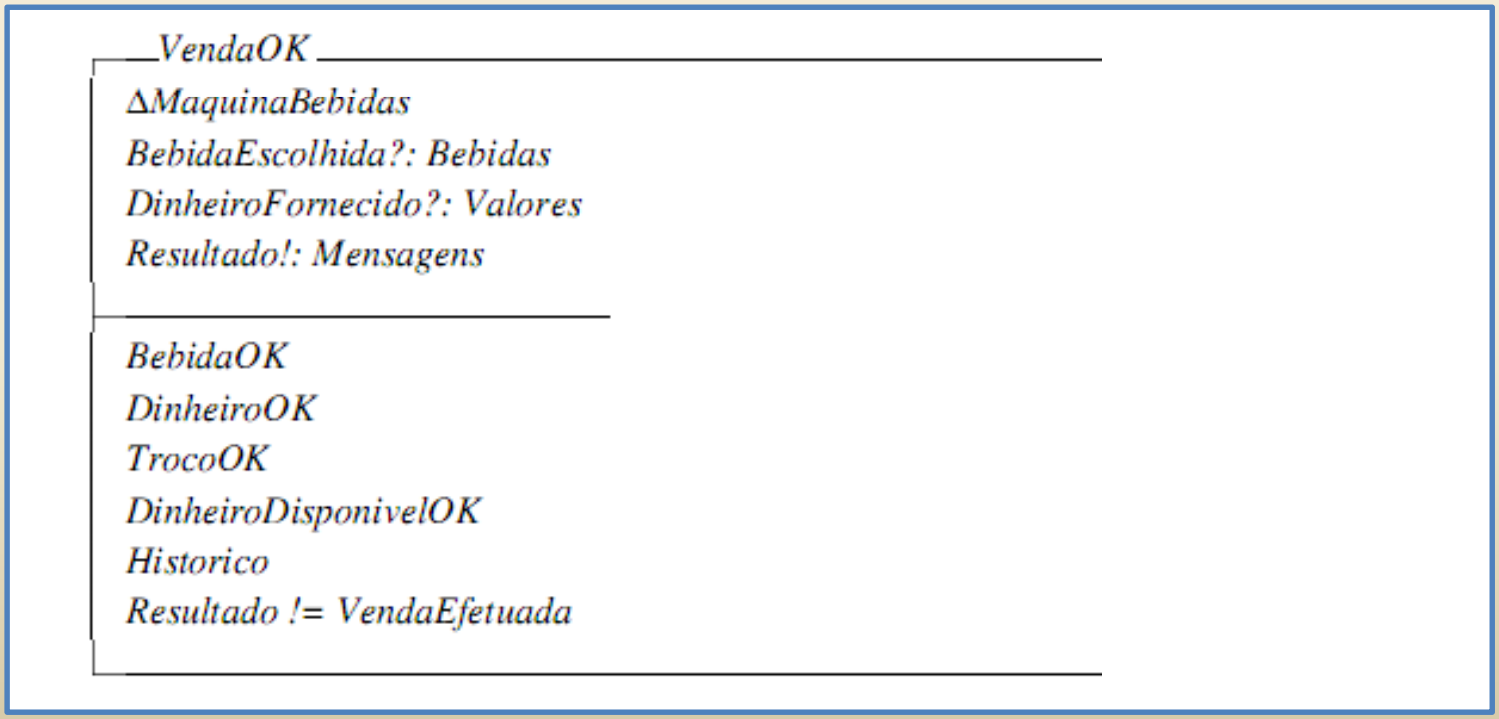
$BebidaEscolhida? \in \text{dom } PrecoBebida$

$DinheiroDisponivel' = DinheiroDisponivel + PrecoBebida(BebidaEscolhida?)$

Klais

O Modelo Formal Z

- Esquemas em predicados:
  - Um esquema pode ser usado como um predicado em outro esquema.
  - Nesse caso, os esquemas representam restrições às variáveis.
  - Um exemplo



Klais

O Modelo Formal Z

- Esquemas como composição lógica
  - Um esquema pode ser definido como uma composição lógica de outros esquemas.
  - Um exemplo:

$$Venda \mathrel{\widehat=} VendaOK \lor VendaNaoOK$$

Klais

O Modelo Formal Z

- Validação da Especificação
  - Sintaxe e semântica precisas => é possível verificar se estão corretas quanto à sintaxe e semântica.
  - Domínios podem ser verificados
  - Verificação da Consistência
    - Verificação do estado inicial com base num teorema apropriado
    - Validação de cada uma das operações, através de teoremas
    - Validação dos refinamentos

Klais

O Modelo Formal Z

- Verificação do estado inicial
  - Esquema p/ iniciação das variáveis
  - Teorema estabelecendo que existe um estado que atenda à invariante de estado e à operação de iniciação.
  - Um exemplo:

$$\exists\, MaquinaBebidas' \bullet\ MaquinaBebidasInit$$

Klais

O Modelo Formal Z

- Validação das Operações
  - Para cada operação o especificador deve definir um teorema apropriado relacionando a operação às propriedades esperadas após a sua realização.
  - Um exemplo:

$$MaquinaBebidas \wedge DinheiroDisponivelOK \Rightarrow MaquinaBebidas'$$

Klais

O Modelo Formal Z

- Especificação rigorosa, notação matemática.
- Usa um provador automático de teoremas.
- Demanda formação específica.
- Foco na validação da especificação.

Klais

O Método B

- Usa os conceitos de Z
- Desenvolvido por Jean-Raymond Abrial, 1985
- Baseado em
  - Máquina abstrata
  - Substituições generalizadas
- Permite a síntese do código executável a partir da especificação

O Método B

- A máquina abstrata
  - Máquina de estados
  - Modela os requisitos funcionais
  - Estimula o encapsulamento

O Método B

- Especificações
  - não são executáveis
  - a máquina modela os requisitos funcionais
- Implementação
  - refina a máquina
  - mantém invariantes
  - semelhante a programação (variáveis, tipos, mudança de estado)

Klais

O Método B

- Não é orientado à ‘camada de aplicações’
  - Os requisitos são imprecisos
- Voltado a camada de componentes ou API
  - Requisitos mais estáveis e formalizáveis

Klais

O Método B – um exemplo

```
MACHINE
  RMan(RES)
INVARIANTS
  rfree ⊆ RES
INITIALISATION
  rfree := ∅
OPERATIONS
  alloc(rr) =
    PRE
      rr ∈ rfree
    THEN
      rfree := rfree − {rr}
    END
  free(rr) =
    PRE
      rr ∈ RES ∧ rr ∉ rfree
    THEN
      rfree := rfree ∪ {rr}
    END
END
```

Rman é o nome da máquina (sistema)

RES é um conjunto

Invariantes p/ a máquina

alloc() e free() são as operações realizadas pela máquina

pré-condições

Klais

O Método B – operações

OPERATIONS

$alloc(rr) =$

PRE

$rr \in rfree$

THEN

$rfree := rfree - \{rr\}$

END

pré-condição: deve-se garantir que é verdadeira antes de realizar a operação.

then-part: efeito da operação se a pré-condição for verdadeira.

Klais

O Método B

- Obrigação de prova
  - A máquina é baseada em construções matemáticas
  - Prova de consistência
    1. Inicialização preserva as invariantes
    2. Cada uma das operações preserva as invariantes

O Método B

1. A inicialização preserva invariantes

$$[rfree := \emptyset]\, rfree \subseteq RES$$

- Forma Geral

  [G] I

  - G é o pseudo-programa
  - I é a invariante
  - substituição [x := E] P

$$\emptyset \subseteq RES$$

Klais

O Método B

2. As operações preservam as invariantes (alloc)

$$rfree \subseteq RES \wedge rr \in rfree \Rightarrow [rfree := rfree - \{rr\}]rfree \subseteq RES$$

- Forma Geral

  I ∧ P => [G] I

  - I é a invariante, P é a pré-condição e G é o pseudo-programa

$$rfree \subseteq RES \wedge rr \in rfree \Rightarrow rfree - \{rr\} \subseteq RES$$

Klais

O Método B

2. As operações preservam as invariantes (free)

$$rfree \subseteq RES \wedge rr \in rfree \Rightarrow$$
$$[\mathtt{PRE}\ \ rr \in rfree\ \ \mathtt{THEN}\ \ rfree := rfree - \{rr\}\ \ \mathtt{END}]\, rfree \subseteq RES$$

- Forma Geral

$$[\mathtt{PRE}\ \ P\ \ \mathtt{THEN}\ \ H\ \ \mathtt{END}]\, I \Leftrightarrow P \wedge [H]I$$

$$rfree \subseteq RES \wedge rr \in rfree \Rightarrow$$
$$rr \in rfree \wedge [rfree := rfree - \{rr\}]\, rfree \subseteq RES$$

$$rfree \subseteq RES \wedge rr \in rfree \Rightarrow [rfree := rfree - \{rr\}]\, rfree \subseteq RES$$

Klais

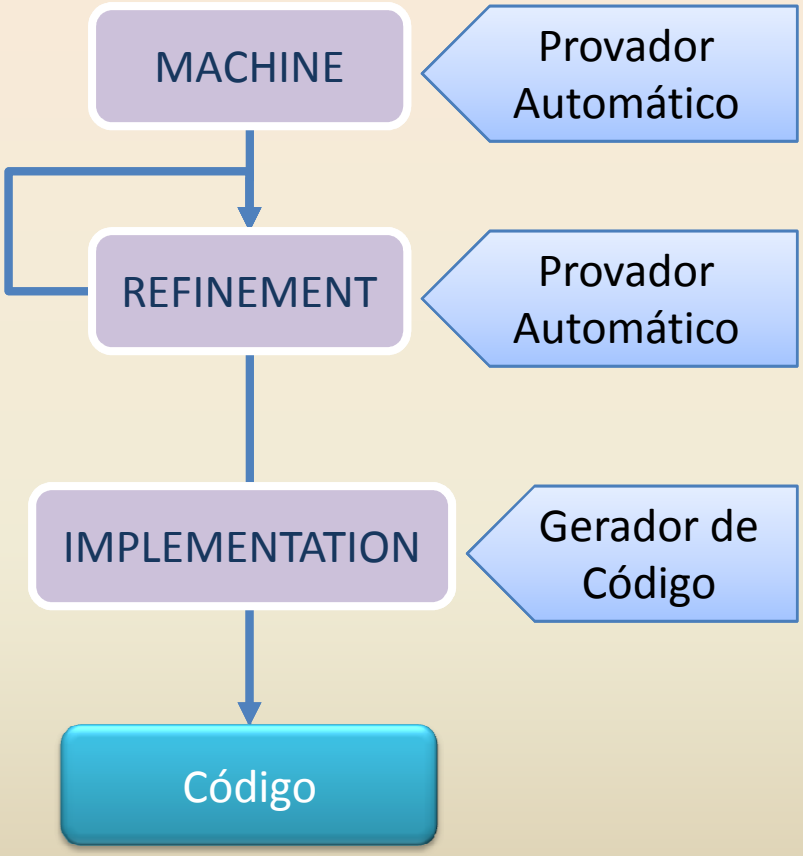
O Método B – um exemplo

```
MACHINE            Square
OPERATIONS
        sqr <-- Sqr(num) =
            PRE num : NAT
            THEN sqr := num*num
            END;

  sqrt <-- ApproxSqrt(num) =
            PRE num : NAT
            THEN
                ANY approx
                WHERE approx : NAT &
                      approx*approx <= num &
                      num < (approx+1)*(approx+1)
                THEN sqrt := approx
                END
            END
END
```

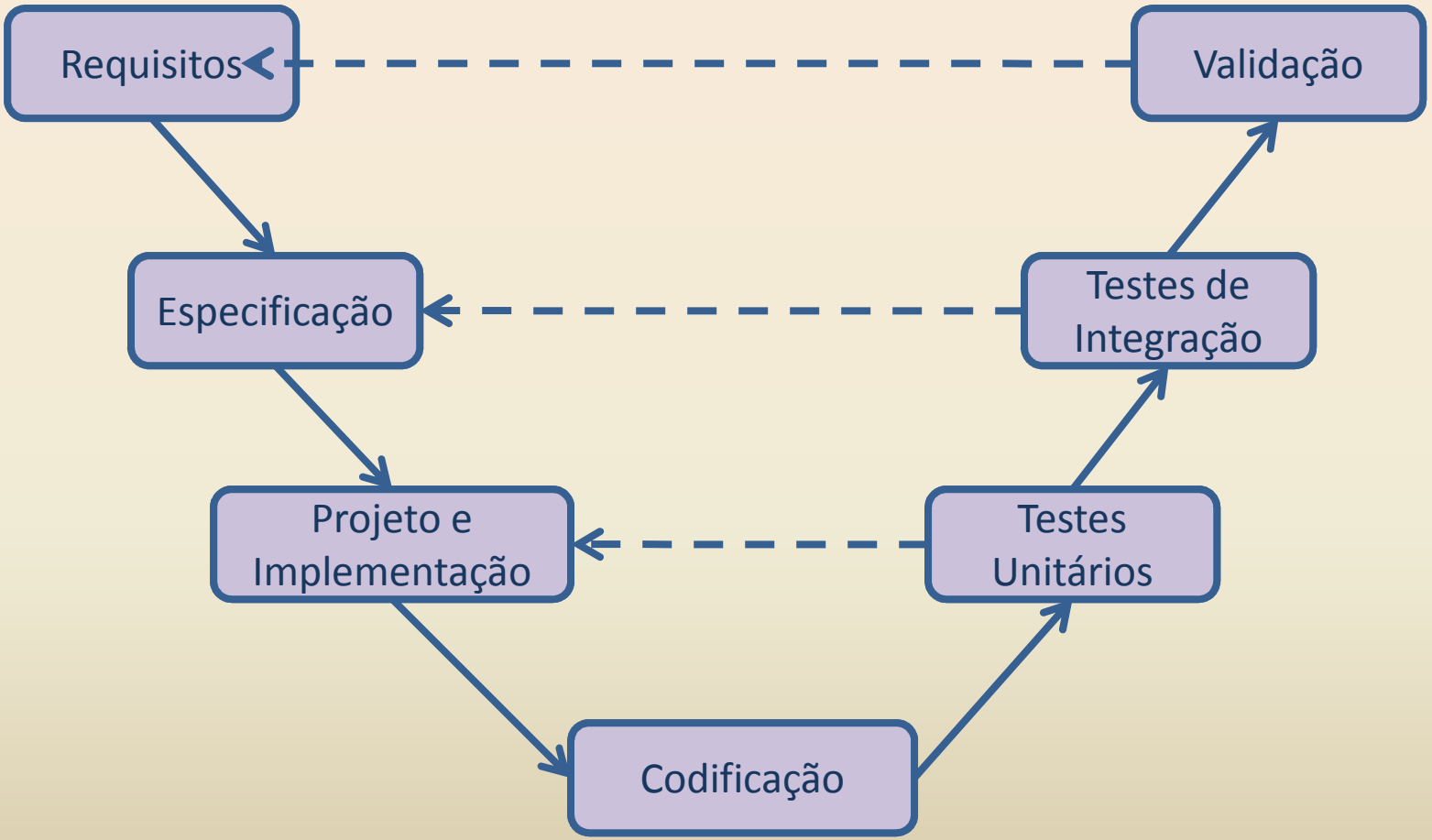
Klais

O Método B em uso



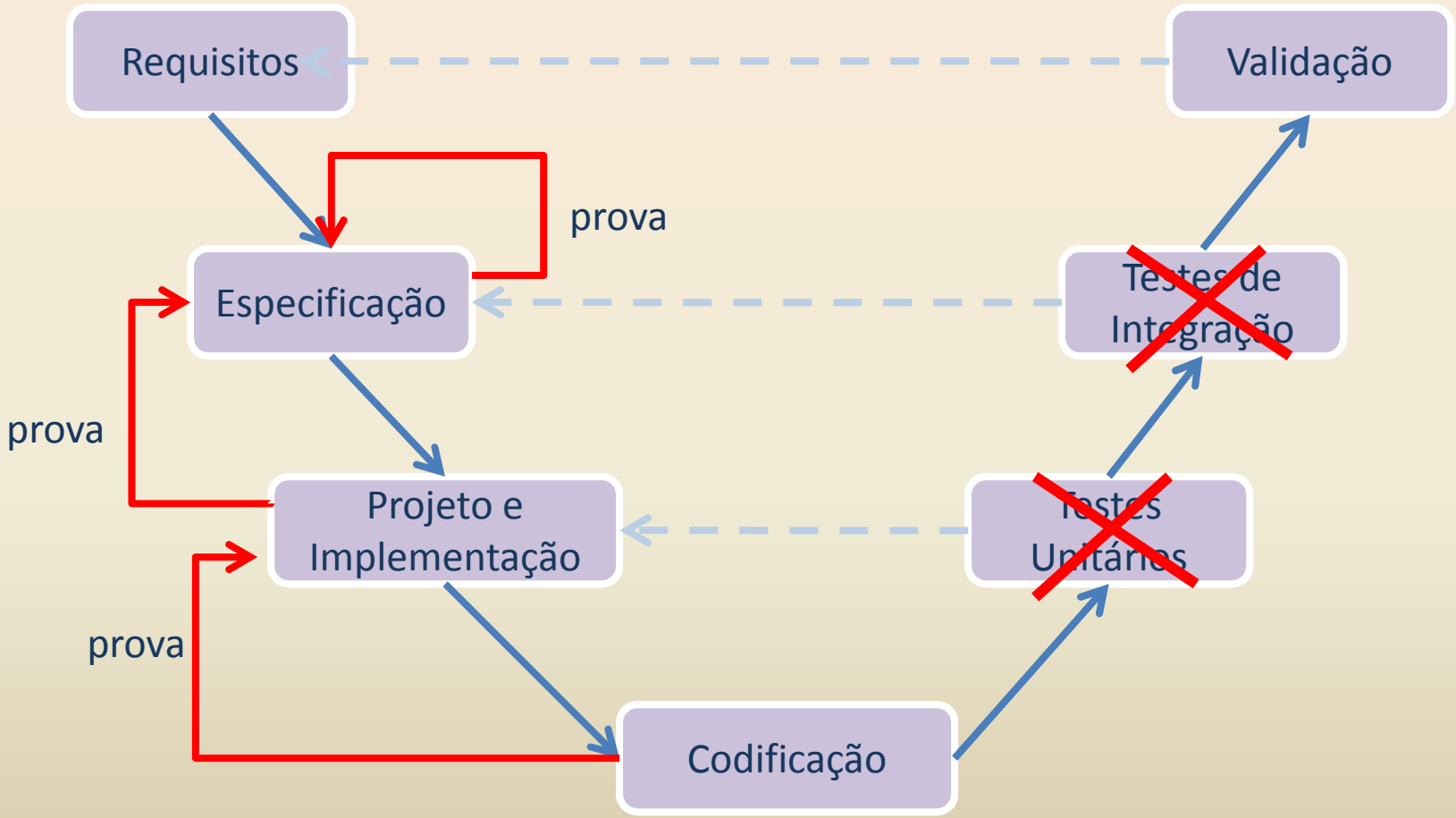
Klais

Processo Tradicional



Klais

Processo Usando o Método B



Klais

OCL – Object Constraint Language

- Criada pela IBM e padronizada pela OMG
- Complementar a UML
- Sintaxe e semântica formais
- Define ‘restrições’ sobre o modelo UML na forma de
  - Invariantes
  - Pré-condições
  - Pós-condições
  - ‘Guards’

Klais

OCL – Object Constraint Language

- Motivação
  - UML não é suficiente para descrever todos os aspectos da aplicação.
  - Anotações em linguagem natural para complementar os diagramas UML são imprecisas, ambíguas e incompletas.
  - Apoio a abordagens MDD/MDA.

Klais

OCL – Object Constraint Language

- Restrições
  - Associadas aos ‘elementos de modelagem’ (classes, relacionamentos, operações, etc)
  - Descritas como expressões lógicas sobre o estado da aplicação.
  - Invariantes: condições que devem se manter válidas durante toda a ‘vida’ da aplicação.
  - Pré-condições: devem ser válidas antes da execução de uma operação.
  - Pós-condições: devem ser válidas depois da execução de uma operação.
  - ‘guards’ – condições que devem ser válidas durante uma transição de estado.

Klais

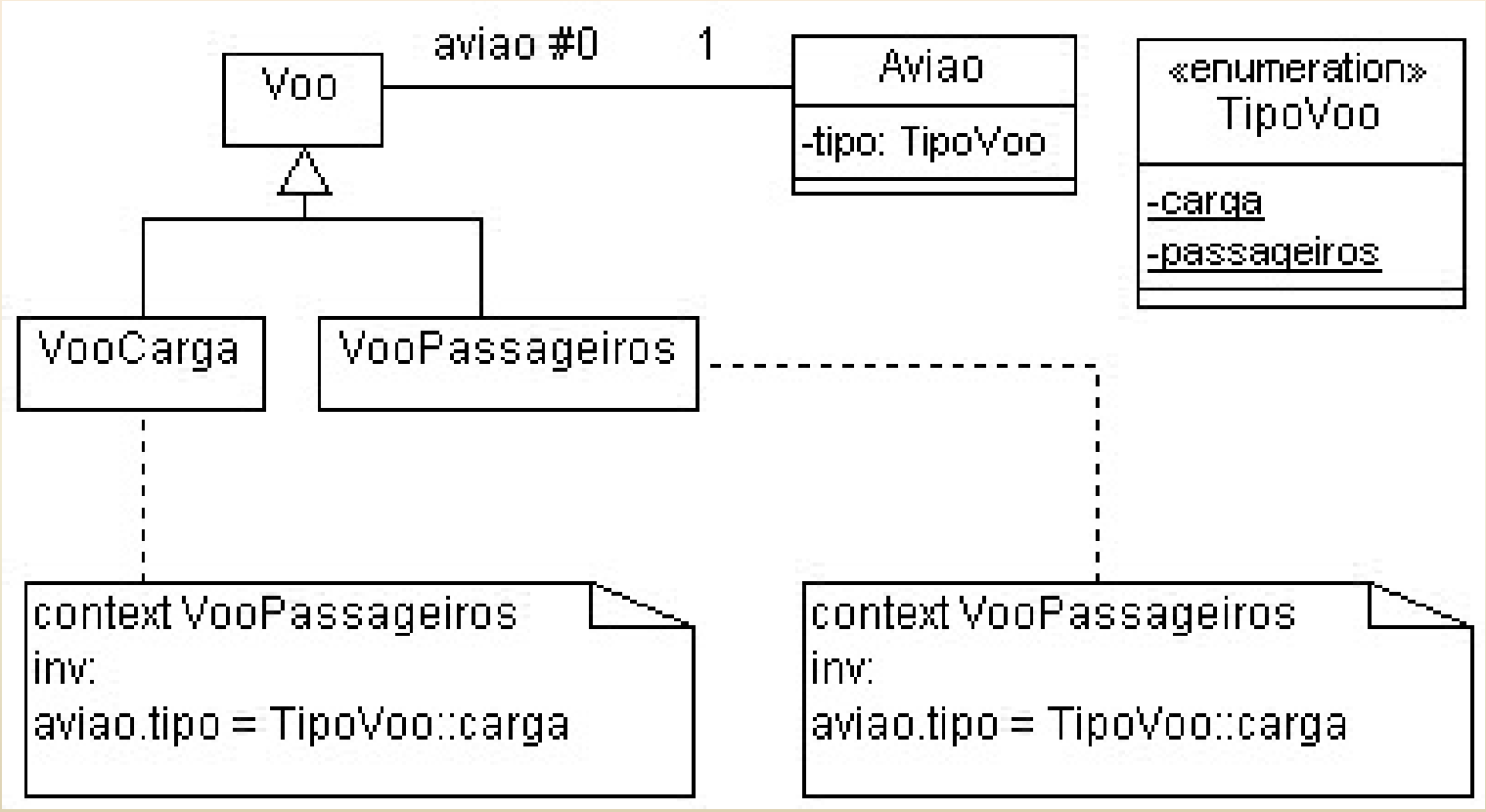
OCL – Object Constraint Language

- Avaliação das Restrições
  - Não tem ‘efeito colateral’
  - É instantânea (durante a avaliação, não ocorre nenhuma mudança de estado)
- OCL não é uma linguagem de programação
  - Baseada numa ‘máquina abstrata’
  - Tipos estereótipos, operações, sintaxe e semântica pré-definidos.

Klais

OCL – Object Constraint Language

- Um exemplo: invariantes



Klais

OCL – Object Constraint Language

- Um exemplo: pré e pós-condições

```
context Passageiro:: reserva(v: Voo)
        pre: v.passageiros → size  < v.maxPass
        post: v.passageiros = v.passageiros@pre→including(self)
```

Klais

OCL – Object Constraint Language

- **Tipos pré-definidos:**
  - Tipos primitivos: Integer, Real, String e Boolean
  - Coleções: Set e Sequence =
- **Tipos definidos pelo usuário (no modelo):**
  - Classes e enumerações

OCL – Object Constraint Language

- Operações com os tipos numéricos

Operation	Notation	Result type
equals	a = b	Boolean
not equals	a <> b	Boolean
less	a < b	Boolean
more	a > b	Boolean
less or equal	a <= b	Boolean
more or equal	a >= b	Boolean
plus	a + b	Integer or Real
minus	a - b	Integer or Real
multiply	a * b	Integer or Real
divide	a / b	Real
modulus	a.mod(b)	Integer
integer division	a.div(b)	Integer
absolute value	a.abs()	Integer or Real
maximum	a.max(b)	Integer or Real
minimum	a.min(b)	Integer or Real
round	a.round()	Integer
floor	a.floor()	Integer

Klais

OCL – Object Constraint Language

- Operações lógicas

Operation	Notation	Result type
or	a or b	Boolean
and	a and b	Boolean
exclusive or	a xor b	Boolean
negation	not a	Boolean
equals	a = b	Boolean
not equals	a <> b	Boolean
implication	a implies b	Boolean
if then else	if a then b1 else b2 endif	type of b

Klais

OCL – Object Constraint Language

- Operações com strings

Operation	Expression	Result type
concatenation	s.concat(string)	String
size	s.size()	Integer
to lower case	s.toLower()	String
to upper case	s.toUpper()	String
substring	s.substring(int, int)	String
equals	s1 = s2	Boolean
not equals	s1 <> s2	Boolean

Klais

OCL – Object Constraint Language

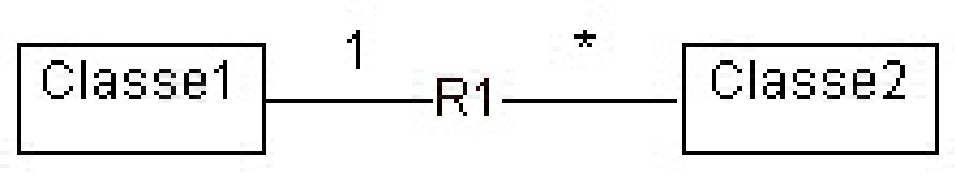
- Operações com strings

Operation	Description
size()	The number of elements in the collection
count(object)	The number of occurences of object in the collection.
includes(object)	True if the object is an element of the collection.
includesAll(collection)	True if all elements of the parameter collection are present in the current collection.
excludes(object)	True if the object is *not* an element of the collection.
excludesAll(collection)	True if all elements of the parameter collection are *not* present in the current collection.
isEmpty()	True if the collection contains no elements.
notEmpty()	True if the collection contains one or more elements.

OCL – ‘navegação’ pelos objetos

A sintaxe de OCL permite ‘navegar’ pelos objetos do modelo

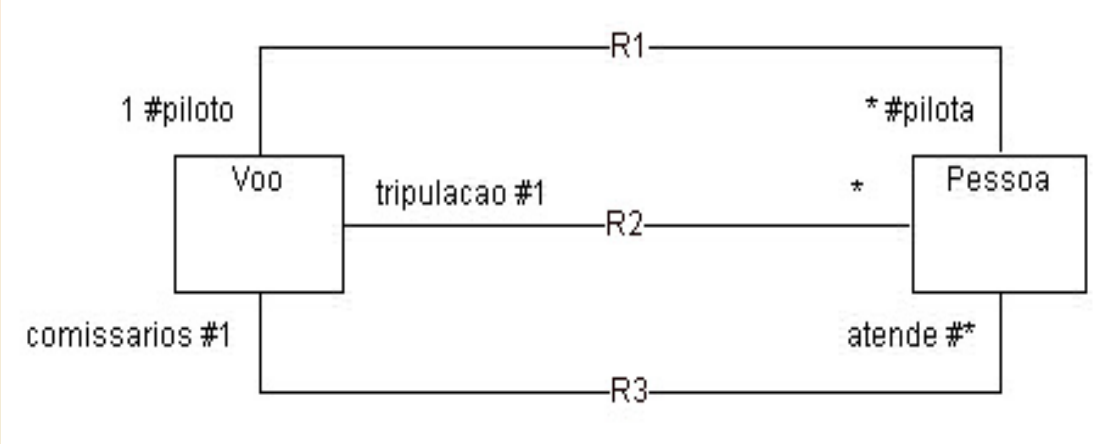
- Numa operação aplicada a um objeto,
  - **`self`** se refere ao próprio objeto
  - **`self.a`** se refere ao atributo a do objeto



  - **`self.R1`** devolve o conjunto de elementos associados ao objeto

Klais

OCL – um exemplo



- **O piloto é membro da tripulação**

```
context Voo
inv: self.R2->includes(self.R1)
```

- **Comissários são membros da tripulação**

```
context Voo
inv: self.R2->includesAll(self.R3)
```

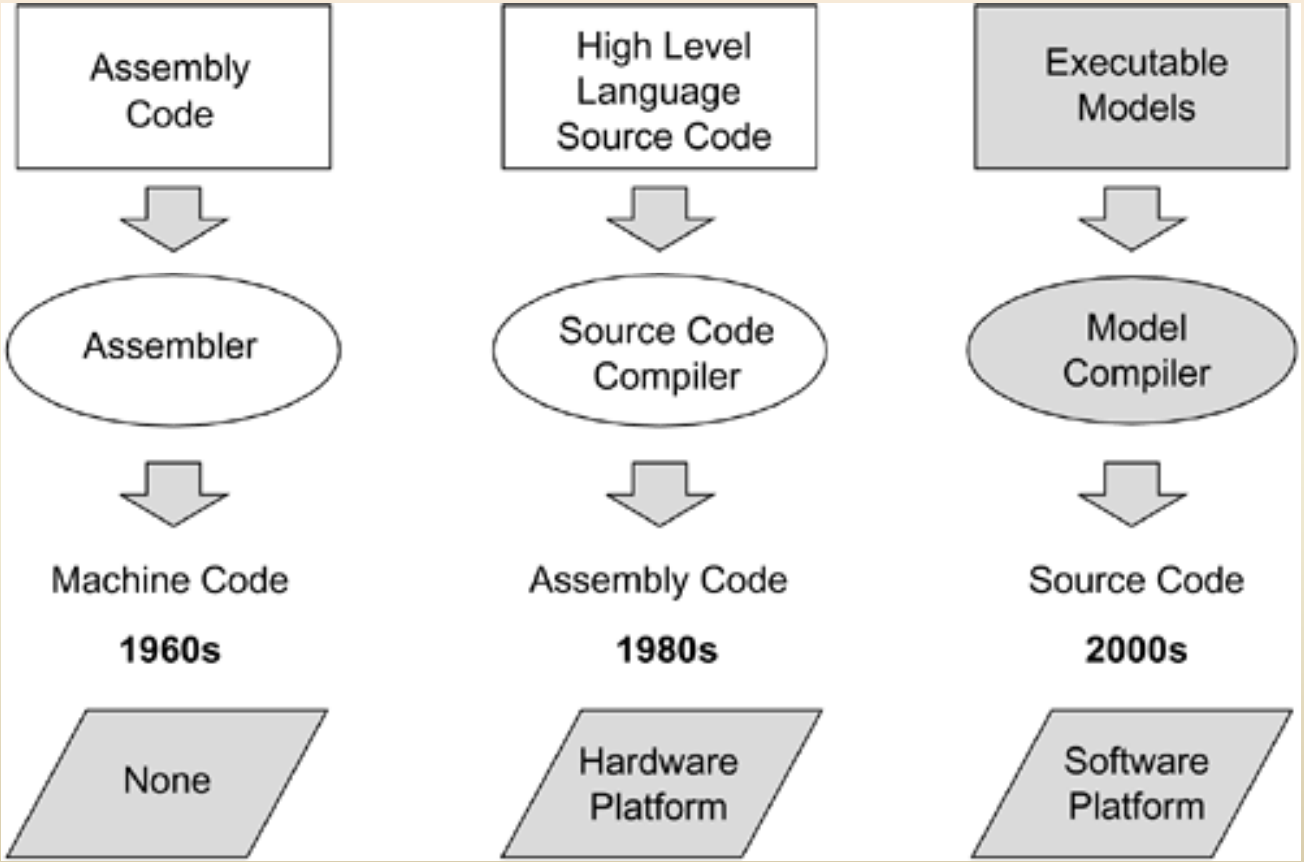
Klais

MDA – Model Driven Architecture

- Idéia geral
  - Utilização de modelos como base de apoio para o processo de desenvolvimento.
- Objetivos
  - Redução dos custos de desenvolvimento, validação e evolução
  - Portabilidadade/Independência de plataforma
- Padrao definido pelo OMG

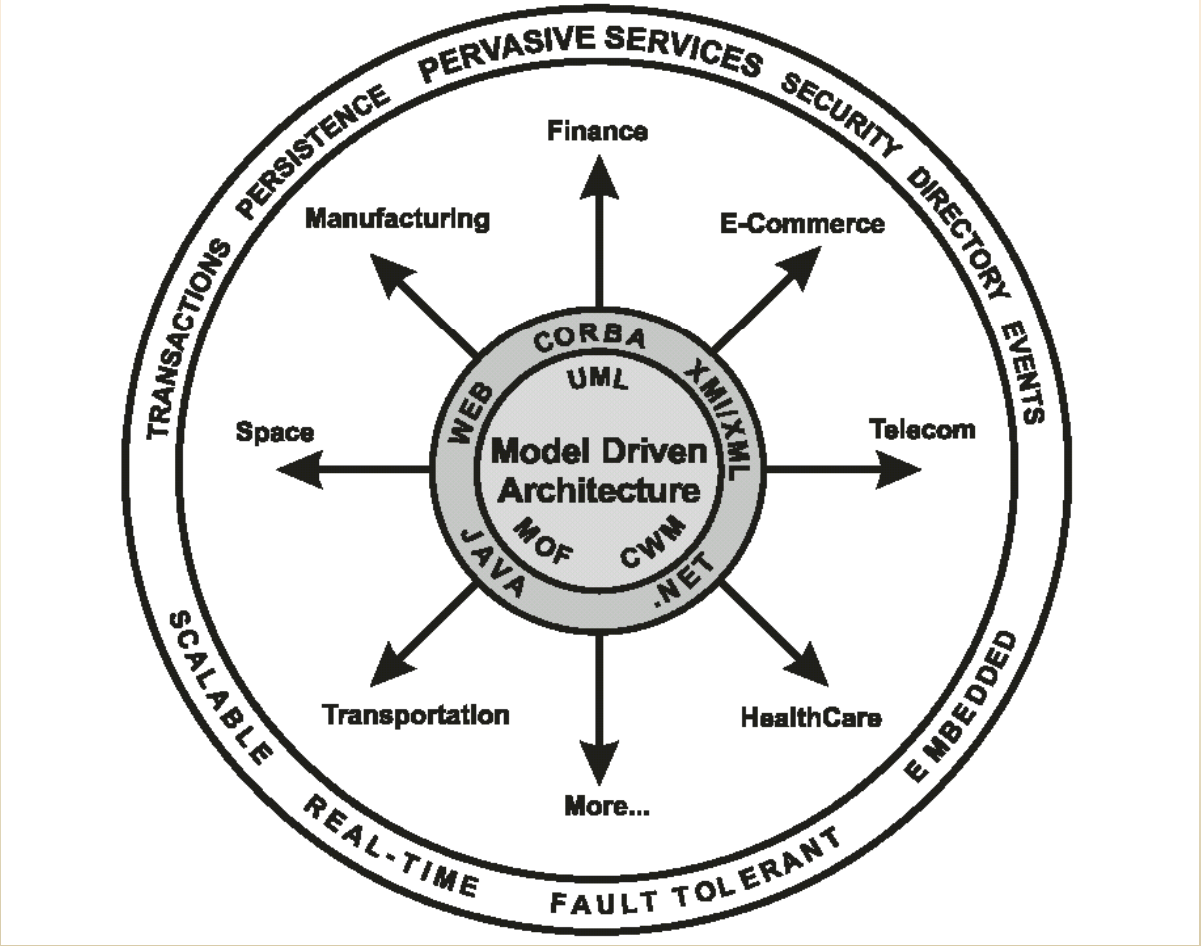
MDA – Model Driven Architecture

- A proposta



Klais

MDA – Model Driven Architecture

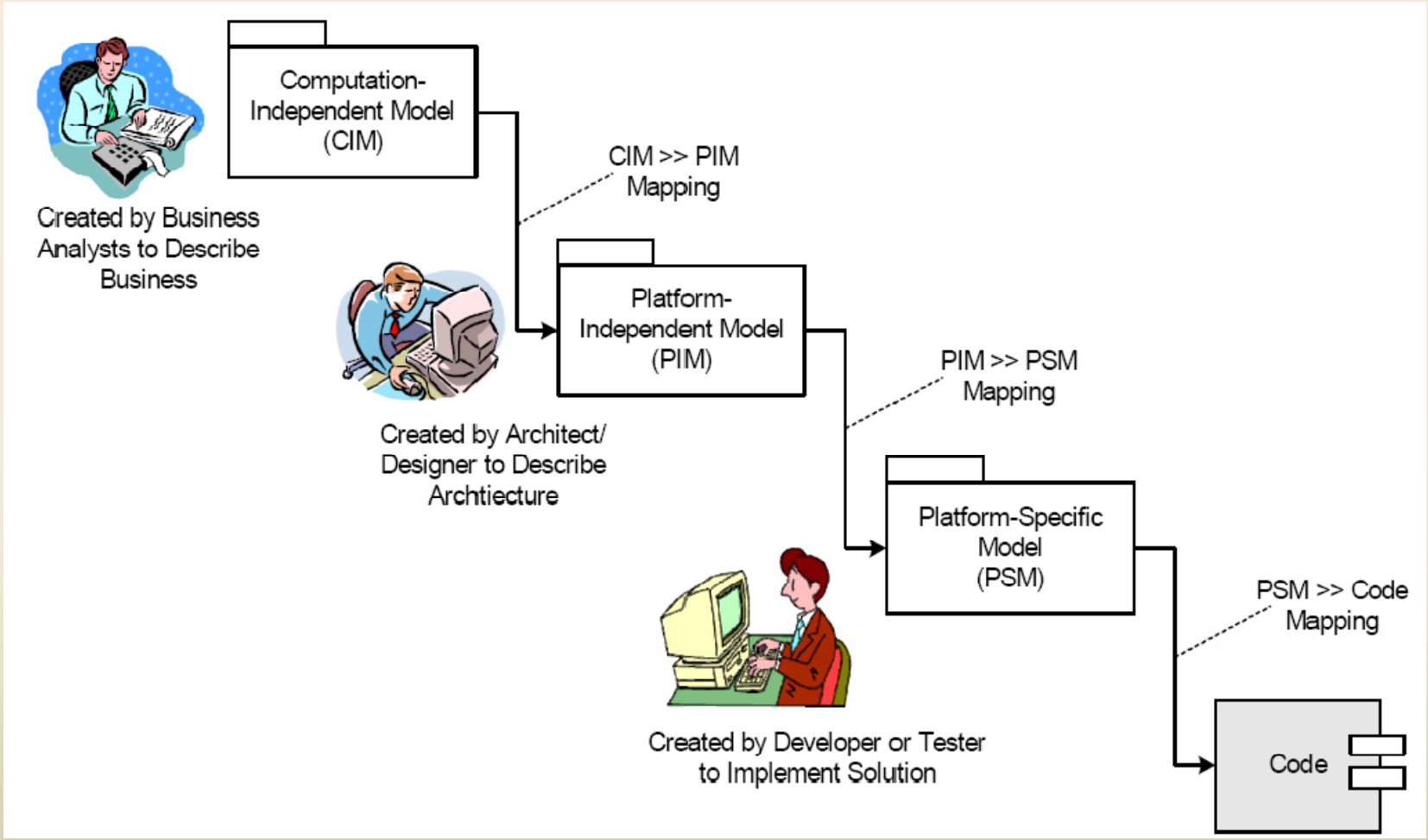


Klais

MDA – Model Driven Architecture

- Os padrões
  - UML – Unified Modeling Language
  - OCL – Object Constraint Language
  - MOF – Meta Object Facility
  - XMI – XML Metadata Interchange
  - CWM – Common Warehouse Metamodel
- Os modelos
  - CIM – Computation Independent Model
  - PIM – Plataform Independent Model
  - PSM – Platform Specific Model

MDA – Model Driven Architecture



Klais

MDA – Abordagem ‘elaboracionista’

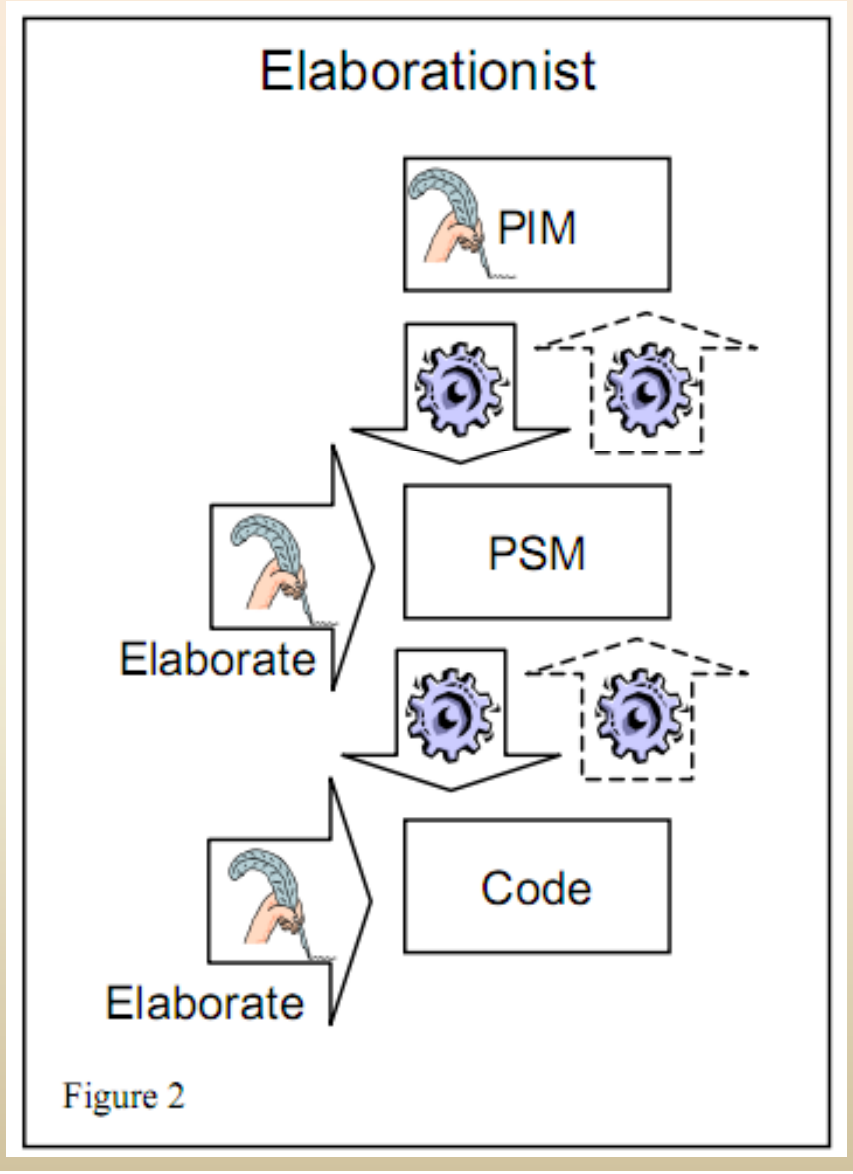


Figure 2

Klais

MDA – Abordagem 'traducionista'

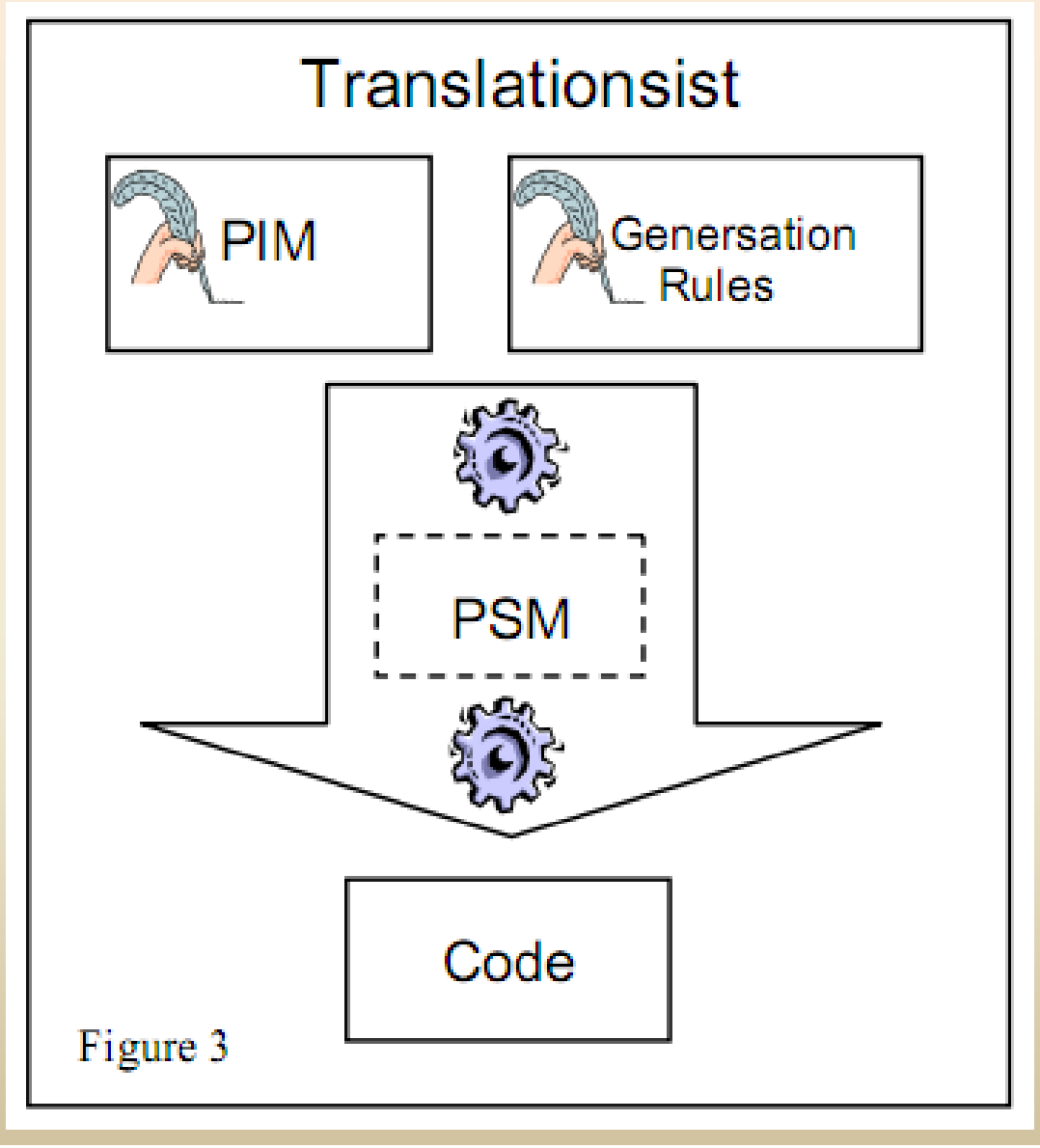


Figure 3

Klais

MDA – Model Driven Architecture

- Ainda é uma área recente.
- Os resultados práticos
  - Voltados a frameworks específicos ou
  - Nichos específicos
- Na prática
  - O enriquecimento dos modelos com as informações necessárias torna a modelagem tão complexa e cara quanto o desenvolvimento.
  - Modelagem independente de plataforma: deve se basear nos recursos comuns a todas as plataformas.

Klais

JML – Java Modeling Language

- Idéia geral
  - pré-condições
  - pós-condições
  - invariantes

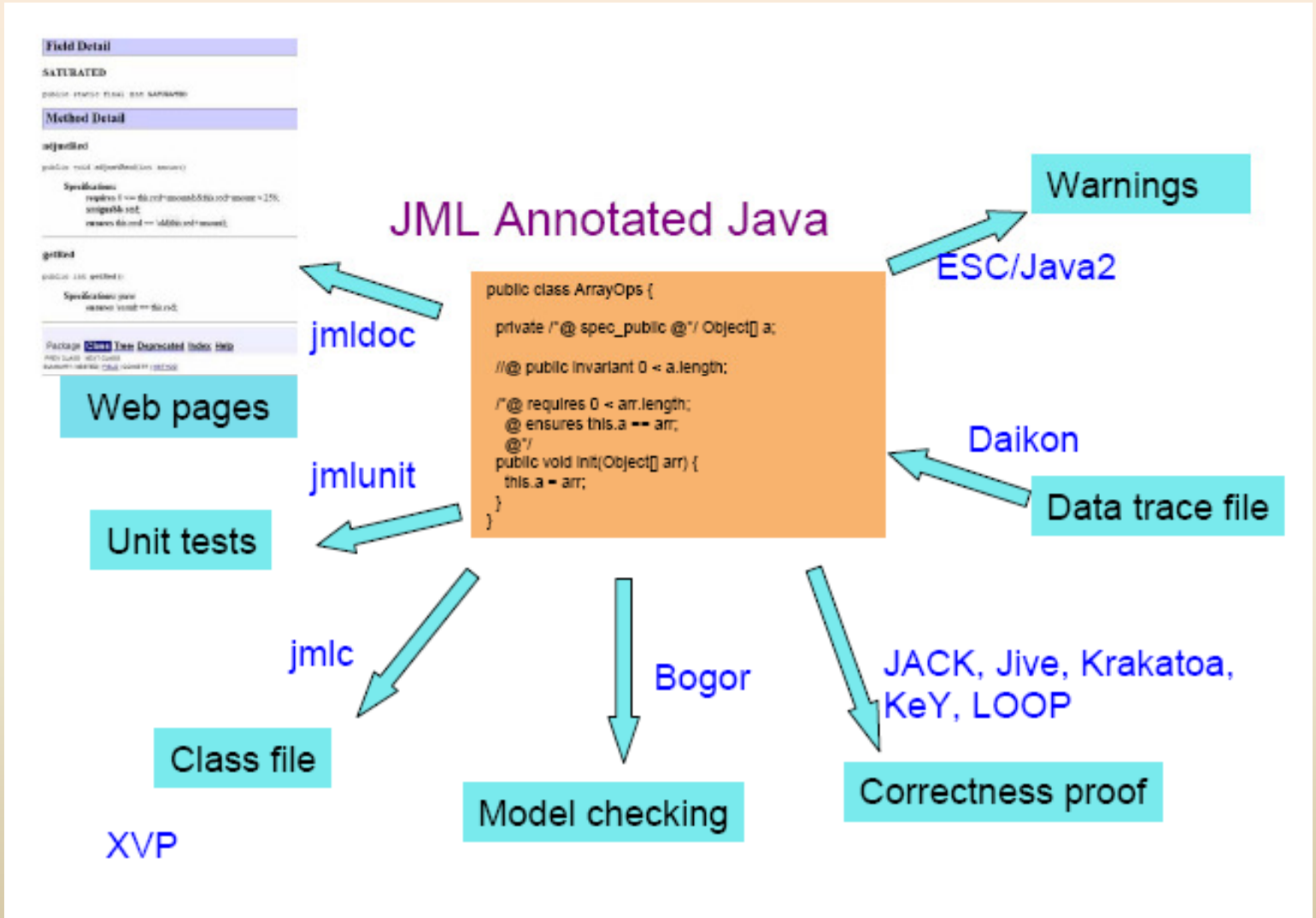
- Descritas com base nos mesmos tipos e operações definidos pela Máquina Virtual Java.
- Aparecem no código Java como anotações em comentários.

Klais

JML – Java Modeling Language

```
public class ArrayOps {
  private /*@ spec_public @*/ Object[] a;
  //@ public invariant 0 < a.length;
  /*@ requires 0 < arr.length;
    @ ensures this.a == arr;
  @*/
public void init(Object[] arr) {
  this.a = arr;
}
```

JML – Java Modeling Language



Klais

JML – Java Modeling Language

- As possibilidades:
  - Prova Formal (Jack, Jive, Krakatoa, KeY, LOOP)
  - Verificação de Modelos (Bogor)
  - Verificação em tempo de execução (jmlc)
  - Testes (jmlunit)
  - Geração de documentação (jmldoc)

Um exemplo (1)

```
public class BoundedStack {

  private /*@ spec_public nullable @*/
     Object[] elems;
  private /*@ spec_public @*/ int size = 0;

  //@ public invariant 0 <= size;
  /*@ public invariant elems != null
    @    && (\forall int i;
    @              size <= i && i < elems.length;
    @              elems[i] == null);
    @*/
```

Um exemplo (2)

```
/*@ requires 0 < n;
  @ assignable elems;
  @ ensures elems.length == n;
  @*/
public BoundedStack(int n) {
  elems = new Object[n];
}
```

Um exemplo(3)

```
/*@ requires size < elems.length-1;
  @ assignable elems[size], size;
  @ ensures size == \old(size+1);
  @ ensures elems[size-1] == x;
  @ ensures_redundantly
  @     (\forall int i; 0 <= i && i < size-1;
  @                 elems[i] == \old(elems[i]));
  @*/
public void push(Object x) {
  elems[size] = x;
  size++;
}
```

Klais

Um exemplo(4)

```
/*@ requires 0 < size;
  @ assignable size, elems[size-1];
  @ ensures size == \old(size-1);
  @ ensures_redundantly
  @       elems[size] == null
  @   && (\forall int i; 0 <= i && i < size-1;
  @                 elems[i] == \old(elems[i]));
  @*/
public void pop() {
  size--;
  elems[size] = null;
}
```

Klais

Um exemplo(5)

```
  /*@ requires 0 < size;
    @ assignable \nothing;
    @ ensures \result == elems[size-1];
    @*/
  public /*@ pure @*/ Object top() {
    return elems[size-1];
  }
}
```

Klais

Referências

- Moura, Arnaldo Vieira, “Especificações em Z, uma introdução”, Editora UNICAMP, 2001
- Spivey, J.M., “An Introduction to Z and formal specifications”, Software Engineering Journal, 1989
- Wanderley Neto, E. B, Moreano N. B., “O Modelo Z”, relatório técnico IC-Unicamp, 2001
- Wirth, N., “Programação Sistemática”, Editora Campus, 1978
- King, Josh, “The B Method”, CSE 888, OSU, CS & E Dept , 2007
- Medeiros Jr., V., et. al., “Ferramentas de Apoio ao Desenvolvimento em Metodologia B”, Lab. Consiste, UFRN
- Warmer, J., Kleppe, A., “The Object Constraint Language. Precise Modeling with UML. Addison-Wesley”, 1999
- Frankel, D., et. al.,“The Zachman Framework and the OMG's Model Driven Architecture. White Paper, Business Process Trend”,2003.
- McNeile,Ashley, “MDA: The vision with a Hole ?”, www.metamaxim.com, 2003
- Leavens,G.T., et. al.,”A JML Tutorial”, Univ. Central Florida, 2007