Alloy

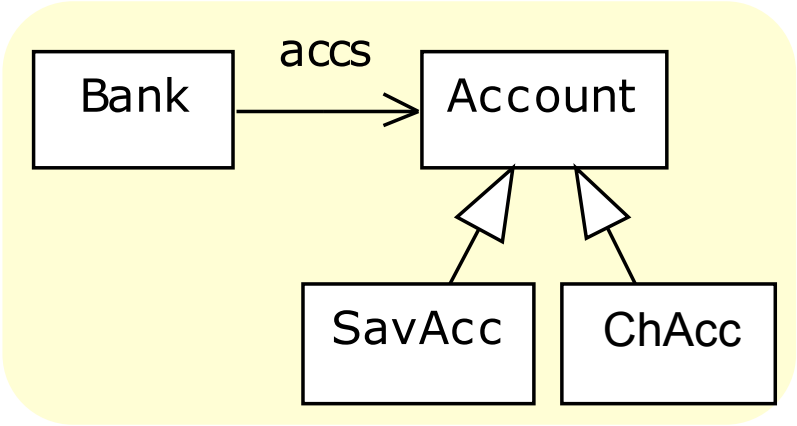
Rohit Gheyi

Tiago Massoni

Instalação

- Home Page
  - http://alloy.mit.edu
- Requer JRE
- Rodar
  - java –jar alloy.jar
- API
  - http://alloy.mit.edu/alloy4/api.html

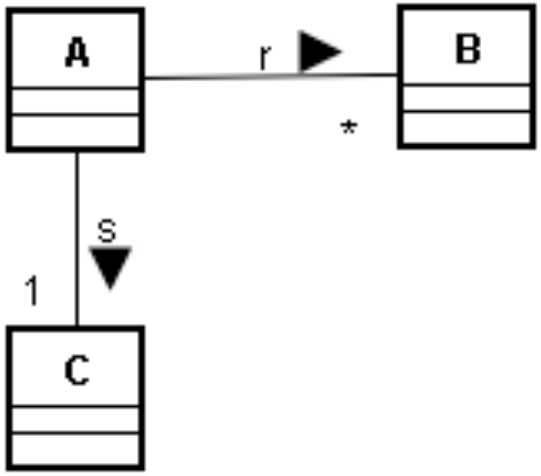
Exemplo: Assinaturas e Relações




```
module banco
sig Banco {
  contas: set Conta
}
sig Conta {}
sig ContaCorr extends Conta {}
sig ContaPoup extends Conta {}
```

Assinaturas e Relações

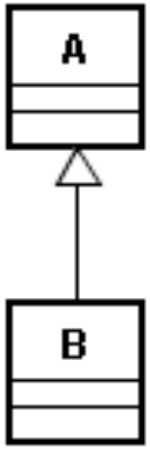
- assinatura = conjunto de objetos
- relação relaciona elementos entre conjuntos (globais)




```
sig A {
  r: set B,
  s: one C,
  t: lone D
}
sig B, D{}
one sig C{}
```

Herança

- B herda as relações e fórmulas sobre a A
- Em Alloy, A tem acesso as relações de B

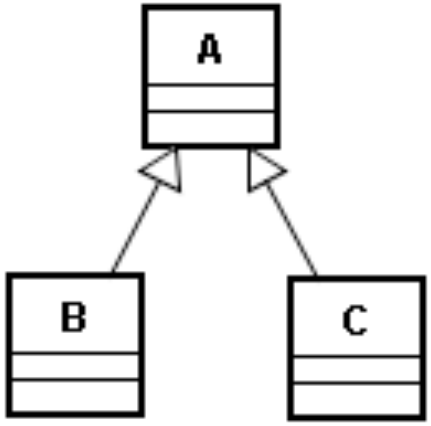



```
sig A { … }
sig B extends A {
  …
}
```

$B \subseteq A$

Subset Signature

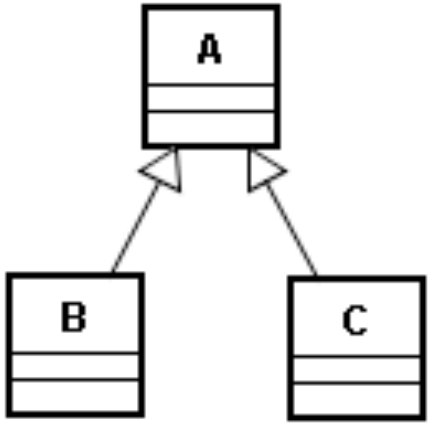
- A diferença é que as subassinaturas não são disjuntas
  - Fato implícito BIC = 0




```
sig A { … }
sig B,C in A {}
```

Assinatura Abstrata

- Todas as instâncias da assinatura A pertencem a assinatura B ou C



abstract sig A { … }
sig B,C **extends** A {}

Fatos

- Declara um conjunto de invariantes do modelo

```
fact nome {
    f1
    f2 …
}
```

Conjunção
$f_1 \wedge f_2 \wedge \ldots$

Cardinalidade

- **#**
  - #exp = 10
  - exp tem exatamente 10 elementos
- **one**
  - **one** exp
  - exp resulta em um elemento
- **some**
  - **some** exp
  - exp possui algum elemento
- **no**
  - **no** exp
  - exp não possui elemento

Operadores de Conjuntos

- União (U)
  - A+B
- Interseção (I)
  - A&B
- Diferença (/)
  - A-B
- Subconjunto (⊆)
  - A **in** B
- Negação (!)
  - ! (A **in** B)

Operadores Lógicos

- Conjunção (∧)
  - P **and** Q
  - P && Q
- Disjunção (v)
  - P **or** Q
  - P || Q
- Implicação (⇒)
  - P => Q
- Biimplicação (Û)
  - P <=> Q

Quantificação

- Universal (∀)
  - **all** x:A | p(x)
  - Para todos os x do tipo A, tal que p(x) é verdade
- Existencial (∃)
  - **some** x:A | p(x)
  - Existe um x do tipo A, tal que p(x) é verdade

Fatos Anexados à assinatura

```
sig Host {}
sig Link{ from,dest: Host}
fact {
    all l:Link | l.from != l.dest
    }
```

é o mesmo de...

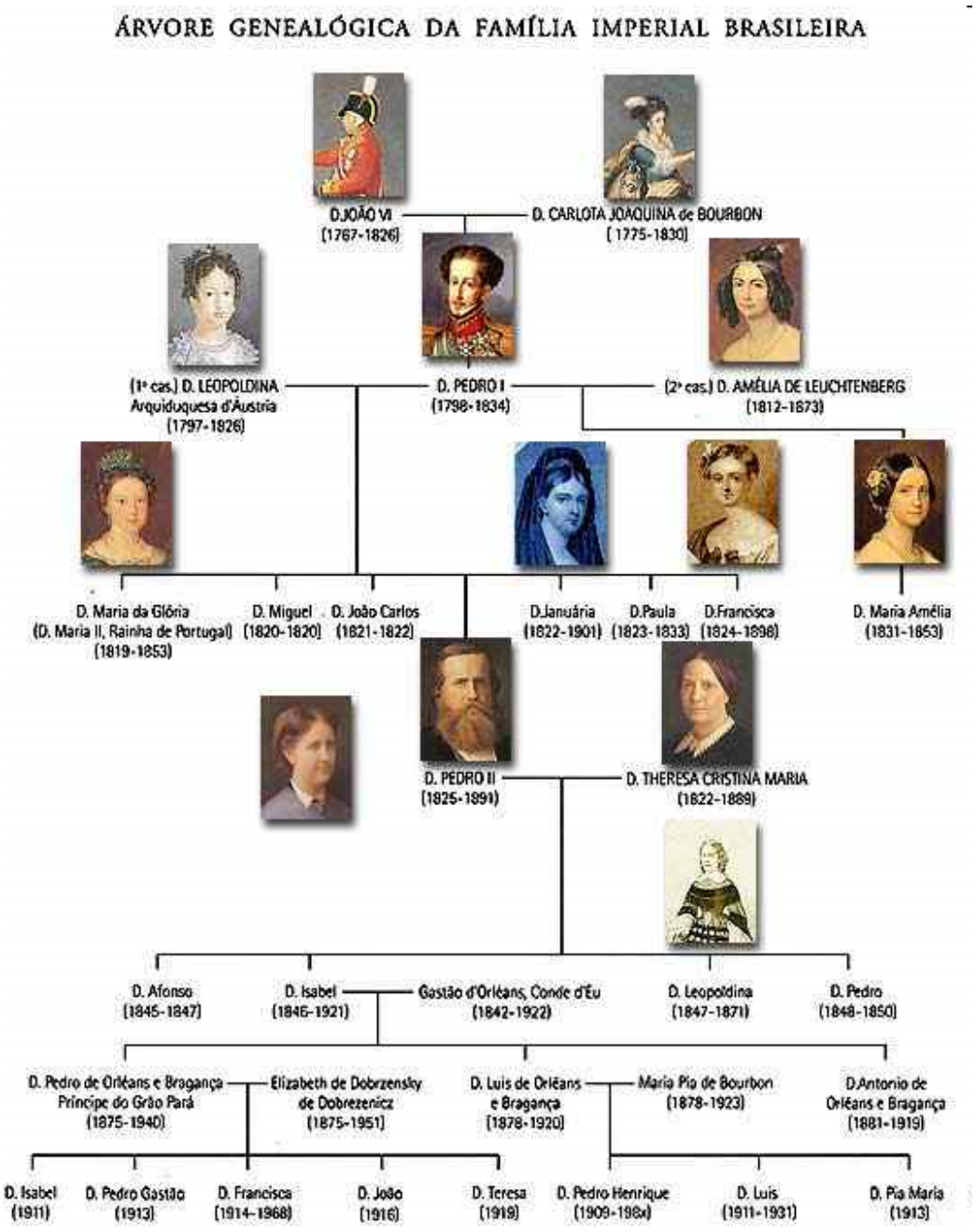
```
sig Host {}
sig Link{ from,dest: Host}{ from != dest }
```

Operadores

- Join (.)
  - Composição relacional
  - (a,b).{(b,c), (d,e), (b,f)} = {(a,c), (a,f)}
- Transpose (~)
  - Reversa
  - Se r = {(b,c), (d,e)}, ~r = {(c,b), (e,d)}

como expressar ancestrais

```
sig Pessoa {
  pais: set Pessoa
}
```

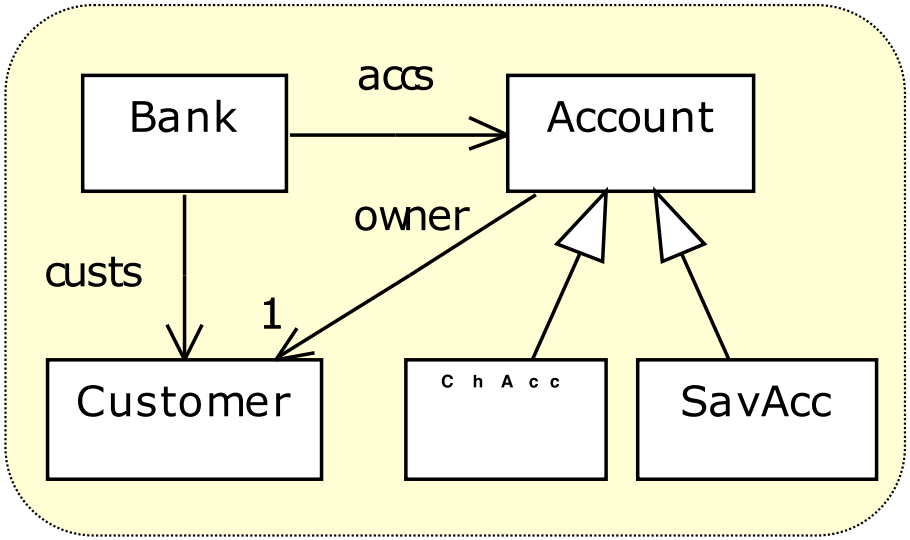


Fecho Transitivo

- Aplicação de r uma ou mais vezes
- Relações binárias de tipos relacionados
- *r e ^r

Exercício

- Especifique em Alloy que os clientes do banco são os mesmos dos donos das contas do banco.



Predicados

- Declara um conjunto de fórmulas, sendo aplicado em outras fórmulas
- e ajuda a especificar operações...

```
pred temContas[b:Banco] {
    some b.contas
}
fact {
    all b: Banco | temContas[b]
}
```

Funções

- Declara um valor relacional, usado em outras fórmulas

```
fun contasDoBanco[b:Banco]: set Conta {
    b.contas
}
fact {
    all b: Banco |
        #contasDoBanco[b] > 1
}
```

Análises

- Run
  - Encontra uma instância válida para o predicado ou função
  - Satisfaz aos invariantes do modelo e as fórmulas do predicado ou função
- Check
  - Checa em um dado escopo se as fórmulas de um asserção são válidas a partir da especificação (todos os casos)

inteiros em Alloy

- número inteiro dentro de um objeto Int

```
sig Node { adj: Node -> lone Int }

fact {
    all n: Node |
    let w= n.(n.adj) |
        some w => int[w] = 0
}
```

cast de Int (objeto) para int (valor)

escopo (número de bits de representação de inteiros)

```
run show for 3 Int
```

boolean em Alloy

```
sig Fone {
    foraGancho, tocando: Boolean
    }
```

não existe!
evitar...

melhor classificar

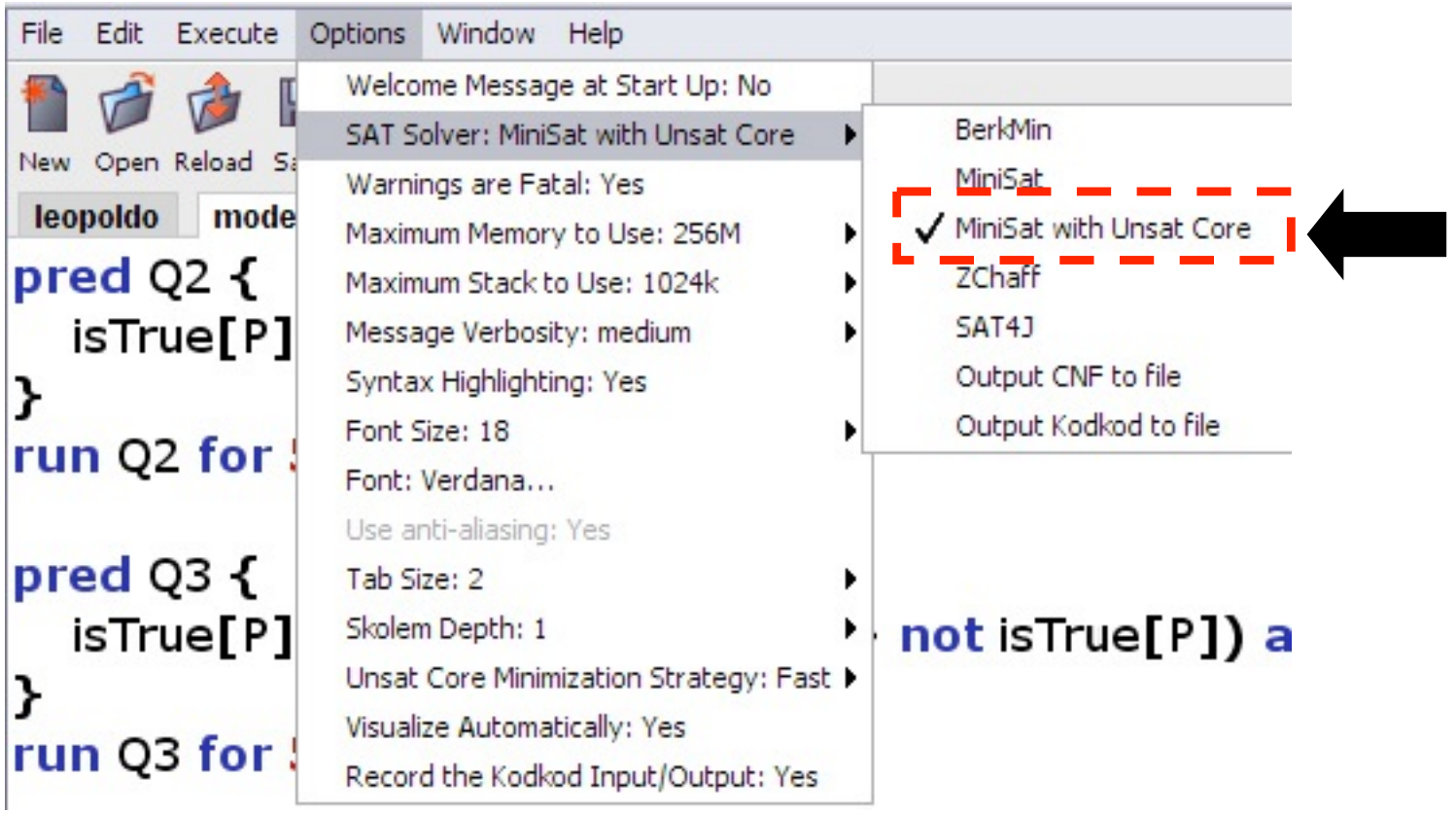
```
sig Fone { }
sig foraGancho, tocando in Fone {}
fact {
    no foraGancho & tocando
}
```

Problema

Se uma especificação estiver inconsistente, o que fazer?

Core Unsat

Tem alguns solvers que ajudam



Exemplo

No instance found. Predicate may be inconsistent. 47ms.
Core contains 2 top-level formulas. 16ms.

```
module inconsistente
sig A {
  r: one B
}
sig B {}
fact {
  some a:A | no a.r
}
pred show[] {}
run show
```