Especificación, Validación y Verificación utilizando Métodos Formales Livianos

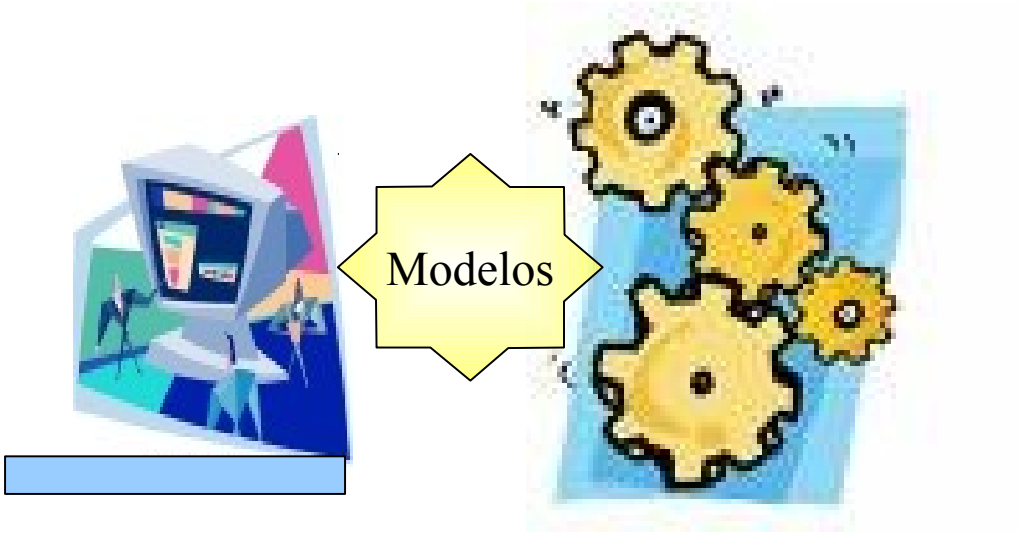
Ana Garis
agaris@unsl.edu.ar

Maestría en Ingeniería de Software - 2014 - UNSL

Agenda

- Introducción
- Alloy
- La lógica de Alloy
- Especificando con Alloy
- Verificación y Validación

Introducción



Proceso de Desarrollo de SW

Introducción

¿Por qué usamos modelos?

Introducción

¿Por qué usamos modelos?

1- Comprender el Problema

Es incompleto ?
Es ambiguo ?
Es contradictorio?

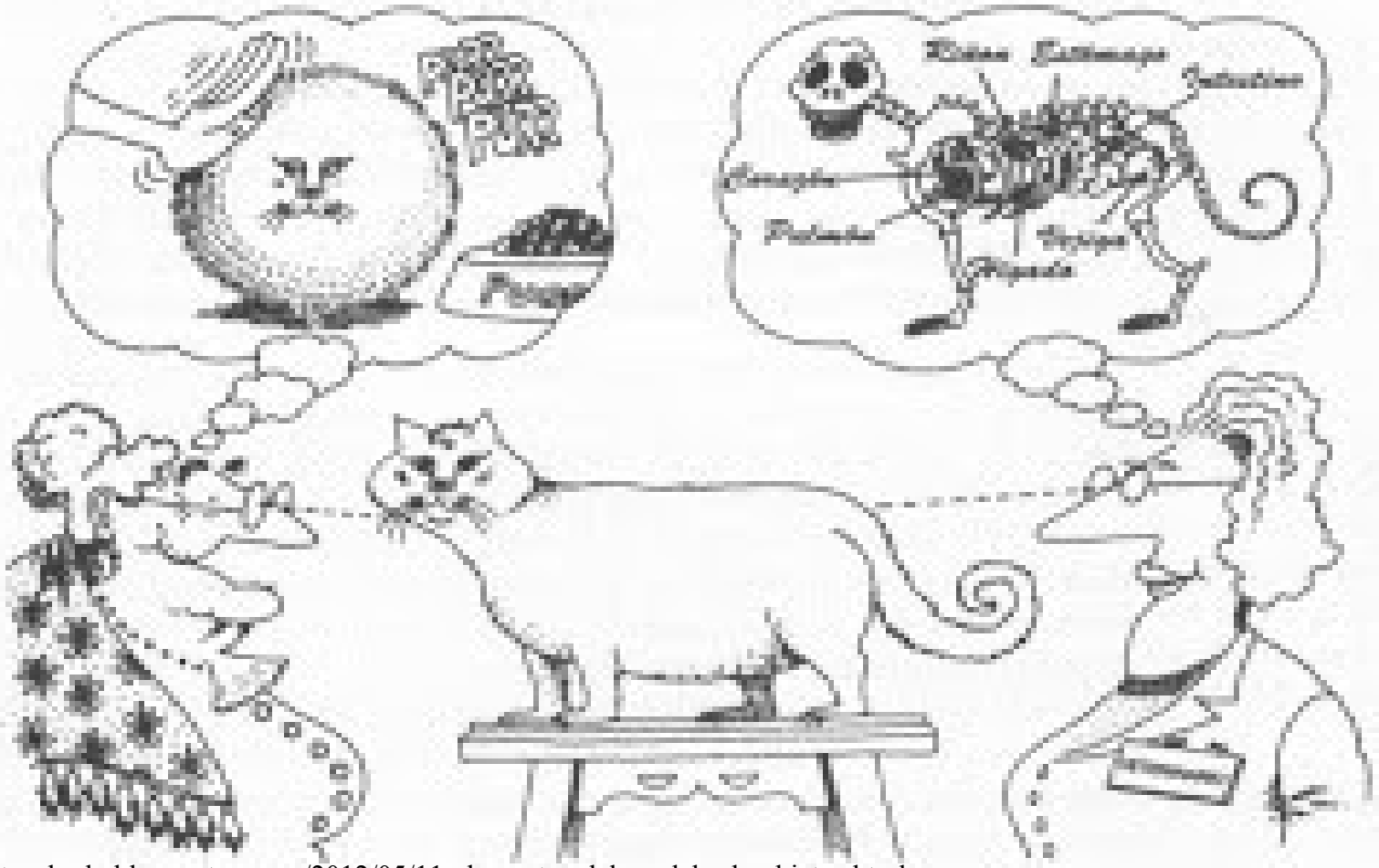
⇨

Especificar enunciado preciso

Introducción

¿Por qué usamos modelos?

2- Abstraer el problema, independiente de la perspectiva del observador.



[Fuente: http://pooitsavlerdo.blogspot.com.ar/2012/05/11-elementos-del-modelo-de-objetos.html

Introducción

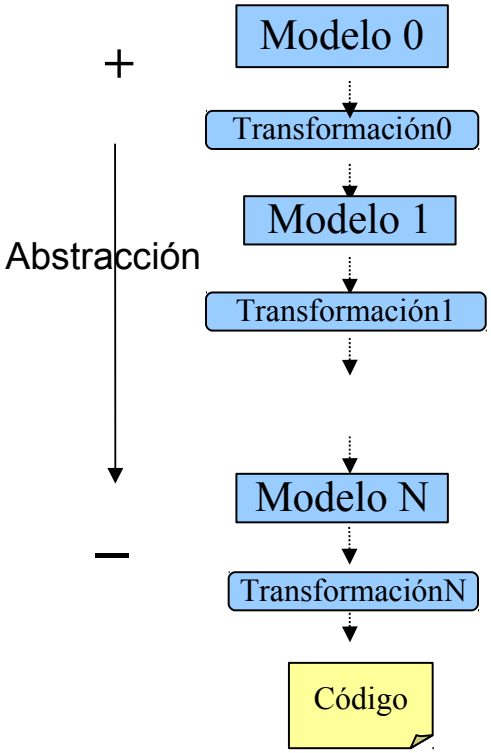
¿Por qué usamos modelos?

- Comprender el problema que se desea resolver
- Especificar un sistema de SW
- Comunicar con otros desarrolladores de SW
- Analizar posibilidades de reutilización

Introducción

Desarrollo Dirigido por Modelos

(Model-Driven Development -MDD)



Introducción

¿Qué necesitamos para construir modelos?

Introducción

¿Qué necesitamos para construir modelos?

- Lenguaje de modelado
  - informal
  - semi-formal -> UML
  - formal -> OCL, Z, Alloy

Agenda

- Introducción
- **Alloy**
- La lógica de Alloy
- Especificando con Alloy
- Verificación y Validación

Alloy

“The core of software development is the design of abstractions.”

“An abstraction is not a module, or an interface, class, or method; it is a structure, pure and simple - an idea reduced to its essential form.”

- Daniel Jackson

Alloy

Filosofía

- micromodelos
- analizables
- declarativos

Alloy

Micromodelos

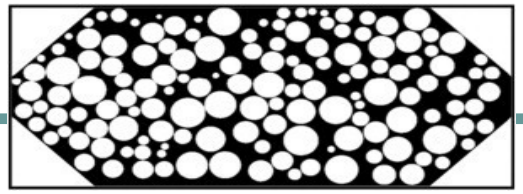
- Representación del problema

  líneas leng. modelado (-) : : líneas leng. programac. (+)

- Aumentar calidad del sistema:

  chequear modelo (- t) : : re-escribir y re-testear código (+ t)

En lugar de modelar todos los aspectos del sistema, focalizar en los mas **críticos**. Factorizar los aspectos mas relevantes que no son visualizados con un modelo complejo.



Alloy

Analizables

- Construir el modelo incrementalmente con un analizador para simular, chequear : : Usar lápiz y papel

- Simular → Ayuda a definir propiedades importantes

- Chequear → Certificar si el modelo tiene propiedades deseadas y cuáles son sus consecuencias

Permite tener una sensación de progreso y confianza que el modelo no tiene errores.

Alloy

Declarativos

Describe el efecto de un comportamiento (el qué)
sin definir el mecanismo (cómo)

- Declarativos adecuados para expresar abstracciones : :
  Operacionales adecuados para expresar detalles

- Permite empezar diciendo muy poco facilitando las descripciones parciales.

Alloy

Alloy vs. Otros enfoques

Metodos Formales

Formalizar diseño
Z, OCL, Isabelle
Sin código
(abstracto)

Ejercitar diseño
Escribir pruebas,
Chequear propiedades
Complejo y costoso

Metodos Ágiles

Formalizar diseño
Java, C
Mucho código
(detallado)

Ejercitar diseño
Escribir casos de test

Costoso
Cobertura pobre

Alloy

Es un lenguaje formal “liviano”, basado lógica de primer orden y soportado por un Analizador SAT (SATisfiability solver).

Alloy

Analizador SAT (SATisfiability solver)

Dada una fórmula booleana F, determinar si existe una interpretación, tal que asignando valores a las variables de la fórmula pueda ser evaluada como VERDADERA.

F es satisfacible si existe al menos un modelo

Alloy

Bases

1- Todo es una relación

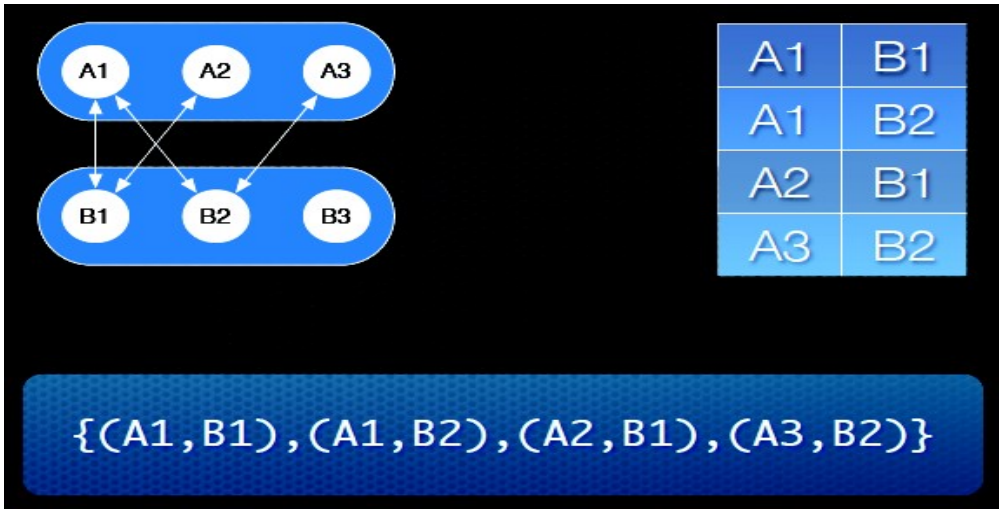
2- Lógica no especializada

3- Contraejemplos y alcance

4- Analizador SAT

Alloy

1- Todo es una relación



Los conjuntos son **relaciones** con aridad igual a 1:

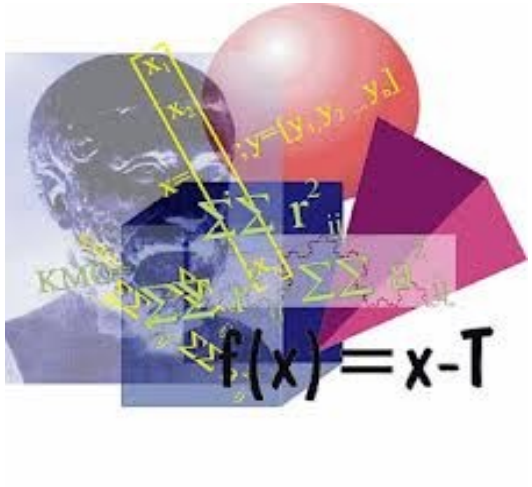
Ej. Time = {(T1),(T2),(T3),(T4)}

Aridad: número de atomos en cada tupla.

Alloy

2- Lógica no especializada

Alloy utiliza lógica básica, no introduce constructores especiales, de sincronización, concurrencia, etc.



Alloy

3- Contraejemplos con alcance

- La mayoría de los **errores** pueden ser encontrados a través de pequeños contraejemplos.

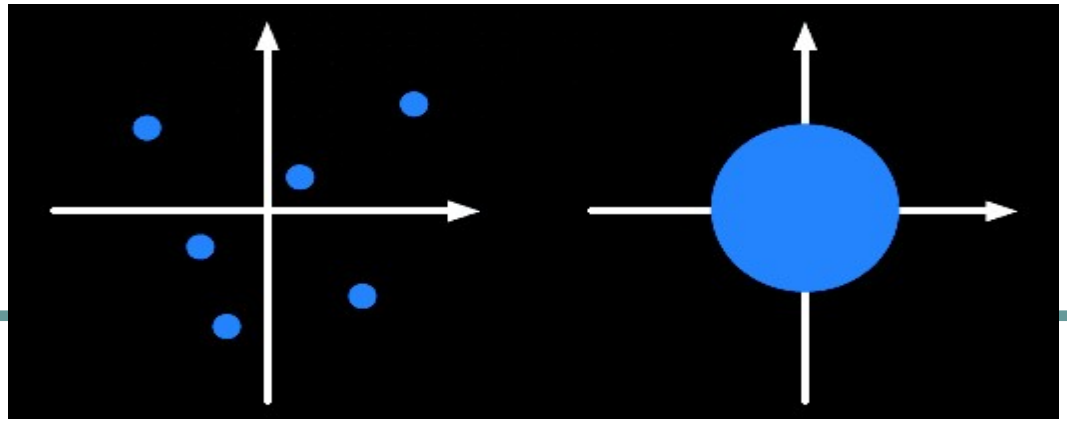
- En lugar de construir una prueba, se busca una refutación.

- Se define un **alcance** para limitar la cantidad de instancias.

- El objetivo es encontrar un modelo, haciéndolo **decidible** pero limitando el universo del problema.

Algunos casos

Alcance arbitrario

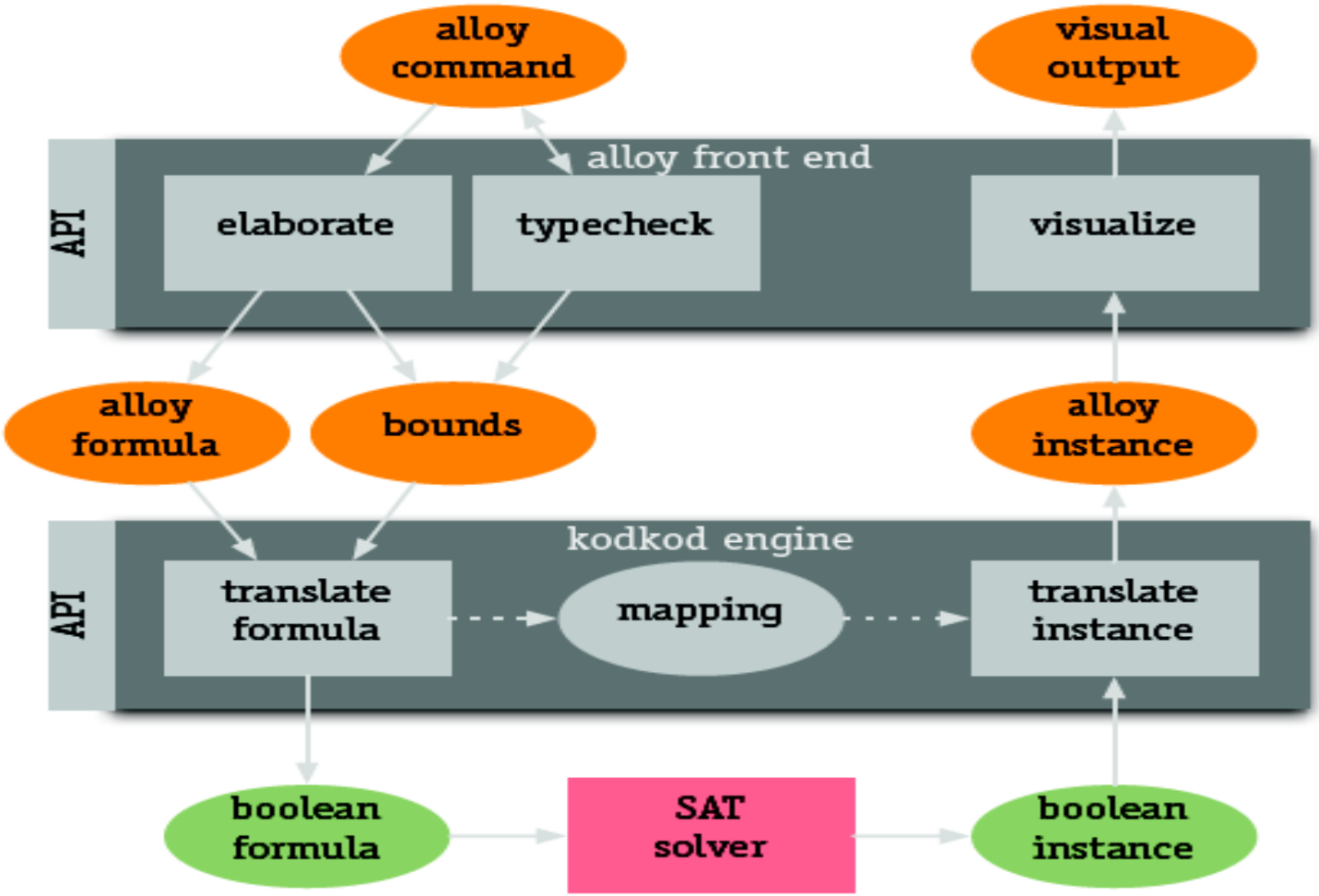


Todos los casos

Alcance pequeño

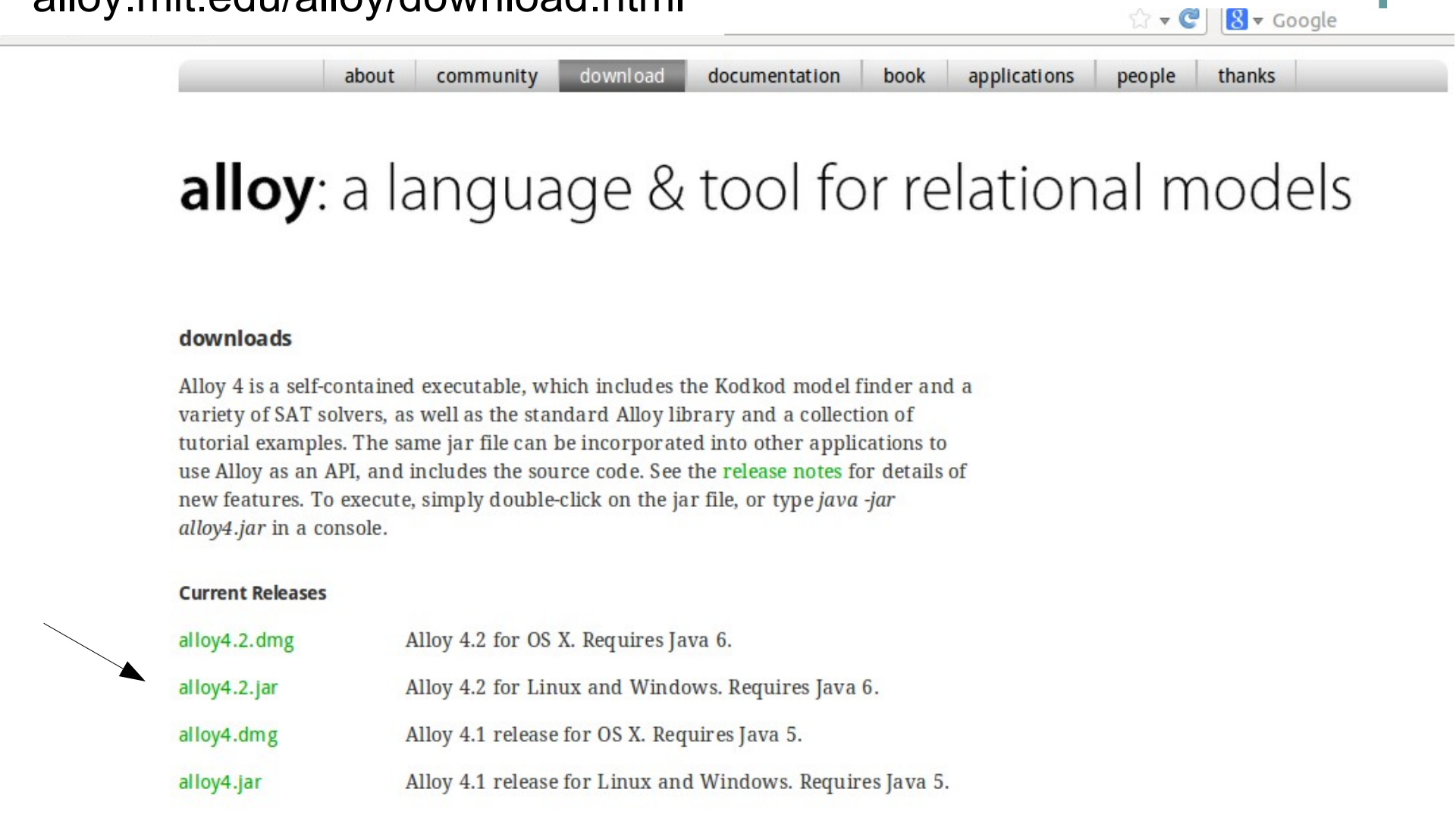
Alloy

4- Analizador SAT



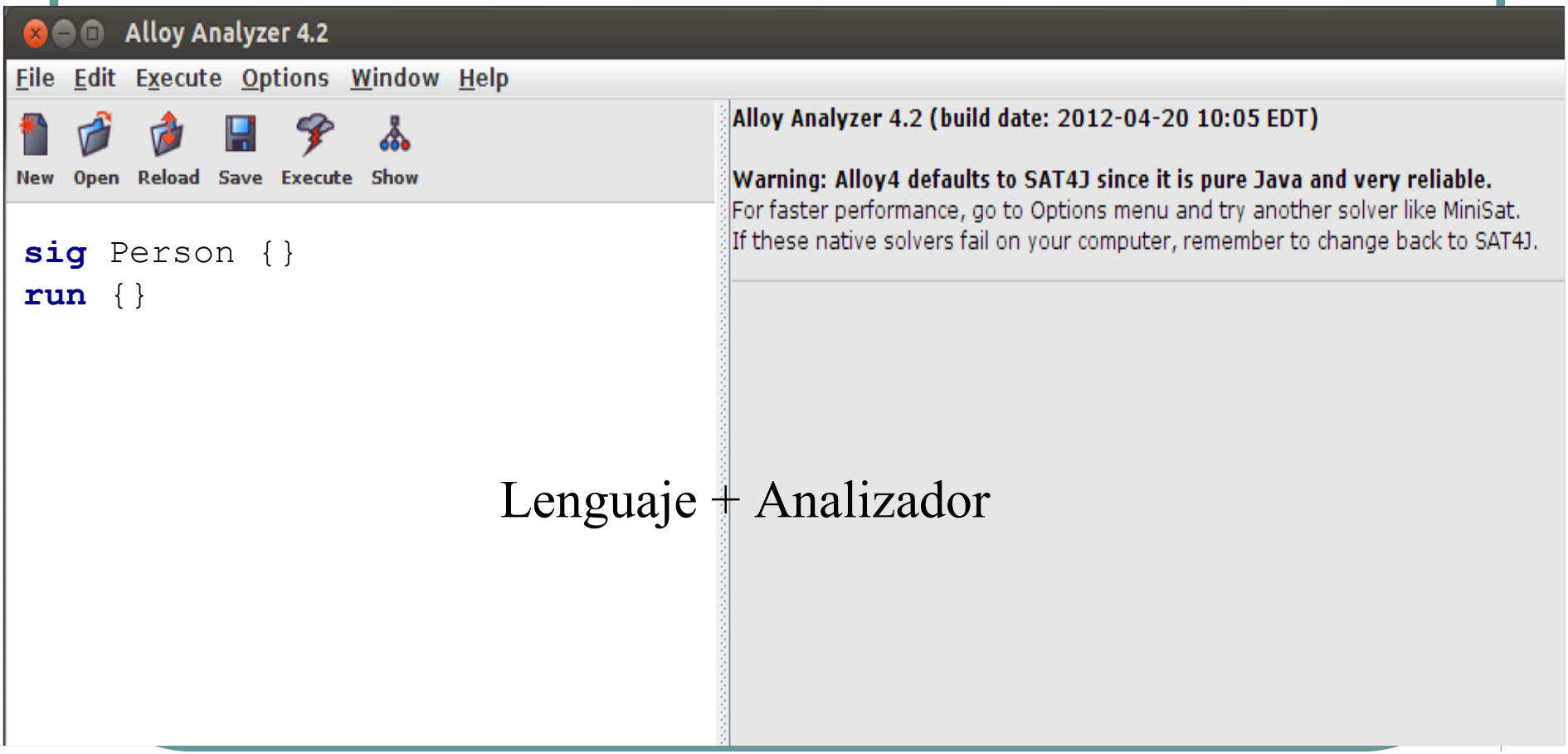
Alloy

alloy.mit.edu/alloy/download.html



Alloy

Herramienta



Lenguaje + Analizador

Agenda

- Introducción
- Alloy
- **La lógica de Alloy**
- Especificando con Alloy
- Verificación y Validación

La lógica de Alloy

Conceptos previos

Cada elemento se construye con **átomos** y **relaciones**.

Átomo: Entidad primitiva. Cumple 3 características:

- **Indivisible:** no puede ser dividido en partes mas pequeñas.
- **Immutable:** sus propiedades no cambian en el tiempo.
- **No interpretada:** por defecto ninguna propiedad asignada.

Relación:
Estructura que relaciona átomos, un cjto. de tuplas.

La lógica de Alloy

Conceptos previos

Ejemplo:

```
Person  = {(P0),(P1)}
Address = {(A0),(A1)}
address = {(P0,A1),(P1,A0)}
```

← Relación unaria

← Relación binaria

Atomo

Tamaño de la relación: número de tuplas.

Aridad de la relación: número de atomos en cada tupla.

La lógica de Alloy

Conceptos previos

Ejemplo:

```
Person  = {(P0),(P1)}
Address = {(A0),(A1)}
address = {(P0,A1),(P1,A0)}
```

```
sig Person {
    address : Address
}
sig Address {}
```

Person

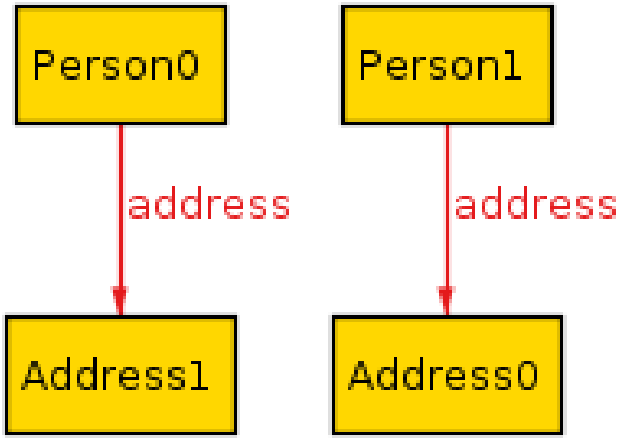
{Person$0, Person$1}

Address

{Address$0, Address$1}

address

{Person$0->Address$1, Person$1->Address$0}



La lógica de Alloy

Conceptos previos

Constantes especiales

none	*conjunto vacío*
univ	*conjunto universal*
iden	*relación identidad*

```
Person   = {(P0),(P1)}
Address  = {(A0),(A1)}

none = {}
univ = {(P0),(P1),(A0),(A1)}
iden = {(P0, P0), (P1, P1),
        (A0, A0), (A1, A1)}
```

La lógica de Alloy

Conceptos previos

Operadores de conjunto

`+`	*unión*
`&`	*intersección*
`-`	*diferencia*
`in`	*subconjunto*
`=`	*igualdad*

```
Person   = {(P0),(P1)}
Address  = {(A0),(A1)}

Person + Address = {(P0),(P1),(A0),(A1)}
Person & Address = {}
Person - (P0)    = {(P1)}
(P0) in Person   = true
Person = Address = false
```

La lógica de Alloy

Conceptos previos

Operadores relacionales

->	*producto*
<:	*restricción de dominio*
:>	*restricción de rango*
++	*sobre-escritura*
.	*composición*
[]	*composición*
~	*transpuesta*
^	*cláusura transitiva*
*	*cláusura reflexivo-transitiva*

La lógica de Alloy

Producto cartesiano (**->**)

```
Person   = {(P0),(P1)}
Address  = {(A0),(A1)}

Person -> Address =
{(P0,A0),(P0,A1),(P1,A0),(P1,A1)}
```

La lógica de Alloy

Restricción de dominio y rango

s **<:** *r* *restricción de dominio*
r **:>** s *restricción de rango*

- Son usados para filtrar las relaciones a un dominio o rango particular.
s: *un conjunto*
r: *una relación*

```
Person  = {(P0),(P1)}
address = {(P0,A1),(P1,A0)}
Address = {(A0),(A1)}

(P0)   <: address    = {(P0,A1)}
address :> A0        = {(P1,A0)}
```

La lógica de Alloy

Sobreescritura (++)

p ++ q

Funciona como la unión, pero las tuplas de "q" pueden reemplazar a tuplas de "p". Del resultado se elimina toda tupla en "p" que concuerde con una tupla en "q"; es decir, que comience con el mismo nombre.

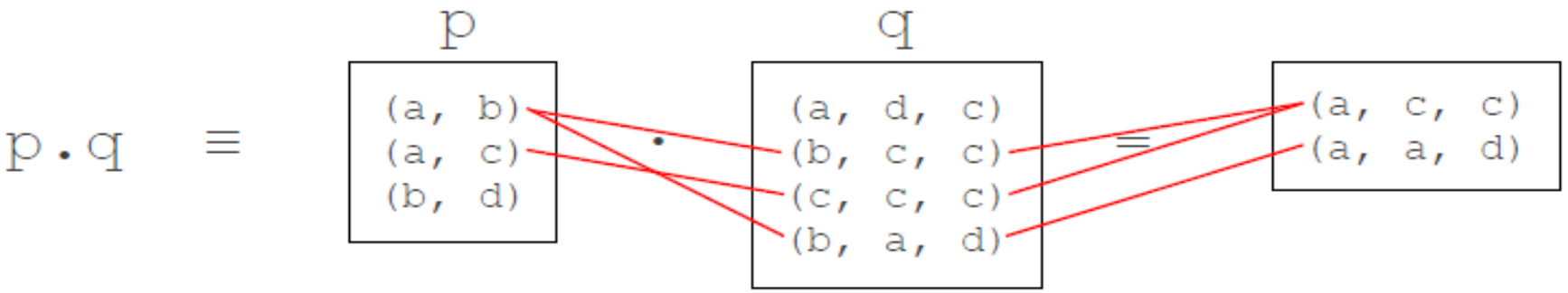
p: *una relación*
q: *una relación*

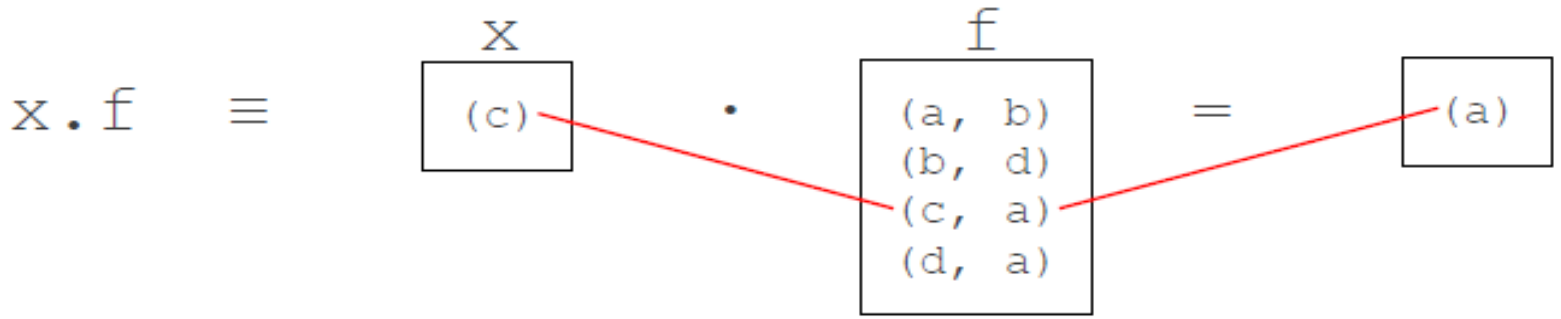
```
address = {(P0, A1), (P1, A0)}
workAddress = {(P0, A0)}

address ++ workAddress = {(P0, A0), (P1, A0)}
```

La lógica de Alloy

Composición (.)





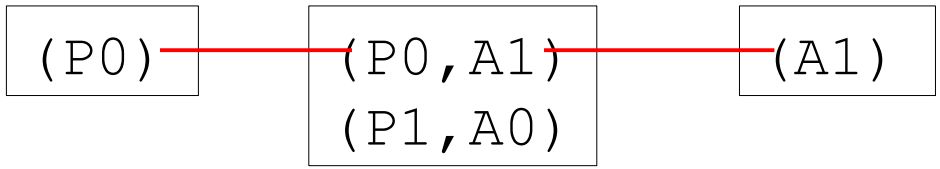
La lógica de Alloy

Composición (.) →

- join relacional
- campo de navegación
- aplicación de función

```
Person  = {(P0),(P1)}
address = {(P0,A1),(P1,A0)}
```

(P0) . Address = (A1)



La lógica de Alloy

Composición ([ ])

```
e1[e2] = e2.e1
a.b.c[d] = d.(a.b.c)
```

La lógica de Alloy

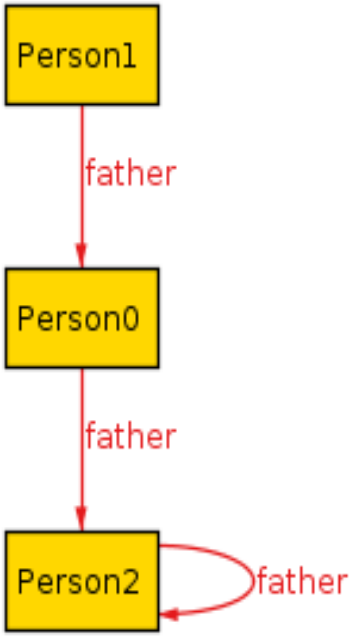
```
^r = r + r.r + r.r.r + …
*r = iden + ^r
```

~	*transpuesta*
^	*cláusura transitiva*
*	*cláusura reflexo transitiva*

Nota: Se aplican sólo a relaciones binarias

```
Person = {(P0), (P1), (P2)}
father = {(P0,P2),(P1,P0),(P2,P2)}

~father = {(P2,P0),(P0,P1),(P2,P2)}
^father = {(P0,P2),(P1,P0),(P2,P2),(P1,P2)}
*father = {(P0,P2),(P1,P0),(P2,P2),(P1,P2),(P0,P0),
(P1,P1)}
```

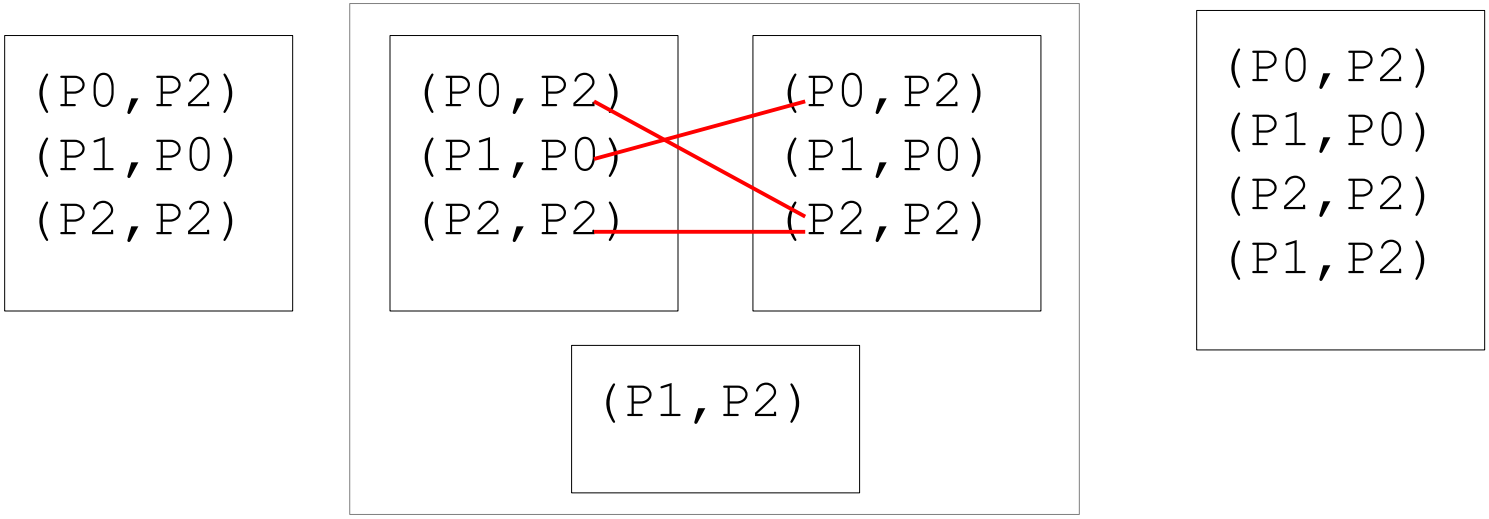


La lógica de Alloy

Cláusura transitiva (^)

```
father  = {(P0,P2),(P1,P0),(P2,P2)}
^father = father + father.father + ...
```

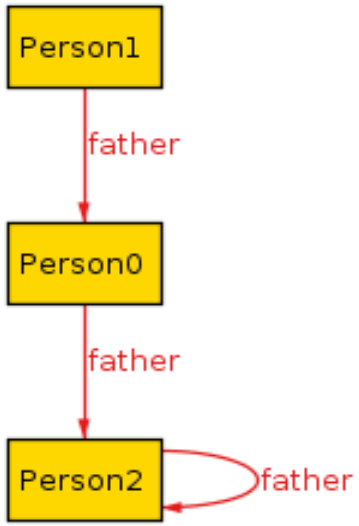
```
^father = father  +  father . father  =
```



La lógica de Alloy

Cláusura reflexo - transitiva (*)

```
father  = {(P0,P2),(P1,P0),(P2,P2)}
iden    = {(P0,P0),(P1,P1),(P2,P2)}
^father = {(P0,P2),(P1,P0),(P2,P2),(P1,P2)}
*father = iden + ^father
```



*father = **iden** + ^father =

iden	^father	
(P0,P0)	(P0,P2)	(P0,P2)
(P1,P1)	(P1,P0)	(P1,P0)
(P2,P2)	(P2,P2)	(P2,P2)
	(P1,P2)	(P1,P2)
		(P0,P0)
		(P1,P1)

La lógica de Alloy

Conceptos previos

Operadores lógicos

!	**not**	*negación*
&&	**and**	*conjunción*
||	**or**	*disjunción*
=>	**implies**	*implicación*
<=>	**iff**	*bi-implicación*

La lógica de Alloy

Conceptos previos

Cuantificadores

all	*para todo*
some	*al menos uno*
no	*ninguno*
lone	*a lo sumo uno*
one	*exactamente uno*

```
Person = {(P0), (P1), (P2)}

some p: Person | p in Person      = true
one Person                        = false
```

La lógica de Alloy

Conceptos previos

Operador de comparación

#	*cardinalidad*
=	*igual*
<	*menor*
>	*mayor*
=<	*menor o igual*
>=	*mayor o igual*

```
Person  = {(P0),(P1),(P2)}
father  = {(P0,P2),(P1,P0),(P2,P2)}

#Person = 3
#father = 3
#Person < #father = false
```

Agenda

- Introducción
- Alloy
- La lógica de Alloy
- **Especificando con Alloy**
- Verificación y Validación

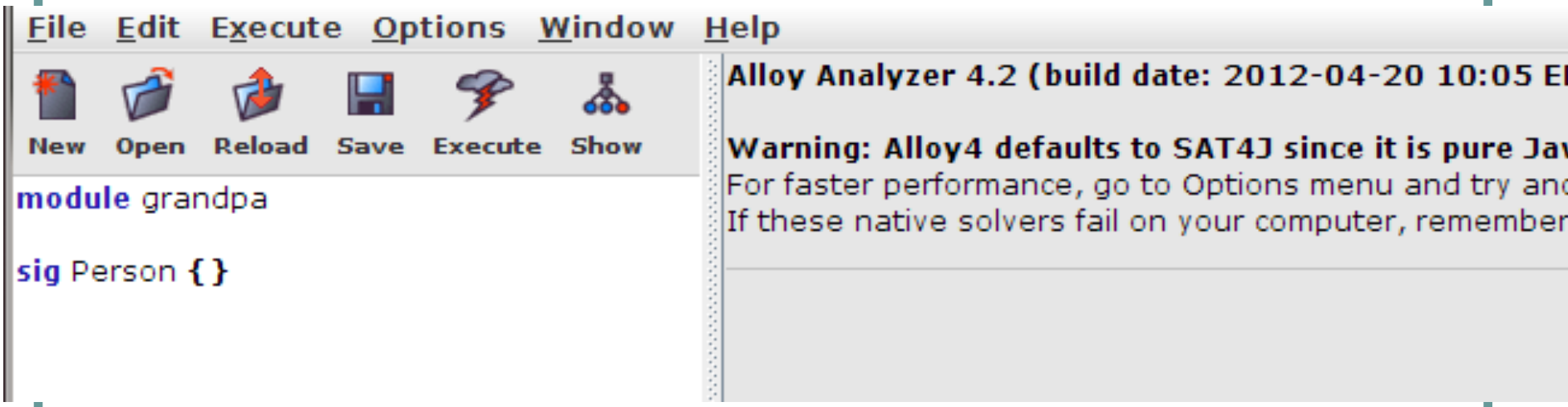
Especificando con Alloy

Las especificaciones típicamente se componen de:

- Módulo
- Signaturas y Campos
- Hechos
- Predicados y Funciones
- Comando “run”
- Aserciones y Comando “check”

Especificando con Alloy

Módulo y Signaturas



Multiplicidad

```
some sig A {}
lone sig B {}
one  sig C {}
```

Especificando con Alloy

Signaturas

```
module grandpa

abstract sig Person {}
sig Man extends Person {}
sig Woman extends Person {}
```

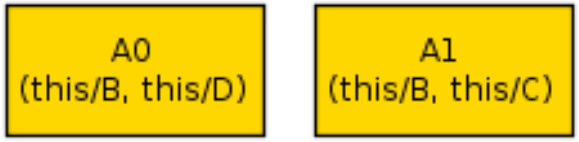
```
Person
   {Man$0, Woman$0, Woman$1}
Man
   {Man$0}
Woman
   {Woman$0, Woman$1}
```

Especificando con Alloy

```
sig A{}
sig B in A{}
sig C in A{}
//B es subcjto. de A, C es subcjto. de A, pero no necesariament
  Éstos son disjuntos.

sig D in B + C {}
//D es un subconjunto de la unión de B y C
```

```
A
  {A$0, A$1}
B
  {A$0, A$1}
C
  {A$1}
D
  {A$0}
```



Especificando con Alloy

Campos

```
sig A {f: m e}
sig A {f: e1 m -> n e2}
```

set	*cualquier número*
some	*al menos uno*
lone	*a lo sumo uno*
one	*exactamente uno*

```
abstract sig Person {
    father: lone Man,
    mother: lone Woman
}
sig Man extends Person {}
sig Woman extends Person {}
```

Especificando con Alloy

Campos

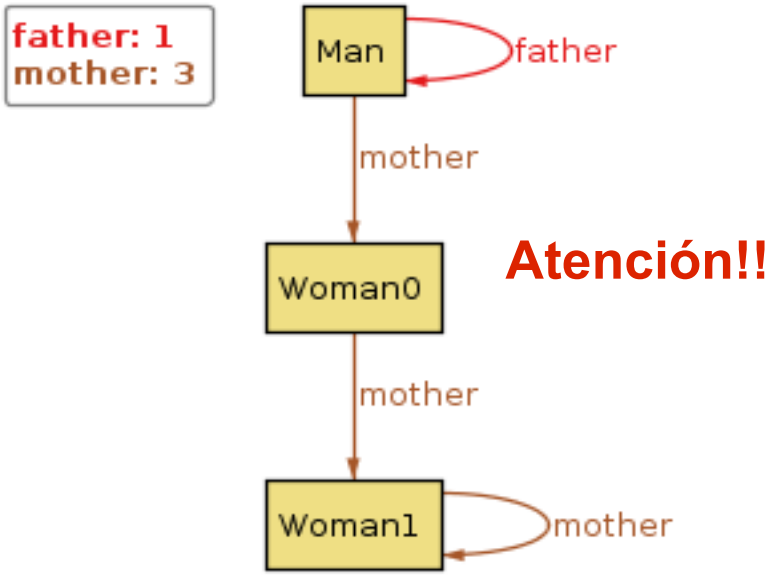
```
abstract sig Person {
    father: lone Man,
    mother: lone Woman
}
sig Man extends Person {}
sig Woman extends Person {}
```

father

{Man$0->Man$0}

mother

{Man$0->Woman$0, Woman$0->Woman$1, Woman$1->Woman$1}



Atención!!

Especificando con Alloy

Hechos

```
fact { F }
fact f { F }
sig S { ... }{ F }
```

Los hechos introducen restricciones que se asumen siempre deben cumplirse.

```
abstract sig Person {
    father: lone Man,
    mother: lone Woman
}
sig Man extends Person {}
sig Woman extends Person {}

fact { no p: Person | p in p.^(mother + father) }
```

Especificando con Alloy

Predicados y Funciones

```
pred p[x1: e1, ..., xn: en] { F }
```

```
fun f[x1: e1, ..., xn: en] : e { E }
```

```
abstract sig Person {
    father: lone Man,
    mother: lone Woman}
sig Man extends Person {}
sig Woman extends Person {}
fact { no p: Person | p in p.^(mother + father) }

fun parents [p: Person]: set Person {
    p.(mother + father)
}
pred brothers [p: Person] {
    some b: Man | (parents[p] = parents [b])
}
```

Agenda

- Introducción
- Alloy
- La lógica de Alloy
- Especificando con Alloy
- **Verificación y Validación**

Verificación y Validación

Las especificaciones típicamente se componen de:

- Módulo
- Signaturas y Campos
- Hechos
- Predicados y Funciones
- Comando “run”
- Aserciones y Comando “check”

Verificación y Validación

Comando **run**

```
pred p[x: X, y: Y, ..] {F}
run p alcance
```

```
fun f[x: X, y: Y, ..]:R {E}
run f alcance
```

Instruye al analizador a buscar instancias con un alcance

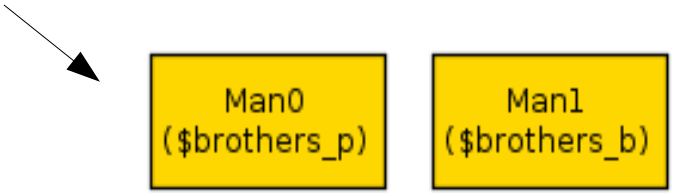
```
abstract sig Person {
    father: lone Man, mother: lone Woman}
sig Man extends Person {}
sig Woman extends Person {}
fact { no p: Person | p in p.^(mother + father) }
fun parents [p: Person]: set Person {
    p.(mother + father)
}
pred brothers [p: Person] {
    some b: Man | (parents[p] = parents [b])
}
run brothers for 4 Person
```

Atención!!

Man
($brothers_b, $brothers_p)

Verificación y Validación

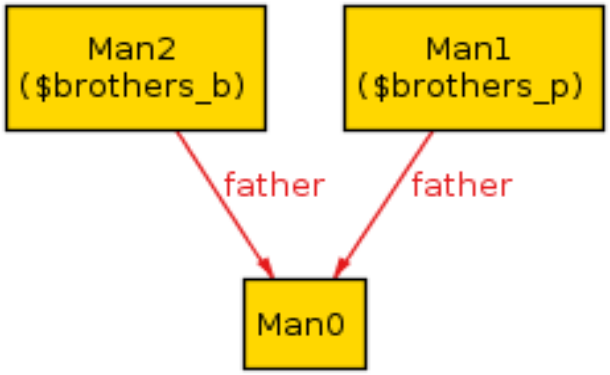
```
abstract sig Person {
    father: lone Man, mother: lone Woman}
sig Man extends Person {}
sig Woman extends Person {}
fact { no p: Person | p in p.^(mother + father) }
fun parents [p: Person]: set Person {
    p.(mother + father)
}
pred brothers [p: Person] {
 some b: Man |
   (not b = p) and (parents[p] = parents [b])
}
run brothers for 4 Person
```



Atención!!

Verificación y Validación

```
abstract sig Person {
    father: lone Man, mother: lone Woman}
sig Man extends Person {}
sig Woman extends Person {}
fact { no p: Person | p in p.^(mother + father) }
fun parents [p: Person]: set Person {
    p.(mother + father)
}
pred brothers [p: Person] {
some b: Man |
  (not b = p) and (some parents[p]) and (parents[p] = parents[b])
}
run brothers for 4 Person
```



Verificación y Validación

Comando **run**

```
abstract sig Person {
    father: lone Man, mother: lone Woman}
sig Man extends Person {}
sig Woman extends Person {}
fact { no p: Person | p in p.^(mother + father) }
fun parents [p: Person]: set Person {
    p.(mother + father)
}
pred brothers [p: Person] {
some b: Man |
 (not b = p) and (some parents[p]) and (parents[p] = parents[b])
}
run brothers for 4 but 2 Man, exactly 2 Woman
```

Verificación y Validación

Aserciones y Comando **check**

```
assert a { F }
check a alcance
```

Las aserciones son restricciones son consecuencia de los hechos del modelo.

*El comando **check** instruye al analizador para que encuentre un contraejemplo.*

```
abstract sig Person {
    father: lone Man,
    mother: lone Woman }
sig Man extends Person {}
sig Woman extends Person {}
fact { no p: Person | p in p.^(mother + father) }

assert noSelfFather {
 no m: Man | m = m.father
}
check noSelfFather
```

Bibliografía

- Jackson, D.: Software Abstractions: Logic, Language, and Analysis, revised edn. MIT Press (2012)

- Dennis G. and Seater R.: Alloy Analyzer 4 Tutorial. Session 1: Intro and Logic, Software Design Group, MIT, disponible en: http://alloy.mit.edu/alloy/tutorials/day-course/s1_logic.pdf (última visita – agosto de 2014)

- Jackson, D.: Alloy & Applications, MIT (2009)