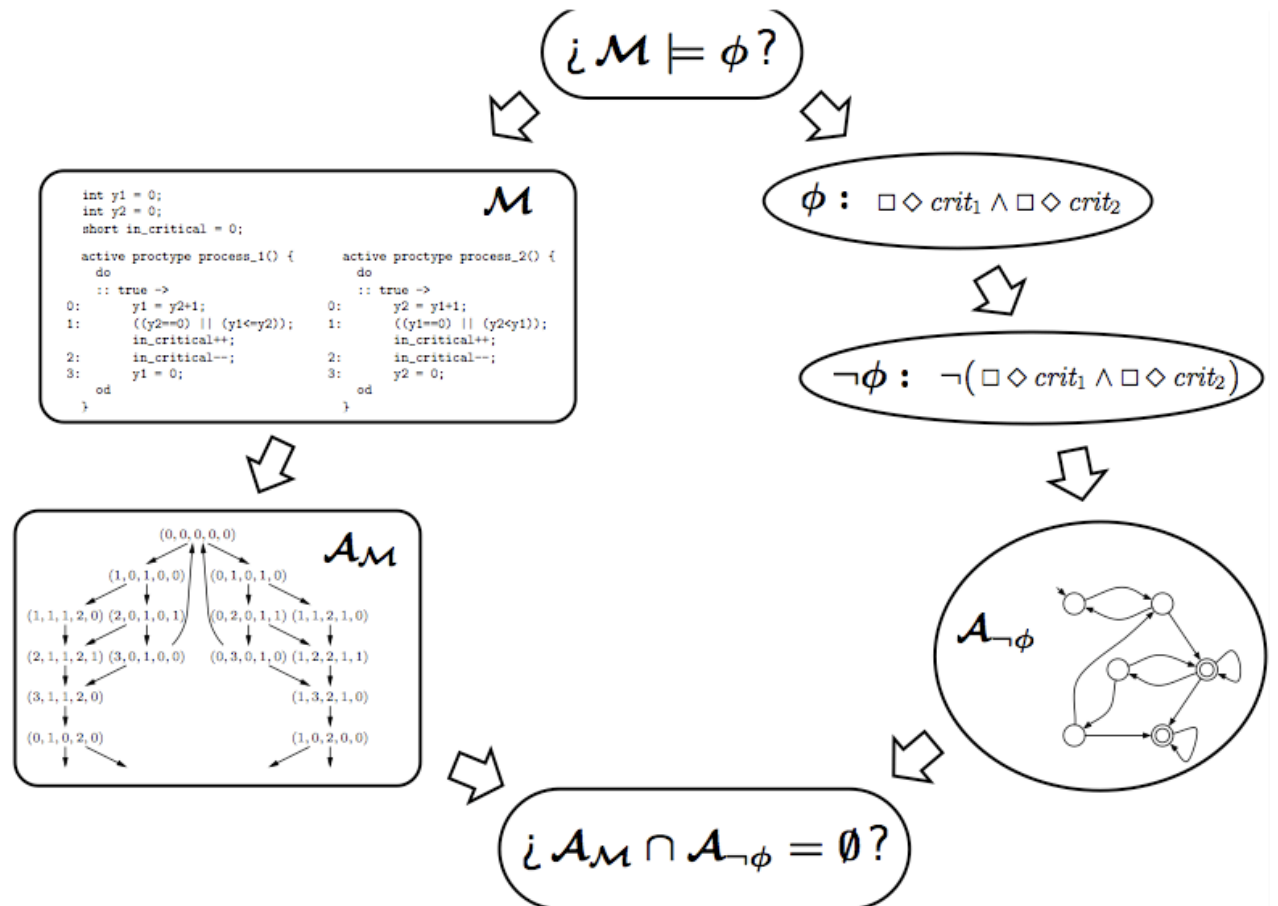


Validación y Verificación de Software

Uso del Lenguaje Alloy para la especificación de software

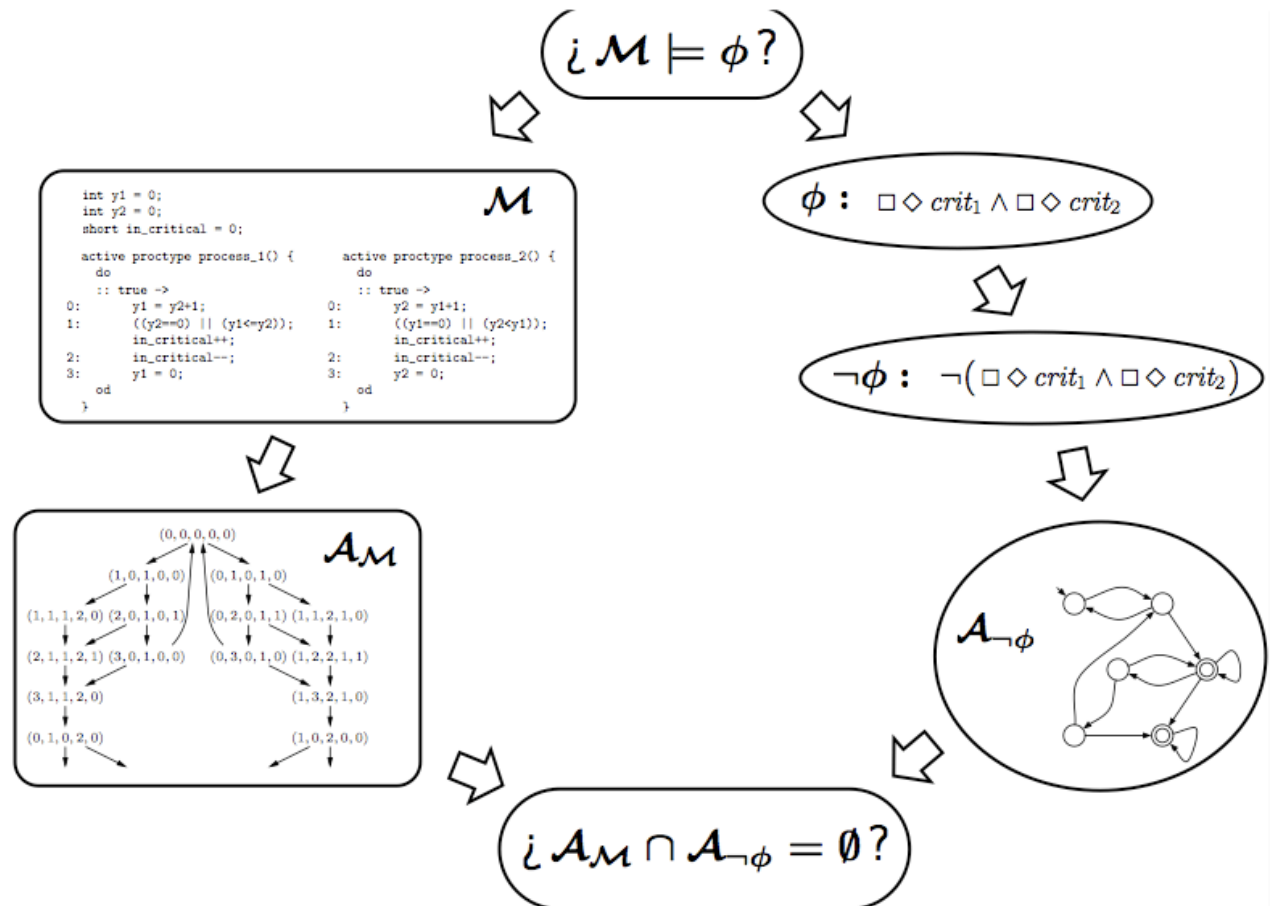
Escenas de Capítulos
Anteriores...

Escenas de Capítulos Anteriores...



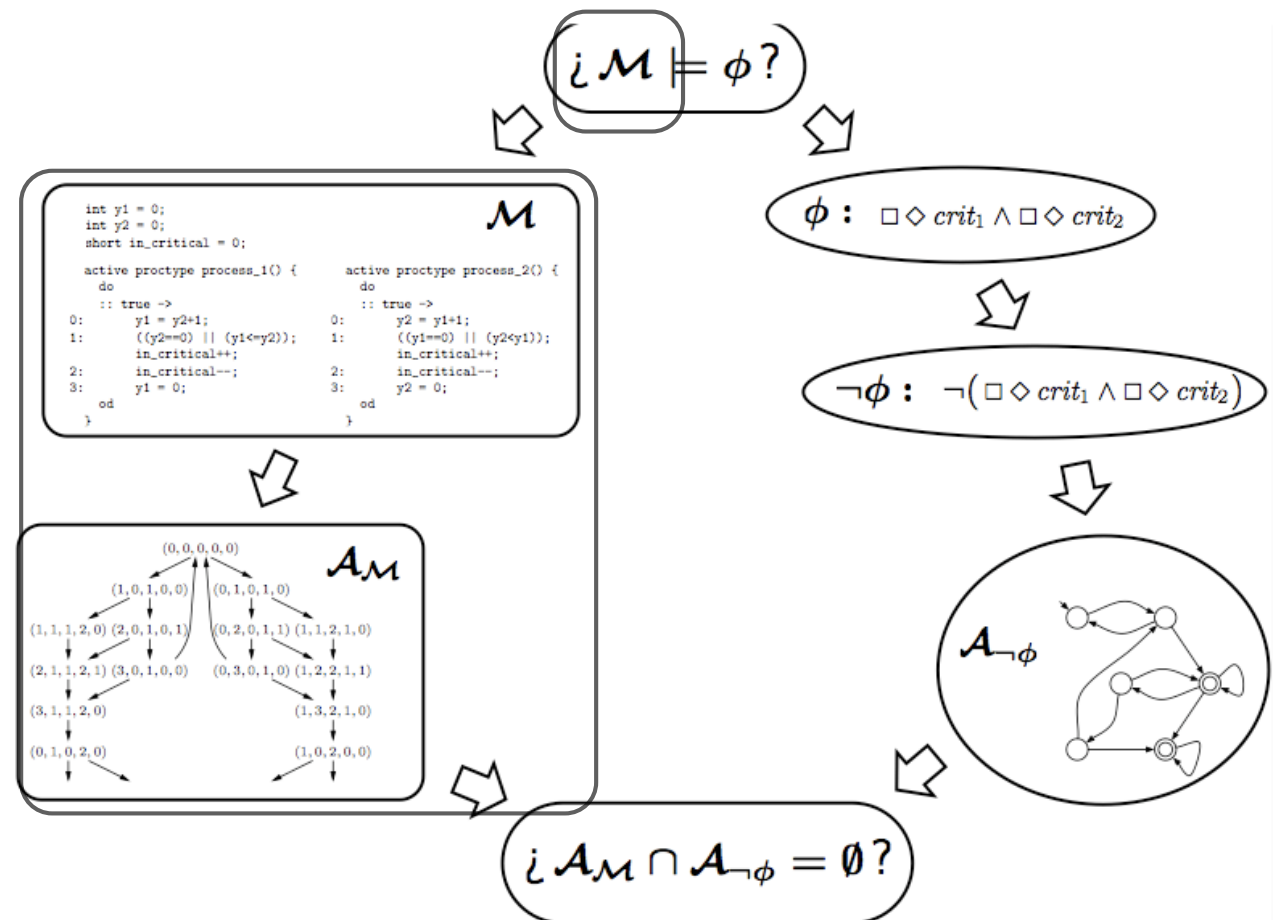
Escenas de Capítulos Anteriores...

Descripciones formales abstractas de ejecuciones de sistemas concurrentes
(álgebras de procesos: FSP, SPIN)



Escenas de Capítulos Anteriores...

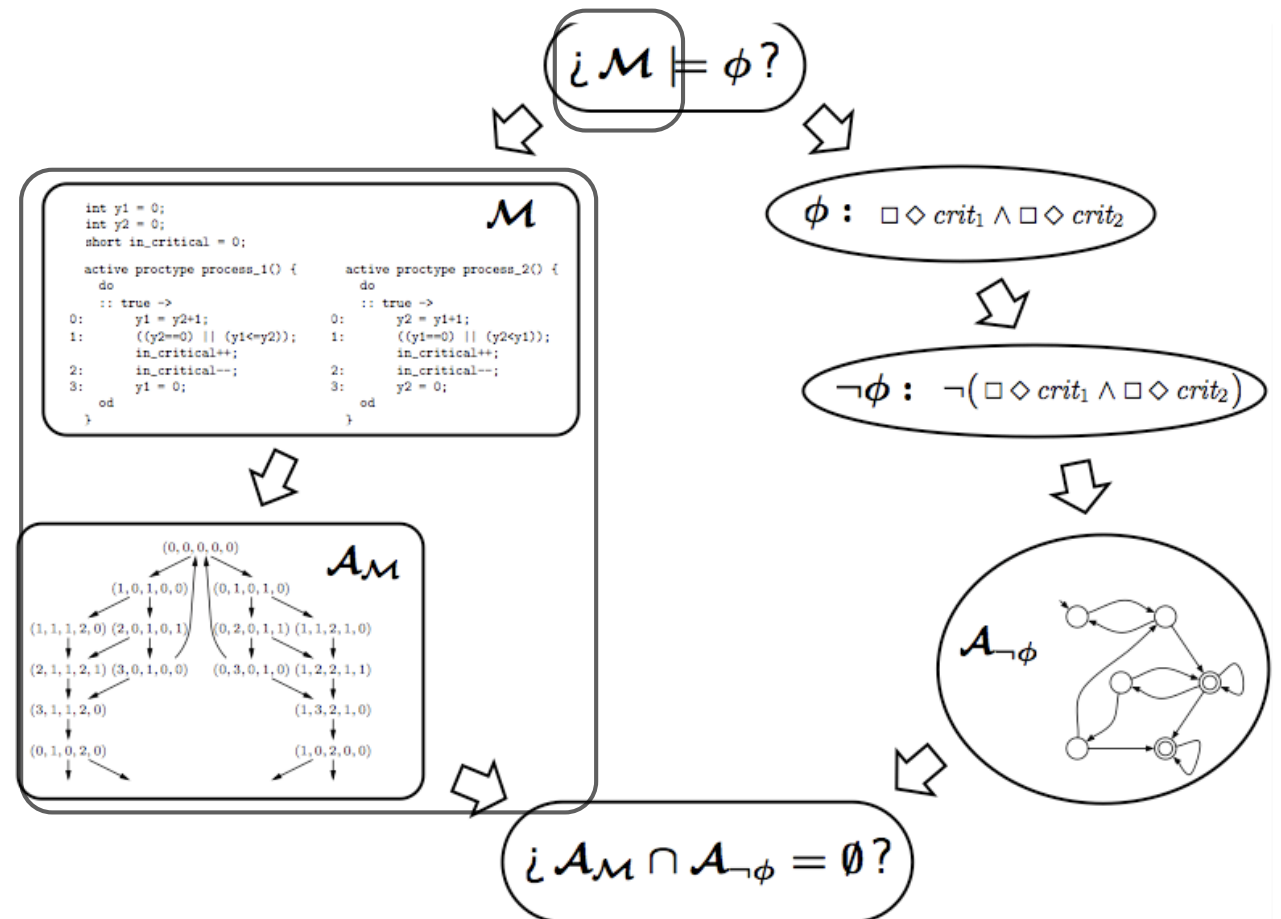
Descripciones formales abstractas de ejecuciones de sistemas concurrentes
(álgebras de procesos: FSP, SPIN)



Escenas de Capítulos Anteriores...

Descripciones formales abstractas de ejecuciones de sistemas concurrentes
(álgebras de procesos: FSP, SPIN)

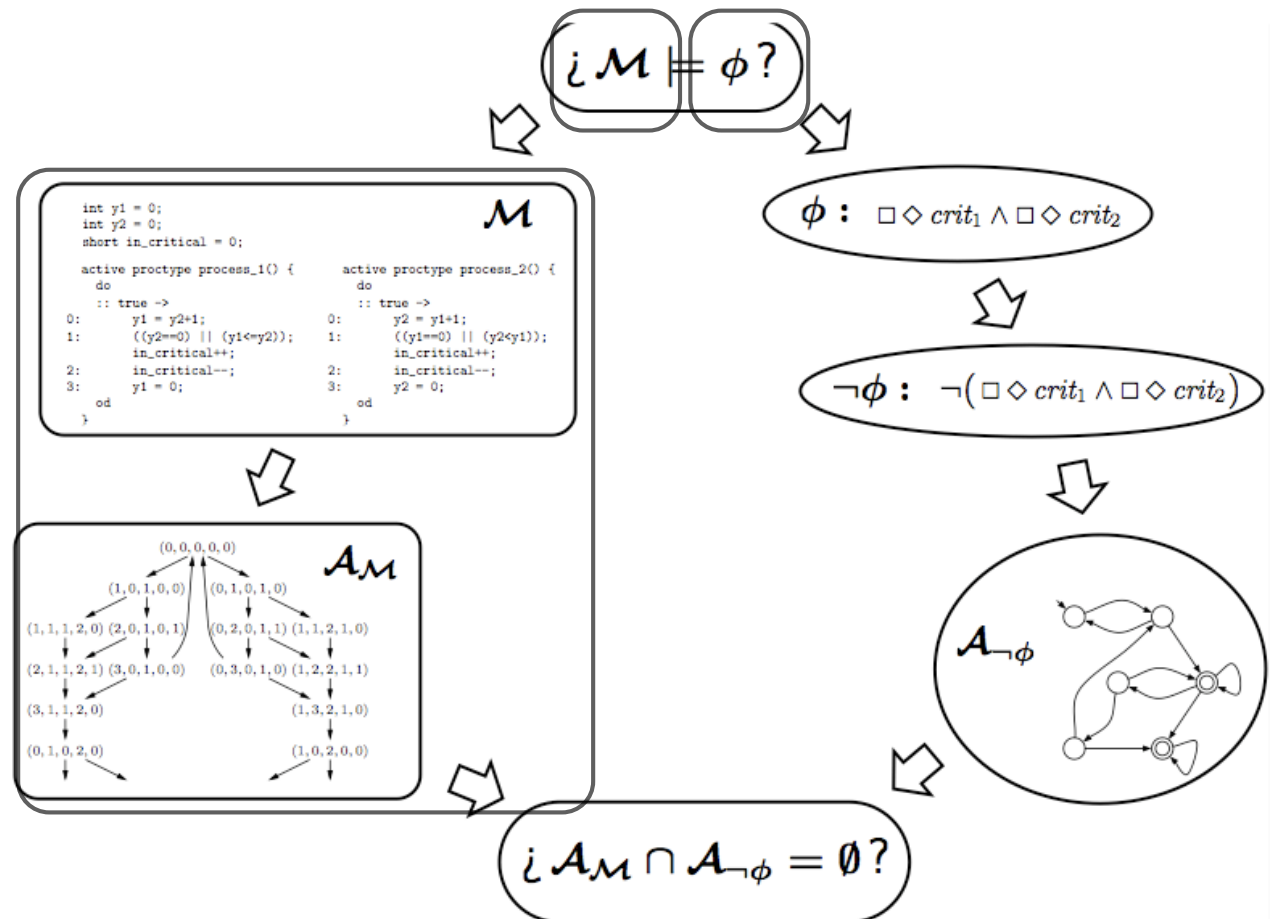
- Descripciones de propiedades de ejecuciones de sistemas
(lógica temporal)



Escenas de Capítulos Anteriores...

Descripciones formales abstractas de ejecuciones de sistemas concurrentes
(álgebras de procesos: FSP, SPIN)

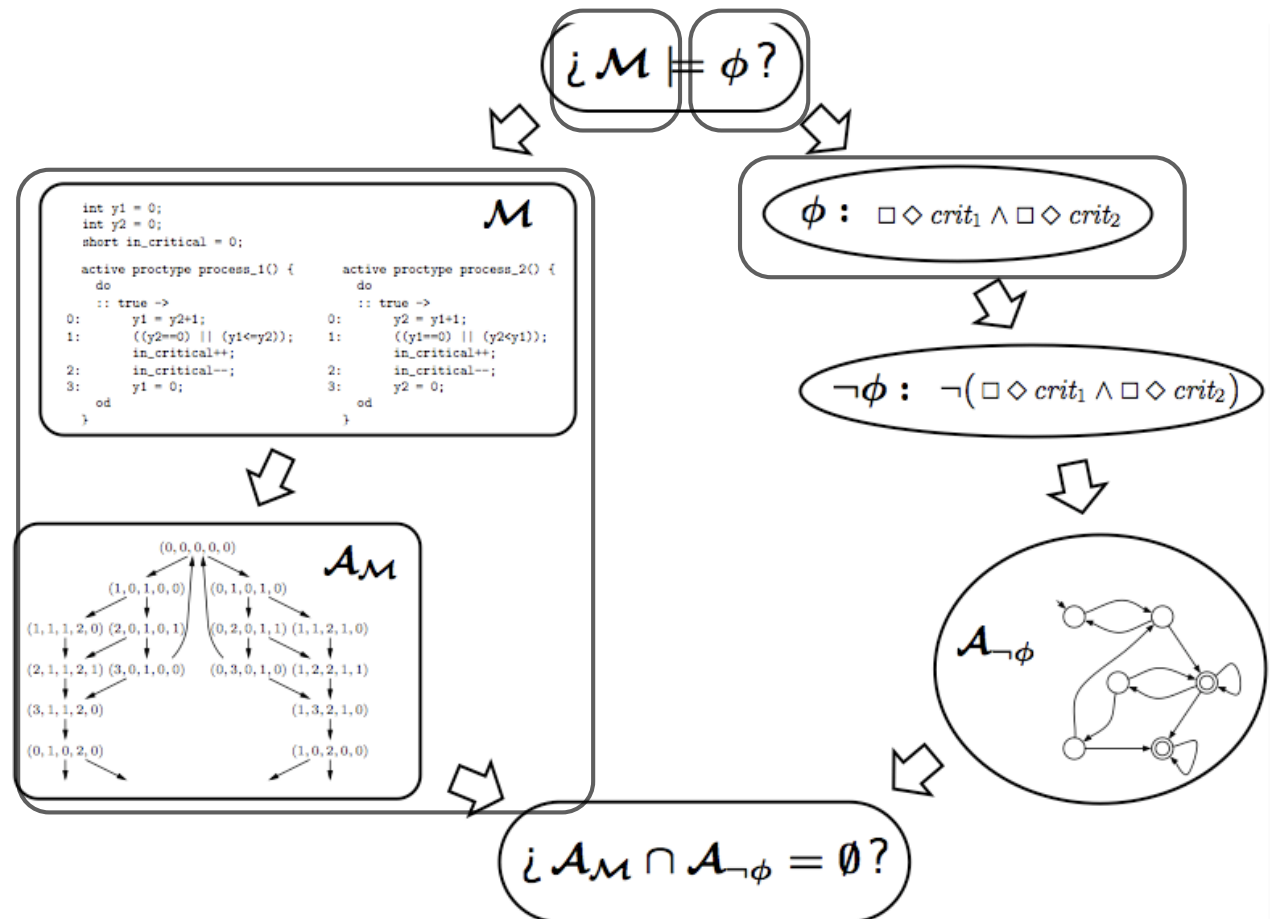
- Descripciones de propiedades de ejecuciones de sistemas
(lógica temporal)



Escenas de Capítulos Anteriores...

Descripciones formales abstractas de ejecuciones de sistemas concurrentes
(álgebras de procesos: FSP, SPIN)

- Descripciones de propiedades de ejecuciones de sistemas (lógica temporal)

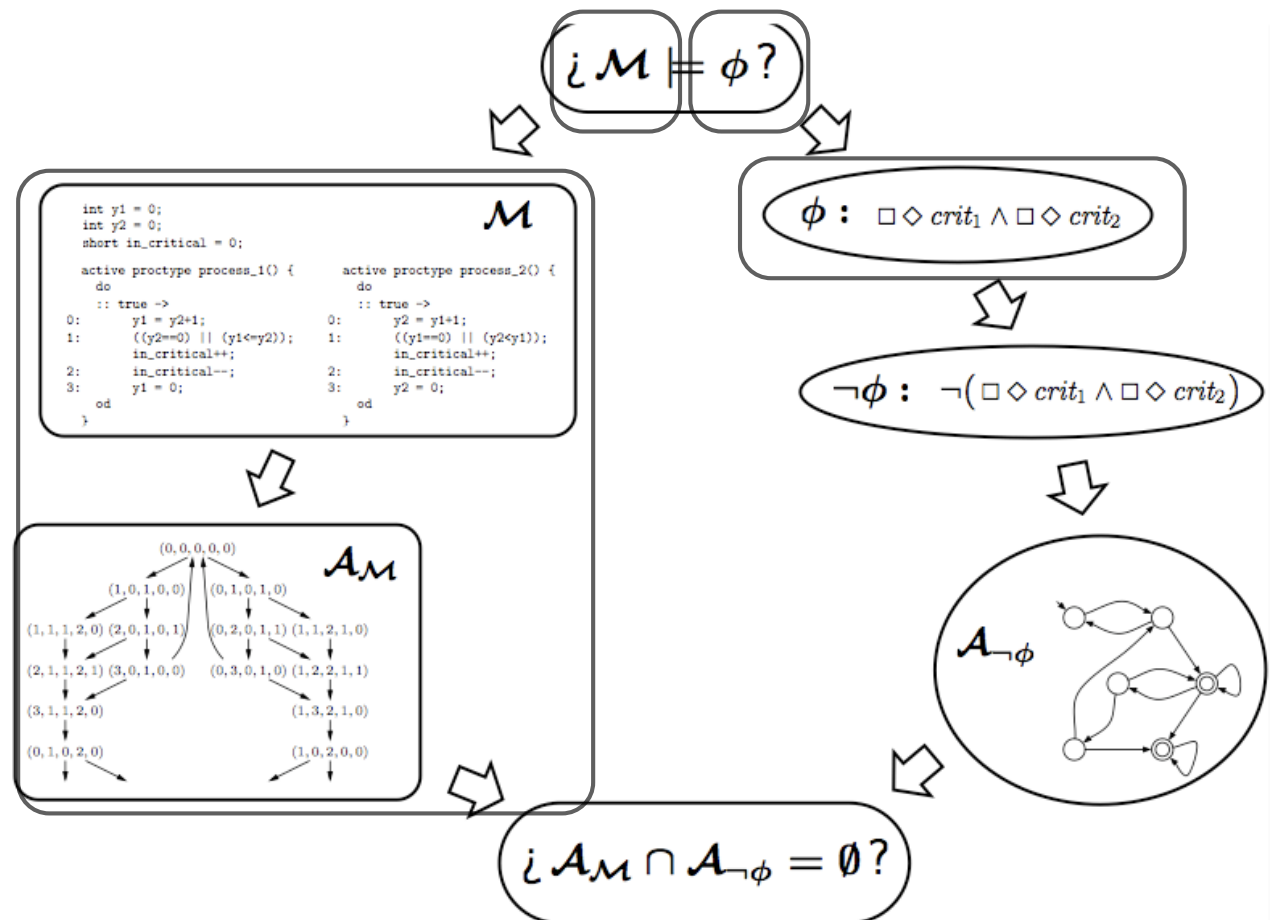


Escenas de Capítulos Anteriores...

Descripciones formales abstractas de ejecuciones de sistemas concurrentes
(álgebras de procesos: FSP, SPIN)

- Descripciones de propiedades de ejecuciones de sistemas (lógica temporal)

- Mecanismos de análisis automático de propiedades de sistemas (model checking)

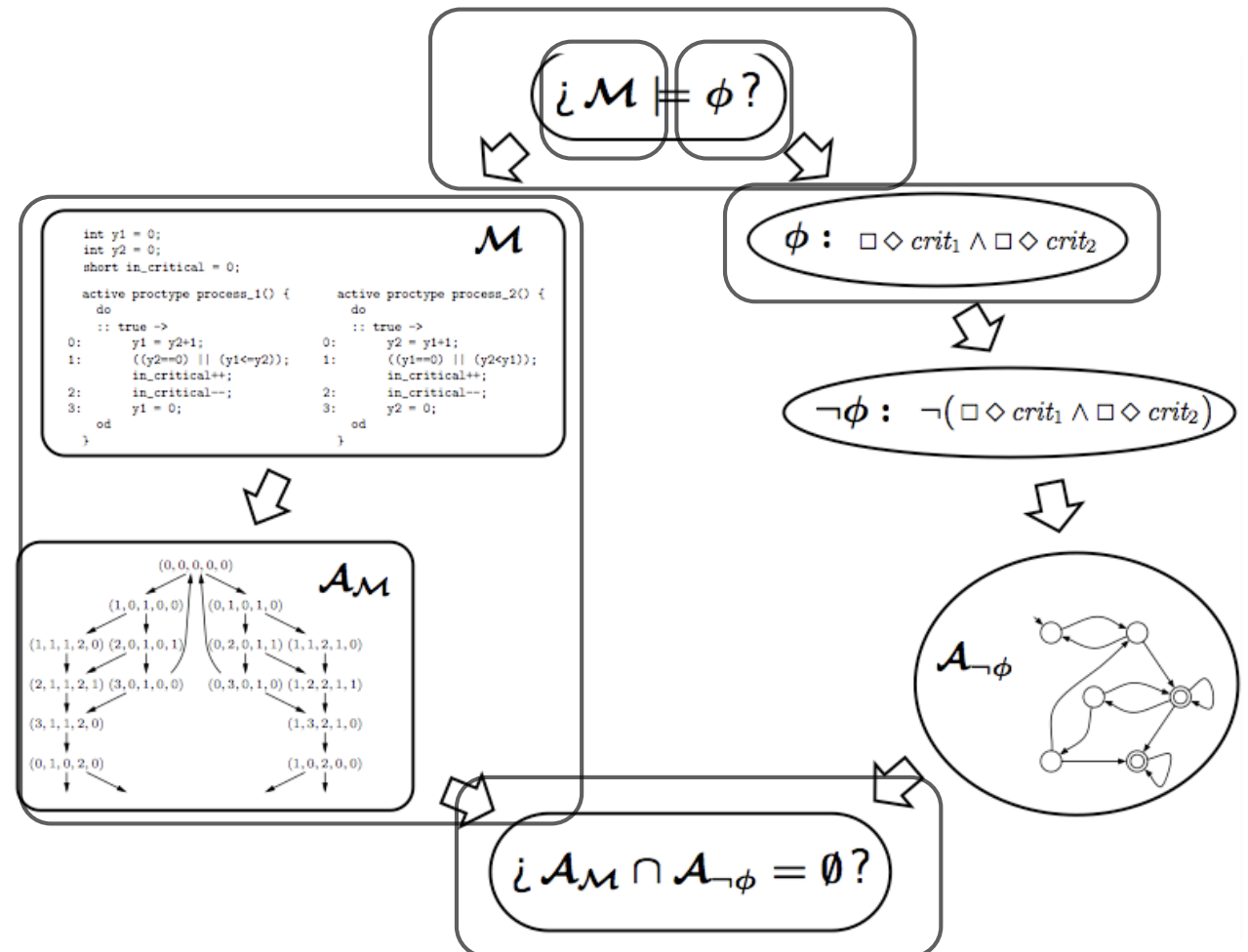


Escenas de Capítulos Anteriores...

Descripciones formales abstractas de ejecuciones de sistemas concurrentes
(álgebras de procesos: FSP, SPIN)

- Descripciones de propiedades de ejecuciones de sistemas (lógica temporal)

- Mecanismos de análisis automático de propiedades de sistemas (model checking)



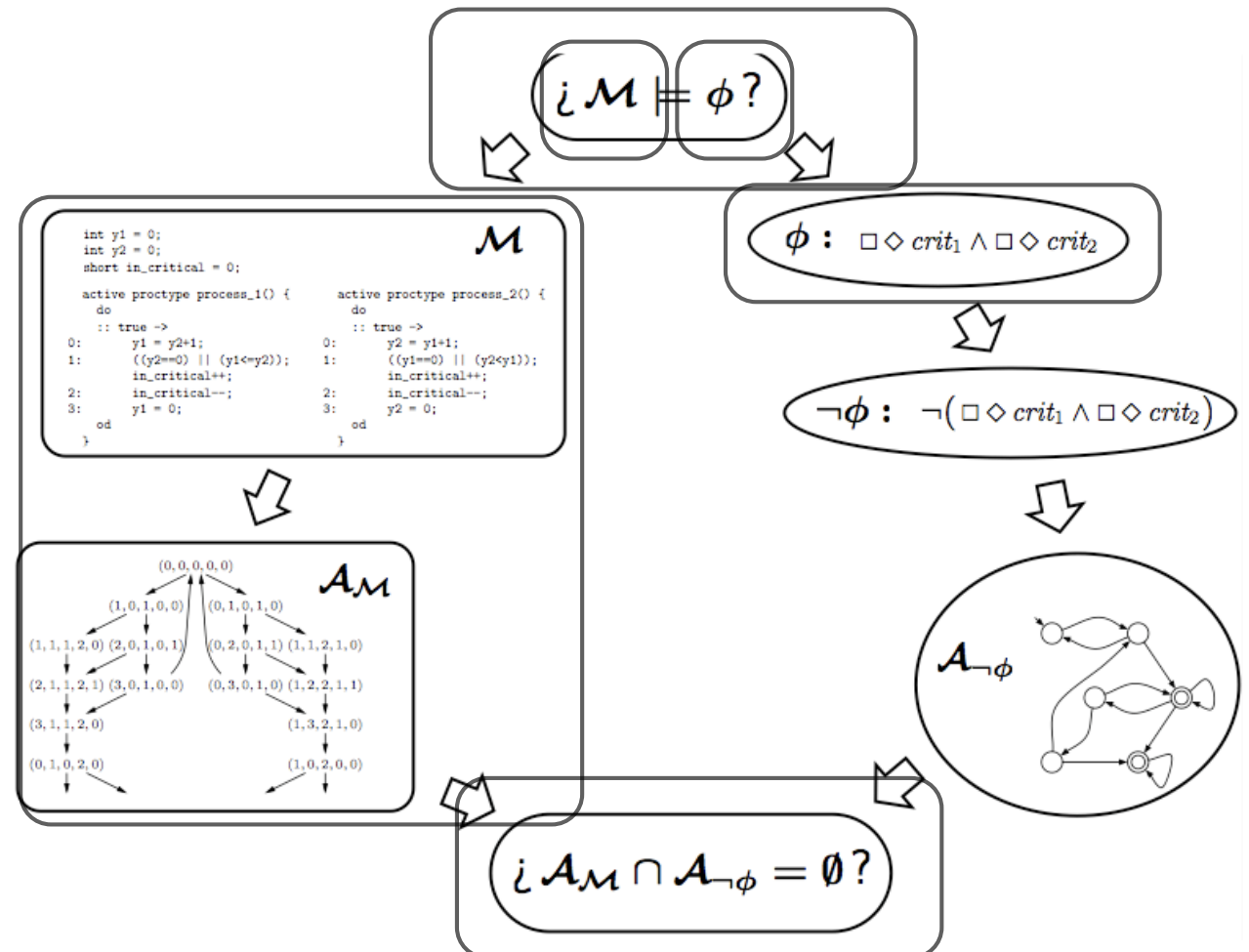
Escenas de Capítulos Anteriores...

Descripciones formales abstractas de ejecuciones de sistemas concurrentes
(álgebras de procesos: FSP, SPIN)

- Descripciones de propiedades de ejecuciones de sistemas (lógica temporal)

- Mecanismos de análisis automático de propiedades de sistemas (model checking)

- Variedad de lenguajes de especificaciones del comportamiento operacional de sistemas (FSP, SPIN, etc)



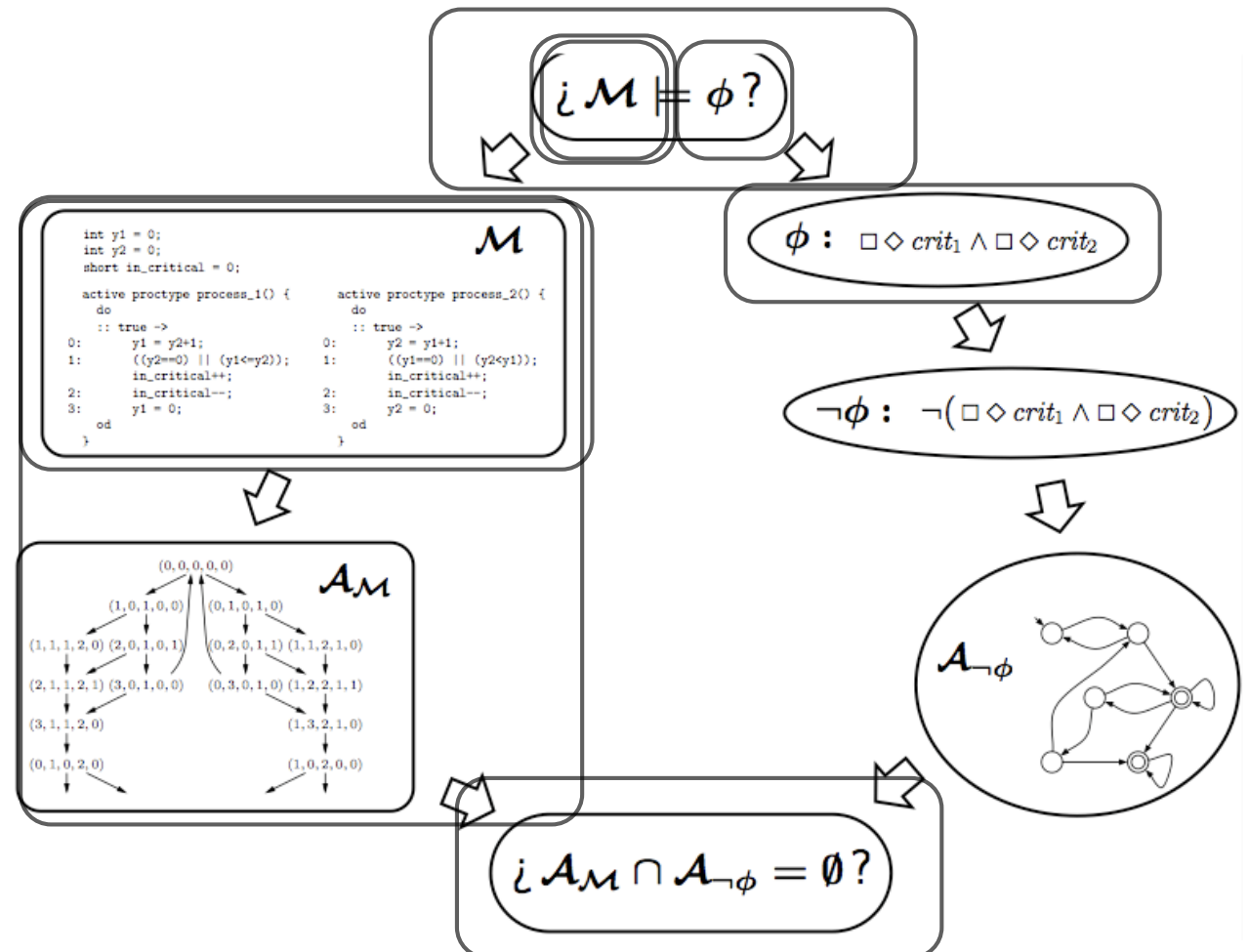
Escenas de Capítulos Anteriores...

Descripciones formales abstractas de ejecuciones de sistemas concurrentes
(álgebras de procesos: FSP, SPIN)

- Descripciones de propiedades de ejecuciones de sistemas (lógica temporal)

- Mecanismos de análisis automático de propiedades de sistemas (model checking)

- Variedad de lenguajes de especificaciones del comportamiento operacional de sistemas (FSP, SPIN, etc)



Especificaciones de Diseños de Software

Especificaciones de Diseños de Software

Al describir diseños de software, en general nuestras descripciones (informales) no tienen una naturaleza operacional

Especificaciones de Diseños de Software

Al describir diseños de software, en general nuestras descripciones (informales) no tienen una naturaleza operacional

Estas especificaciones suelen estar compuestas por abstracciones de datos y capacidades asociadas con estas abstracciones

Especificaciones de Diseños de Software

Al describir diseños de software, en general nuestras descripciones (informales) no tienen una naturaleza operacional

Estas especificaciones suelen estar compuestas por abstracciones de datos y capacidades asociadas con estas abstracciones

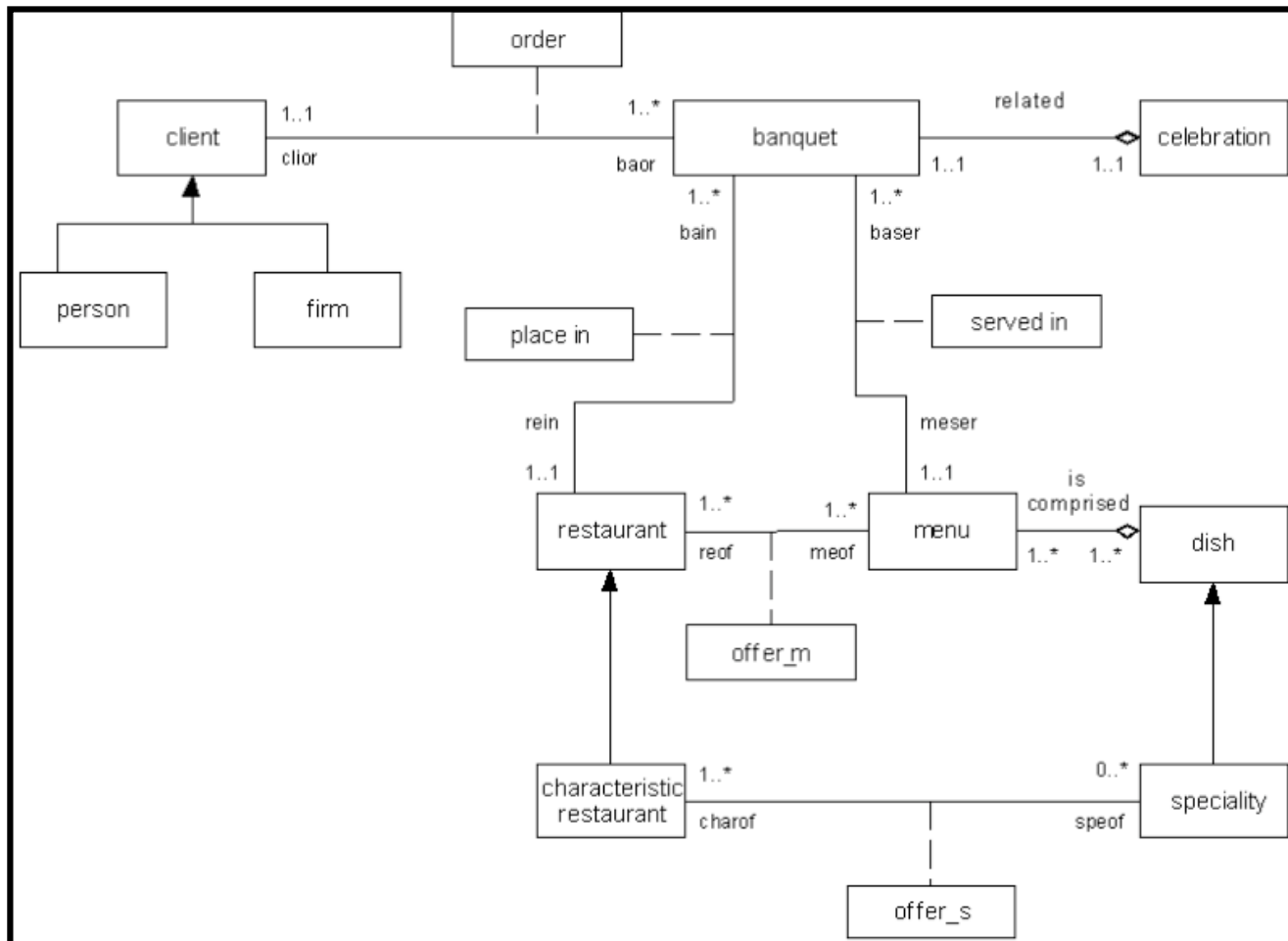
Ej.: diagramas de clases

Especificaciones de Diseños de Software

Consideremos, por ejemplo, la siguiente descripción informal de un diseño:

Especificaciones de Diseños de Software

Consideremos, por ejemplo, la siguiente descripción informal de un diseño:



La Importancia del Modelado/ Diseño de Software

La complejidad asociada al desarrollo de software nos ha llevado a observar que:

Concentrar las metodologías de desarrollo en etapas de implementación es en muchos casos contraproducente (e.g., pueden llevar a descubrir errores en los requisitos cuando ya se ha invertido mucho esfuerzo en su implementación).

Es importante analizar el problema hasta tener una buena comprensión del mismo.

Es importante contar con un plan para el desarrollo de una solución al problema, antes de comenzar a implementar la solución.

Los últimos dos puntos corresponden a las etapas de análisis y diseño. Para llevar adelante las tareas de análisis y diseño necesitamos poder describir, o modelar, diferentes aspectos del problema y de su solución.

El Modelado de Software

Modelar es describir algún aspecto del problema a resolver o de su solución. Esta actividad en general demanda:

- la interacción entre varios desarrolladores (e.g., discutiendo ideas, decisiones de diseño, etc)

- la interacción entre desarrolladores y clientes (e.g., para validar requisitos, etc)

Los lenguajes informales permiten esencialmente sólo la comunicación/documentación de modelos. Los lenguajes formales, en cambio, son propicios para la aplicación de diferentes tareas de análisis (muchas automatizables), PERO requieren semántica formal (y en general la manipulación de expresiones matemáticas o lógicas)

El Modelado de Software

Varios enfoques tradicionales para la construcción formal y la verificación de sistemas se basan en el uso de especificaciones. Una especificación puede usarse para varias tareas, tales como:

descubrir inconsistencias (por ejemplo, requerimientos contradictorios) o falta de detalles en las funcionalidades requeridas del sistema a construir,

análisis de propiedades del sistema a construir antes de comenzar con su construcción,

verificación del sistema construido con respecto a su especificación, construcción rigurosa de implementaciones a partir de especificaciones.

Características Deseables de Lenguajes Formales de especificaciones

si el lenguaje de especificaciones es **formal**, éste no da lugar a ambigüedades y, por lo tanto, no requiere esfuerzos adicionales para evitar confusiones e interpretaciones erróneas de las especificaciones,

si el lenguaje de especificaciones es **declarativo**, éste facilita la caracterización tanto del sistema a construir como de las propiedades a analizar,

Características Deseables de Lenguajes Formales de especificaciones

si el lenguaje de especificaciones es **suficientemente expresivo**, se podrán describir una amplia gama de características de los sistemas a construir, y de propiedades a analizar,

si el lenguaje cuenta con **mecanismos efectivos para el análisis** de propiedades, éste demanda menos esfuerzos para el estudio de las consecuencias de las funcionalidades requeridas del sistema a construir.

Alloy: Un Lenguaje Formal de Modelado Basado en Relaciones

Alloy es un lenguaje de especificaciones formales que reúne varias de las características deseables mencionadas:

- es formal, con una semántica simple basada en relaciones,

- es declarativo, ofreciendo un lenguaje que se asemeja a las abstracciones a las cuales están acostumbrados los programadores,

- es expresivo, proveyendo construcciones que permiten expresar una amplia variedad de propiedades,

- permite el análisis automático de especificaciones, a través de un mecanismo de análisis denominado **SAT solving**.

Alloy: Un Lenguaje Formal de Modelado Basado en Relaciones

Alloy es un lenguaje de especificaciones formales que reúne varias de las características deseables mencionadas:

es formal, con una semántica simple basada en relaciones,

es declarativo, ofreciendo un lenguaje que se asemeja a las abstracciones a las cuales están acostumbrados los programadores,

es expresivo, proveyendo construcciones que permiten expresar una amplia variedad de propiedades,

permite el análisis automático de especificaciones, a través de un mecanismo de análisis denominado **SAT solving**.

Alloy Analyzer transforma las especificaciones que Alloy debe resolver en una especificación lógica para una herramienta de SAT solving.

La Sintaxis Alloy: Cómo describir modelos

La sintaxis del lenguaje Alloy es simple, con pocas construcciones que permiten aprender el lenguaje rápidamente. Las especificaciones sólo cuentan con 4 construcciones:

signaturas, para describir dominios de datos, abstracciones de datos y módulos

predicados, para describir operaciones asociadas con los dominios de datos, abstracciones de datos y módulos

hechos, para describir propiedades que se suponen verdaderas en el modelo

aserciones, para describir las propiedades cuya validez el diseñador quiere chequear.

Ausencia de Tipos en Alloy

Una característica importante de Alloy, relacionada con la analizabilidad de las especificaciones, es que no provee tipos predefinidos (con excepción de int). El diseñador deberá especificar entonces los dominios de datos necesarios para su problema a través de firmas básicas.

Por ejemplo, si necesitamos manipular nombres y direcciones (para un modelo de una agenda, por ejemplo), podemos hacerlo de la siguiente manera:

```
sig Name { }      sig Addr{ }
```

Ausencia de Tipos en Alloy

Una característica importante de Alloy, relacionada con la analizabilidad de las especificaciones, es que no provee tipos predefinidos (con excepción de int). El diseñador deberá especificar entonces los dominios de datos necesarios para su problema a través de signatures básicas.

Por ejemplo, si necesitamos manipular nombres y direcciones (para un modelo de una agenda, por ejemplo), podemos hacerlo de la siguiente manera:

sig Name { }

sig Addr{ }



cada una de estas signatures
denota un conjunto

Signaturas y Predicados en la Definición de Abstracciones de Datos

Podemos combinar **signaturas (compuestas)** y predicados para definir abstracciones de datos, de manera similar a la definición de clases en lenguajes orientados a objetos. Por ejemplo, podemos definir una agenda de direcciones de la siguiente manera:

```
sig AddressBook {  
  
    names: set Name,  
  
    addr: names -> Address  
  
}
```

Signaturas y Predicados en la Definición de Abstracciones de Datos

Podemos combinar **signaturas (compuestas)** y predicados para definir abstracciones de datos, de manera similar a la definición de clases en lenguajes orientados a objetos. Por ejemplo, podemos definir una agenda de direcciones de la siguiente manera:

```
sig AddressBook {  
  
    names: set Name,  
  
    addr: names -> Address  
  
}
```

relación ternaria, asocia libretas, nombres y direcciones : esta relación contiene $b \rightarrow n \rightarrow a$ cuando en la libreta b el nombre n mapea a la dirección a

Signaturas y Predicados en la Definición de Abstracciones de Datos

Podemos combinar **signaturas (compuestas)** y predicados para definir abstracciones de datos, de manera similar a la definición de clases en lenguajes orientados a objetos. Por ejemplo, podemos definir una agenda de direcciones de la siguiente manera:

```
sig AddressBook {  
    names: set Name,  
    addr: names -> Address  
}
```

relación binaria, asocia libretas con nombres : esta relación contiene $b \rightarrow n$ cuando la libreta b contiene el nombre n

relación ternaria, asocia libretas, nombres y direcciones : esta relación contiene $b \rightarrow n \rightarrow a$ cuando en la libreta b el nombre n mapea a la dirección a

Signaturas y Predicados en la Definición de Abstracciones de Datos

Signaturas y Predicados en la Definición de Abstracciones de Datos

Podemos combinar signaturas (compuestas) y **predicados** para definir abstracciones de datos, de manera similar a la definición de clases en lenguajes orientados a objetos. Por ejemplo, podemos definir una agenda de direcciones de la siguiente manera:

Signaturas y Predicados en la Definición de Abstracciones de Datos

Podemos combinar signaturas (compuestas) y **predicados** para definir abstracciones de datos, de manera similar a la definición de clases en lenguajes orientados a objetos. Por ejemplo, podemos definir una agenda de direcciones de la siguiente manera:

Signaturas y Predicados en la Definición de Abstracciones de Datos

Podemos combinar signaturas (compuestas) y **predicados** para definir abstracciones de datos, de manera similar a la definición de clases en lenguajes orientados a objetos. Por ejemplo, podemos definir una agenda de direcciones de la siguiente manera:

```
sig AddressBook {  
  
    names: set Name,  
  
    addr: names -> Address  
  
}
```

Signaturas y Predicados en la Definición de Abstracciones de Datos

Podemos combinar signaturas (compuestas) y **predicados** para definir abstracciones de datos, de manera similar a la definición de clases en lenguajes orientados a objetos. Por ejemplo, podemos definir una agenda de direcciones de la siguiente manera:

```
sig AddressBook {  
  
    names: set Name,  
  
    addr: names -> Address  
  
}
```

```
pred addAddress(b,b': AddressBook, n: Name, a:  
Address) {  
  
    b'.names = b.names + n  
  
    b'.addr = b.addr + (n->a)  
  
}
```

Signaturas y Predicados en la Definición de Abstracciones de Datos

declarativo: no hay cambio de estado explícito.
estado “antes” y “después” tienen diferentes nombres, y permite chequear cuando un cambio de estado es válido

Podemos combinar signaturas (compuestas) y **predicados** para definir abstracciones de datos, de manera similar a la definición de clases en lenguajes orientados a objetos. Por ejemplo, podemos definir una agenda de direcciones de la siguiente manera:

```
sig AddressBook {  
  
    names: set Name,  
  
    addr: names -> Address  
  
}
```


```
pred addAddress(b,b': AddressBook, n: Name, a:  
Address) {  
  
    b'.names = b.names + n  
  
    b'.addr = b.addr + (n->a)  
  
}
```

Analizando Modelos en Alloy

Tenemos dos formas de analizar modelos en Alloy.

Podemos pedir instancias de predicados:

run addAddress




busca una instancia de addAddress (que satisfaga el modelo) con a lo sumo 3 direcciones, 3 nombres y 3 libretas

Analizando Modelos en Alloy

Tenemos dos formas de analizar modelos en Alloy.

Podemos pedir instancias de predicados:

run addAddress



busca una instancia de addAddress (que satisfaga el modelo) con a lo sumo 3 direcciones, 3 nombres y 3 libretas

Para poder realizar estas tareas de análisis, Alloy necesita que demos una cota k para los tamaños de los dominios (por defecto es tres).

Analizando Modelos en Alloy

Tenemos dos formas de analizar modelos en Alloy.

Podemos pedir instancias de predicados:

run addAddress

busca una instancia de addAddress (que satisfaga el modelo) con a lo sumo 3 direcciones, 3 nombres y 3 libretas

Para poder realizar estas tareas de análisis, Alloy necesita que demos una cota k para los tamaños de los dominios (por defecto es tres).

run addAddress for 3 but exactly 2 AddressBook

busca una instancia de addAddress (que satisfaga el modelo) con a lo sumo 3 direcciones, 3 nombres y 2 libretas

Analizando Modelos en Alloy

Tenemos dos formas de analizar modelos en Alloy.

Podemos pedir instancias de predicados:

run addAddress

busca una instancia de addAddress (que satisfaga el modelo) con a lo sumo 3 direcciones, 3 nombres y 3 libretas

Para poder realizar estas tareas de análisis, Alloy necesita que demos una cota k para los tamaños de los dominios (por defecto es tres).

run addAddress for 3 but exactly 2 AddressBook

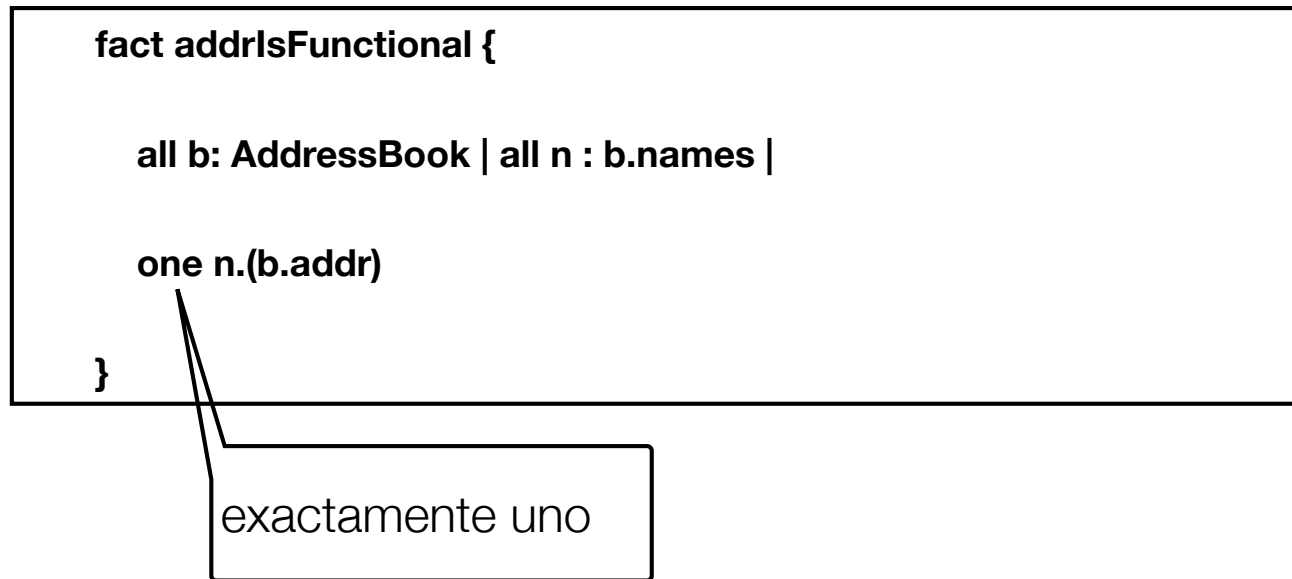
busca una instancia de addAddress (que satisfaga el modelo) con a lo sumo 3 direcciones, 3 nombres y 2 libretas

Restringiendo Modelos Válidos con Hechos

Los hechos nos permiten restringir los modelos que consideramos válidos con ciertas propiedades. Por ejemplo, podemos requerir que `addr` es funcional:

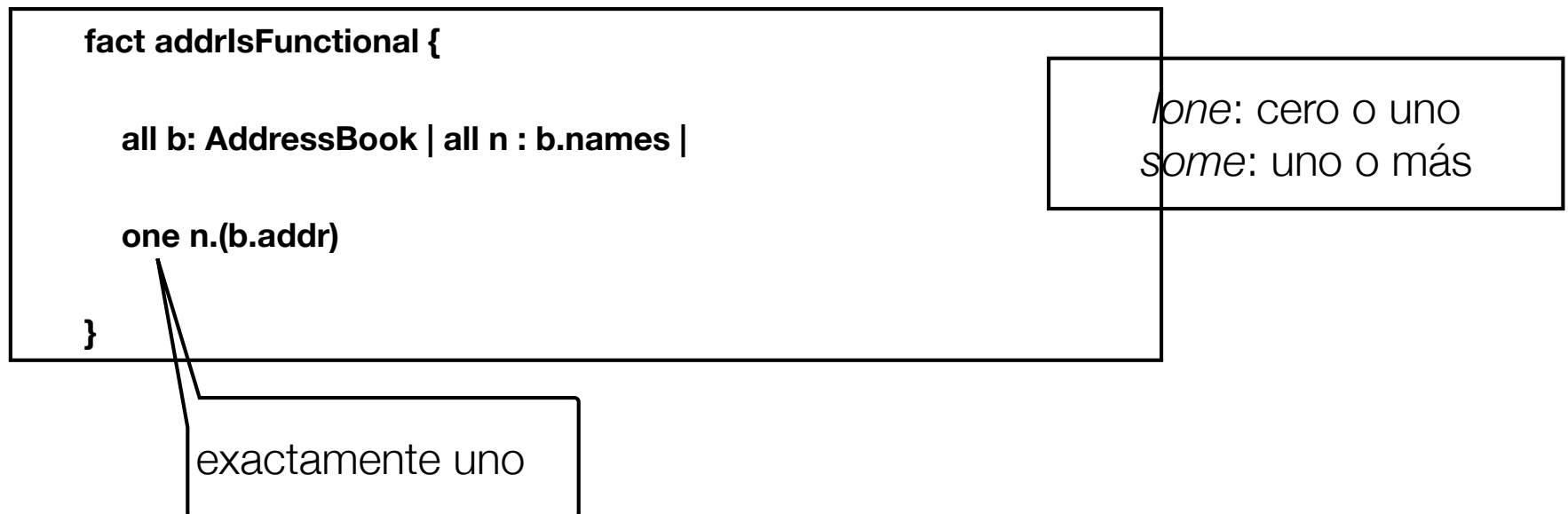
Restringiendo Modelos Válidos con Hechos

Los hechos nos permiten restringir los modelos que consideramos válidos con ciertas propiedades. Por ejemplo, podemos requerir que `addr` es funcional:



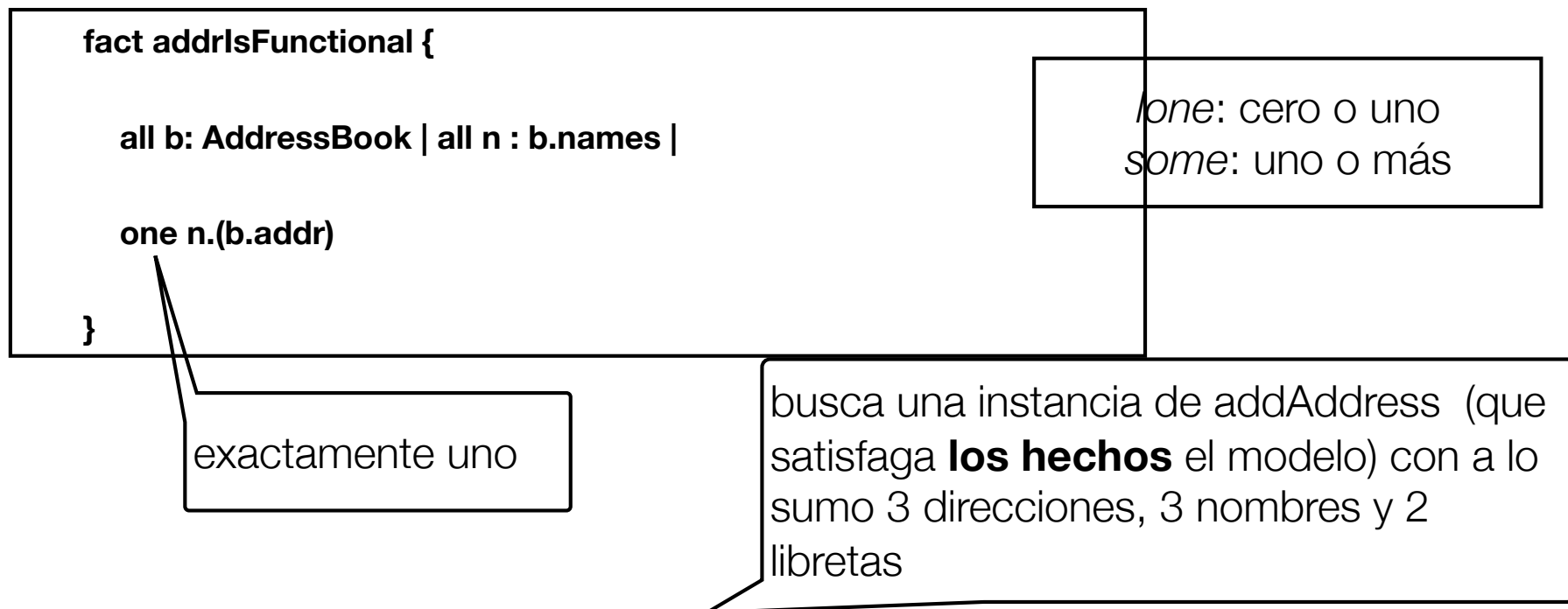
Restringiendo Modelos Válidos con Hechos

Los hechos nos permiten restringir los modelos que consideramos válidos con ciertas propiedades. Por ejemplo, podemos requerir que `addr` es funcional:



Restringiendo Modelos Válidos con Hechos

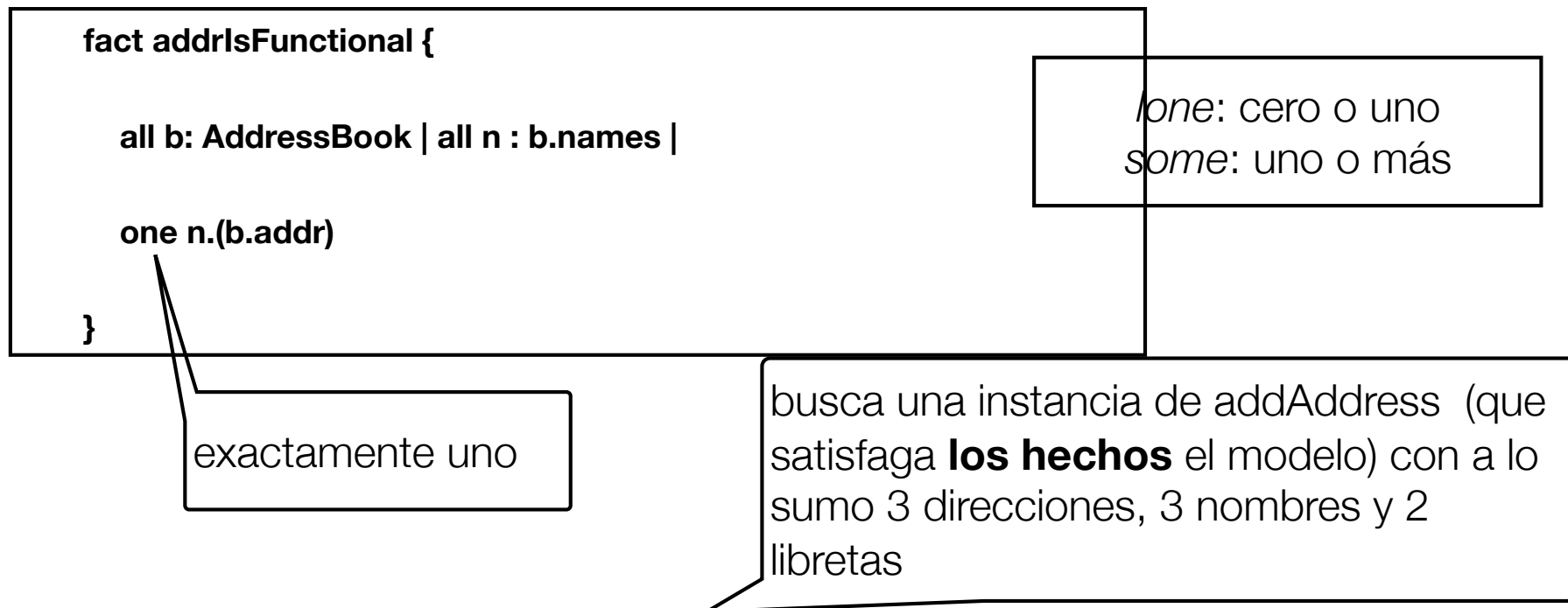
Los hechos nos permiten restringir los modelos que consideramos válidos con ciertas propiedades. Por ejemplo, podemos requerir que `addr` es funcional:



run addAddress for 3 but exactly 2 AddressBook

Restringiendo Modelos Válidos con Hechos

Los hechos nos permiten restringir los modelos que consideramos válidos con ciertas propiedades. Por ejemplo, podemos requerir que `addr` es funcional:



Propiedades a Chequear en Alloy

Las **aserciones** nos permiten expresar, al igual que los **hechos**, propiedades de los modelos. Sin embargo, a diferencia de los hechos (que se suponen verdaderos), las aserciones expresan propiedades que queremos chequear en nuestros modelos:

Propiedades a Chequear en Alloy

Las **aserciones** nos permiten expresar, al igual que los **hechos**, propiedades de los modelos. Sin embargo, a diferencia de los hechos (que se suponen verdaderos), las aserciones expresan propiedades que queremos chequear en nuestros modelos:

```
assert addAddrChangesAddressBook {
```

```
    all b, b': AddressBook | all n: Name | all a: Address | addAddress[b, b', n, a] => b != b'
```

```
}
```


Analizando Modelos en Alloy

Tenemos dos formas de analizar modelos en Alloy.

Podemos pedir instancias de predicados,

chequear la validez de aserciones en nuestras especificaciones:

check addAddrChangesAddressBook

busca contraejemplos de la aserción, que satisfagan los axiomas, y con a lo sumo 3 direcciones, 3 nombres y 3 libretas

Analizando Modelos en Alloy

Tenemos dos formas de analizar modelos en Alloy.

Podemos pedir instancias de predicados, `run addAddress`

chequear la validez de aserciones en nuestras especificaciones:

`check addAddrChangesAddressBook`

busca contraejemplos de la aserción, que satisfagan los axiomas, y con a lo sumo 3 direcciones, 3 nombres y 3 libretas

Analizando Modelos en Alloy

Tenemos dos formas de analizar modelos en Alloy.

Podemos pedir instancias de predicados, `run addAddress`

chequear la validez de aserciones en nuestras especificaciones:

`check addAddrChangesAddressBook`

busca contraejemplos de la aserción, que satisfagan los axiomas, y con a lo sumo 3 direcciones, 3 nombres y 3 libretas

Para poder realizar estas tareas de análisis, Alloy necesita que demos una cota k para los tamaños de los dominios (por defecto es tres).

Analizando Modelos en Alloy

Tenemos dos formas de analizar modelos en Alloy.

Podemos pedir instancias de predicados, `run addAddress`

chequear la validez de aserciones en nuestras especificaciones:

`check addAddrChangesAddressBook`

busca contraejemplos de la aserción, que satisfagan los axiomas, y con a lo sumo 3 direcciones, 3 nombres y 3 libretas

Para poder realizar estas tareas de análisis, Alloy necesita que demos una cota k para los tamaños de los dominios (por defecto es tres).

Expresiones en Alloy

Tanto las firmas como los campos de las firmas definen relaciones. Por ejemplo, `Name` define una relación unaria, y `names` define una relación binaria entre libretas de direcciones y nombres.

Tenemos algunas constantes relacionales:

none (rel. vacía), **univ** (relación unaria universal) e **iden** (relación identidad sobre el universo)

`Name = {(N0), (N1), (N2)}`

`Address = {(D0), (D1)}`

estas constantes tienen los valores

none = {}

univ = {(N0), (N1), (N2), (D0), (D1)}

iden = {(N0, N0), (N1, N1), (N2, N2), (D0, D0), (D1, D1)}

Expresiones en Alloy

Varias operaciones pueden aplicarse a relaciones:

unión (+), **intersección** (&), **diferencia** (-) y **producto cartesiano** (->)

composición (.)

conversa (\sim)

overriding ($r++s = (r - (\text{dom}[s] \rightarrow \text{univ})) + s$)

clausura transitiva (\wedge^r) y **reflexo-transitiva** (\wedge^*r)

Fórmulas en Alloy

Usando las expresiones relacionales, podemos construir fórmulas:

inclusión relacional ($r \text{ in } s$)

igualdad relacional ($r = s$)

vacuidad relacional ($\text{no } r$)

conjunción ($f1 \text{ and } f2$), **disyunción** ($f1 \text{ or } f2$), **negación** ($\text{not } f1$) e **implicación** ($f1 \Rightarrow f2$) de fórmulas

cuantificación universal ($\text{all } x: D \mid f1$)

cuantificación existencial ($\text{some } x: D \mid f1$)

En Alloy, toda expresión denota una relación. Los elementos se caracterizan mediante singletons (relaciones unarias con un único elemento)

Ejercicios Interactivos en Alloy

1. Descargue el Alloy Analyzer de <http://alloy.mit.edu/> e instálelo.

2. Complete el modelo de la agenda, con operaciones de agregación, consulta y eliminación de direcciones.

Chequee las siguientes propiedades:

si se elimina una persona de la agenda, ésta ya no pertenece a la misma

si se agrega una dirección y luego se elimina, volvemos a la agenda original

sobreescribir una dirección en la agenda pierde el primer valor

3. Describa en Alloy un modelo de personas, con información sobre cónyuge e hijos. Agregue hechos que describan restricciones biológicas, normas sociales, etc.

Si agrega el hecho de que todas las personas descenden de Adán y Eva, si se respetan las normas sociales y las restricciones biológicas impuestas, podrá haber más personas, además de Adán y Eva?

Bibliografía

Software Abstractions: Logic, Language, and Analysis Daniel Jackson

<http://alloy.lcs.mit.edu/alloy/index.html>