# HERRAMIENTAS AVANZADAS PARA EL DESARROLLO DE SOFTWARE

## Tema IV
## Model Checking Abstracto

**Alicia Villanueva**

Curso 05/06

---

MODEL CHECKING ABSTRACTO

# Índice

**1. Introducción** **3**
 1.1. Objetivos del tema . . . . . . . . . . 4

**2. Model Checking** **4**
 2.1. El modelo . . . . . . . . . . 5
 2.2. La propiedad . . . . . . . . . . 7
  2.2.1. Clasificación . . . . . . . . . . 7
 2.3. La lógica temporal lineal . . . . . . . . . . 8
  2.3.1. La lógica temporal lineal proposicional . . . . . . . . . . 9
 2.4. La lógica temporal ramificada . . . . . . . . . . 10
  2.4.1. La lógica temporal ramificada proposicional . . . . . . 11

**3. Model Checking Abstracto** **14**
 3.1. La lógica . . . . . . . . . . 14
 3.2. Sistemas de transiciones . . . . . . . . . . 16
 3.3. Abstracción del modelo . . . . . . . . . . 19
  3.3.1. Preservación de resultados . . . . . . . . . . 21
  3.3.2. Estructura de *Kripke* abstracta . . . . . . . . . . 22
  3.3.3. La relación de transición vinculada . . . . . . . . . . 25
  3.3.4. La relación de transición libre . . . . . . . . . . 26
  3.3.5. Modelo abstracto final . . . . . . . . . . 28
 3.4. Abstracción de programas . . . . . . . . . . 29
  3.4.1. Ejemplo de los matemáticos: Final . . . . . . . . . . 31

**4. Ejercicios** **34**

# 1. Introducción

Ya sabemos que la interpretación abstracta es una técnica introducida por Cousot y Cousot en 1977 como forma unificada para el análisis de programas. La intuición nos dice que un programa denota computaciones en un universo de objetos determinado. La interpretación abstracta hemos visto que usa esta denotación para describir computaciones en un universo diferente (abstracto) de objetos de forma que los resultados en el universo abstracto nos dan información sobre las ejecuciones en el universo original.

En nuestro caso, la idea principal que debe regir todo marco de trabajo es que cualquier elemento concreto debe estar relacionado (mediante $\alpha$) con un elemento abstracto. Además, cualquier elemento abstracto debe estar relacionado (mediante $\gamma$) con al menos un elemento concreto y, por fin, sabemos que la relación entre elementos debe ser un preorden.

En el *model checking* abstracto, la idea fundamental es que si conseguimos probar una propiedad en el universo abstracto, garantizaremos que dicha propiedad se cumple también en el universo concreto. Sin embargo, si no conseguimos probar una propiedad, entonces puede que en el universo real se cumpla o no. Gracias a las sub-aproximaciones, podemos llegar a tener también una *conservación estricta*, aunque no suele ser tan frecuente. En este caso cuando una propiedad sea falsa en el abstracto, entonces lo será también en el concreto.

Ya sabemos que existen distintos tipos de sistemas. En una clasificación muy simplificada, tenemos los sitemas funcionales por un lado y los sistemas reactivos por el otro. Los sistemas funcionales tienen un punto final concreto, donde podemos verificar una posible postcondición. Para dichos sistemas tenemos lógicas como la de Hoare, de precondiciones y postcondiciones, etc. a nuestra disposición, y las técnicas de *testing* y *depuración*, aunque bajo determinado contexto se pueden aplicar a sistemas reactivos, son especialmente adecuadas para verificar sistemas funcionales, ya que analizan el resultado final, la relación de entrada/salida de los programas.

Sin embargo, en los sitemas reactivos no hay un *fin*. Los clásicos sistemas reactivos son los sistemas empotrados (el *software* que controla las máquinas de café, los cajeros automáticos, etc.) pero también los sistemas operativos, los protocolos de comunicación, y en general todo sistemas que interactúe con el usuario o algún otro *software*. Normalmente estos sistemas se especifican como sistemas concurrentes, por lo que su análisis suele ser complejo, sobretodo si se realiza *a mano*. Si añadimos el hecho de que podemos tener sistemas no deterministas, vemos el motivo por el que las técnicas de verificación automática para sistemas reactivos son tan importantes en la actualidad: hoy en día el *software* que se produce es cada vez más grande y complejo.

En un sistema reactivo, la ejecución podría continuar y continuar de forma que nunca habrá un sitio donde poder comprobar o verificar una

postcondición. Por ello nos hace falta otro tipo de lógica, una lógica que razone y analice las trazas de ejecución y no el resultado final. Además, las propiedades que normalmente queremos verificar o analizar suelen ser propiedades dinámicas, y lo queremos hacer de una forma estática (es decir, sin necesidad de ejecutar el programa). Esto hace que el problema se convierta en un problema NP-completo ya que aparecene el ya famosos problema de la explosión del espacio de estados y de la falta de cobertura. Las lógicas que tenemos que usar en estos casos se han estudiado ya en otras asignaturas, pero en este boletín daremos las nociones básicas para que la documentación sea completa.

En cualquier caso, lo que sí debemos tener claro en todo momento es que en los sistemas reactivos en general podemos estudiar cuatro tipo de propiedades fundamentales. En primer lugar, podemos estudiar propiedades de seguridad (*safety*) o de viveza (*liveness*), pero también podemos dividir el tipo de propiedades a analizar entre *universales* o *existenciales*. Obviamente, normalmente tendremos combinaciones de estos dos criterios, podremos por tanto poder estudiar propiedades de seguridad universales, o de viveza universales, de seguridad existenciales o de seguridad universales. Las técnicas usadas para verificar, propiedades de seguridad universales son distintas de las usadas para analizar propiedades existenciales o de viveza ya que en el caso de las de seguridad universales, suele resultar más sencillo buscar contraejemplos que analizar la satisfacción de la propiedad.

### 1.1. Objetivos del tema

El objetivo de este tema es, principalmente, ver la aplicacin de la técnica de interpretación abstracta aplicada al *model checking*. Se considerarn todos los aspectos de la metodología (abstraccin de dominios, de relaciones, etc.) y se estudiará el problema del refinamiento de la asbtracción, o la optimalidad de la abstracción.

## 2. Model Checking

La técnica de *model checking* ya ha sido estudiada en la asignatura MFI del primer cuatrimestre. *model checking* consiste básicamente en la verificacin de que un determinado sistema satisface una propiedad. Es una técnica formal porque tiene una base fundamental matemática, y por lo tanto da una respuesta *segura*. Sin embargo, debido a que está basada en la búsqueda exhaustiva en el espacio de estados del sistema, a priori puede aplicarse únicamente a sistemas con un espacio de búsqueda finito (e incluso no demasiado grande). Ya se ha visto que existen muchas técnicas que permiten mejorar la eficiencia, y por lo tanto la aplicabilidad, de la técnica. Por ejemplo, en el cuatrimestre anterior se estudió el *model checking* simbólico, basado

en la representación mediante funciones booleanas del modelo y la propiedad. Pero también existen otro tipo de optimizaciones como son las técnicas *on-the-fly*, basadas en lenguajes regulares, en métodos composicionales, en evaluación parcial, en simulaciones y órdenes parciales, y como veremos en este tema, en interpretación abstracta.

La técnica simbólica está restringida a la verificación de propiedades CTL debido a que se basa en la caracterización de punto fijo de las propiedades CTL. Veremos que en el caso de la técnica de *model checking* abstracto podremos verificar propiedades CTL$^*$, que son estrictamente más expresivas que las CTL.

Recordaremos que el problema del *model checking* se formula de la siguiente forma: dado el modelo del sistema $\mathcal{M}$ y una propiedad que queremos verificar $\phi$, se establece si

$$\mathcal{M} \models \phi$$

El algoritmo puede dar una respuesta positiva, en cuyo caso sabemos que el sistema satisface la propiedad, o bien una respuesta negativa, y en este caso se dará también un contraejemplo: la traza en la que la propiedad ha fallado. Recordemos que el enunciado nos dice que vamos a ver si, para todo estado inicial del modelo, la propiedad se satisface, es decir,

$$\forall s \in \mathit{init}(\mathcal{M}) \;\; \mathcal{M}, s \models \phi$$

## 2.1. El modelo

La estructura clásica usada para la representación del sistema (el modelo), suele ser algún tipo de *estructura de Kripke*:

**Definición 1 (Estructura de *Kripke*)** *Una estructura de* Kripke *es una tupla $\langle S, I, R, L\rangle$ donde*

- *$S$ es un conjunto de estados (todos los estados del sistema)*
- *$I \subseteq S$ es el conjunto de estados iniciales del sistema*
- *$R \subseteq S \times S$ es una relación (total) binaria definida entre estados, y*
- *$L : \wp(\Sigma) \rightarrow S$ es una función de etiquetado que define, para cada estado, qué expresiones son ciertas. Dicho de otra forma, para cada conjunto de expresiones del lenguaje, define en qué estados se satisfacen.*

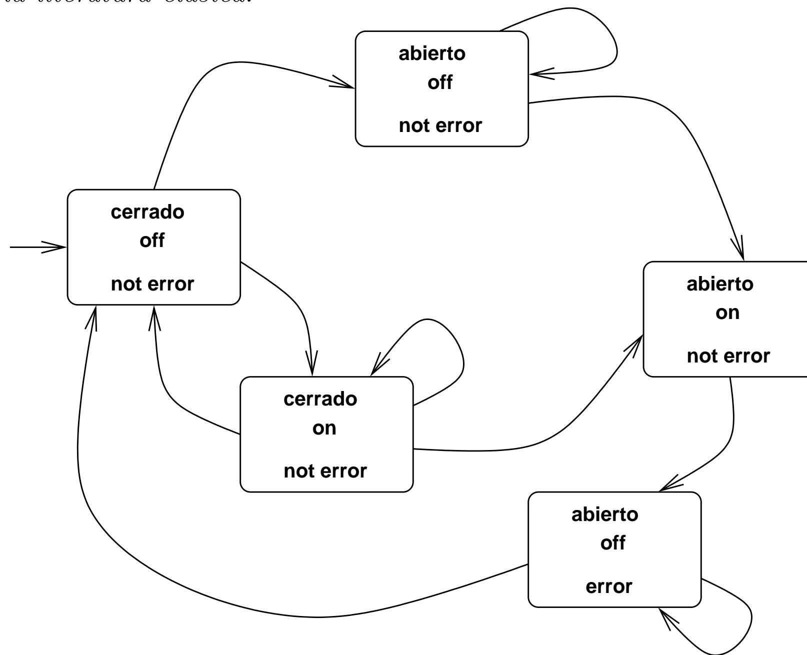
El hecho de exigir que la relación de transición sea total se debe a que estamos interesados en las trazas infinitas, y una forma de garantizarlo es pidiendo que todo estado tenga un sucesor. Sin embargo existen otras alternativas con las que no es necesario tener una relación total, ya que podemos

eliminar las trazas finitas a nivel de semántica de la lógica (capturando únicamente las trazas infinitas).

Esta estructura proviene de la interpretación de la *lógica modal* clásica, en la que cada nodo en realidad representaba un mundo posible, y las transiciones se correspondían con el paso de un mundo a otro en función del conocimiento adquirido hasta ese momento.

Una estructura de *Kripke* puede ser representada de forma gráfica mediante un grafo dirigido. Cada nodo del grafo representará un estado del sistema, mientras que la relación $R$ entre estados se representará mediante arcos entre los nodos. La función de etiquetado se representará definiendo en cada nodo el conjunto de expresiones o proposiciones que satisface.

**Ejemplo 1** *En la siguiente figura vamos a representar un típico ejemplo de estructura de* Kripke *que especifica el comportamiento de un microondas. El ejemplo, aunque parecido, no se corresponde exactamente con el presentado en la literatura clásica.*




Un modelo como el del ejemplo puede ser calculado a partir de un programa o especificación de alto nivel de formas distintas. En primer lugar, podemos generarlo *a mano*, aunque el problema de hacerlo así es que podríamos introducir errores adicionales (que no se corresponden con el programa real) sin darnos cuenta. Desde luego, es preferible hacerlo de forma automática aunque no todas las metodologías contemplan esta posibilidad y muchas veces nos vemos obligados a fiarnos de nuestra destreza a la hora de definirlo. Hay que tener en cuenta que, si detectamos un error en el modelo, después debemos ser capaces de identificarlo en el programa.

Por último debemos mencionar que existe una metodología que en vez de estructuras de *Kripke* maneja autómatas. Los algoritmos tienen un fundamento completamente distinto que no estudiaremos en este curso, pero no dejaremos de mencionar algunas características importantes. En general, los algoritmos que manejan autómatas suelen ser más eficientes que los que manejan otro tipo de modelos. Sin embargo, el problema principal de estos métodos es el hecho de que normalmente hay que traducir el sistema a autómata y esta transformación no es trivial. Tiene asociada un coste computacional importante, por lo que las ventajas de tener algoritmos eficientes para manejar autómatas se ven mitigadas. Siempre tendremos que estudiar el tipo de aplicación que queremos analizar y el tipo de propiedades que nos interesan para poder decidir qué metodología debemos usar.

## 2.2. La propiedad

Normalmente la propiedad que queremos verificar en el modelo suele expresarse en algún tipo de lógica temporal. En esta sección vamos a definir y describir las tres lógicas más frecuentes en este campo.

La lógica temporal es una clase particular de lógica modal que proporciona los mecanismos formales necesarios para describir de forma cualitativa ciertos aspectos d elos sistemas y para razonar sobre cómo evoluciona el valor de verdad de las afirmaciones a lo largo del tiempo. Las modalidades temporales clásicos son *eventualmente* y *siempre*.

Las lógicas temporales fueron rescatadas de los libros en el año 83 y siguientes debido a la necesidad mencionada antes de razonar sobre trazas y no sobre resultados finales de los programas. A partir de entonces, dejaron de ser un tipo de lógica olvidada y empezaron a aplicarse a la verificación de sistemas reactivos.

### 2.2.1. Clasificación

Podemos clasificar las lógicas temporales (TL) para analizar sistemas concurrentes según distintos criterios: lógicas proposicionales frente a de primer orden, globales frente a composicionales, ramificadas frente a lineales, puntuales frente a de intervalos, discretas frente a continuas o lógicas del pasado frente a lógicas del futuro.

Las lógicas temporales proposicionales toman la lógica proposicional clásica y le añaden una serie de operadores temporales. Así pues, las lógicas temporales proposicionales se basan en un conjunto de proposiciones atómicas que expresan hechos atómicos sobre los estados del sistema. Por otro lado, las lógicas temporales *de primer orden* se inspiran en la lógica de primer orden tradicional. No entraremos en detalle en este aspecto ya que no tratamos de dar un curso sobre lógicas.

En cuanto a la siguiente criterio, las lógicas globales se interpretan en

un único universo que se corresponde en nuestro contexto con un único programa. Por otro lado, la sintaxis de las lógicas composicionales nos permite expresar propiedades sobre distintos programas (o fragmentos del programa) en una misma fórmula.

La naturaleza del tiempo nos divide las lógicas en dos tipos: las ramificadas y las lineales, dependiendo de si estamos considerando un tiempo en el que puede haber distintos futuros (natural en sistemas no deterministas), o un tiempo en el que sólo hay un posible futuro. En el primer caso tenemos un tiempo en forma de árbol y los operadores de la lógica nos permiten cuantificar las ramas del árbol.

La mayoría de los formalismos lógicos están basados en el análisis del valor de verdad de una fórmula en un instante determinado. Son las lógicas de punto. Existen otras lógicas que tienen operadores que permiten ser evaluados en un intervalo de tiempo, lo que a veces puede simplificar la especificación de algunas propiedades.

Otra característica importante del tiempo nos proporciona la clasficación en cuanto a continuo o discreto. En la mayoría de sistemas, el tiempo se asume que es discreto donde el estado actual es el presente y el siguiente momento se corresponde con el siguiente estado del programa. También existen las lógicas *densas* en las que las fórmulas se interpretan según un tiempo continuo como pueden ser los reales o los racionales. Este segundo tipo de lógica es importante para expresar propiedades de sistemas híbridos, donde ciertas variables pueden tener una naturaleza continua (por ejemplo expresando temperaturas, volúmenes en sensores, etc.).

Por último, una lógica temporal puede tener operadores de tiempo que hacen referencia al pasado, que hacen referencia al futuro, o ambos. Hoy en día el uso de los operadores del pasado está limitado al hecho de que puedan simplificar la especificación de ciertas propiedades. En realidad, la introducción de operadores pasados sólo añade expresividad cuando manejamos una noción de equivalencia global, si por el contrario usamos una noción de equivalencia dependiendo de un instante inicial, entonces las dos versiones de la lógica son equivalentes.

## 2.3. La lógica temporal lineal

La estructura en la que se basa la lógica temporal lineal (LTL) es un conjunto totalmente ordenado. Normalmente se asume el orden $(\mathbb{N}, <)$. Por tanto, usaremos una noción de tiempo discreto, con un punto inicial que no tiene predecesores (un mínimo en el orden) e infinito. Esta es una noción apropiada para razonar sobre sistemas concurrentes ya que la ejecución de un programa es discreta (paso de estado a estado) y siempre comienza en un estado inicial. Los valores de verdad de las fórmulas se interpretan sobre secuencias de estados (computaciones o trazas).

### 2.3.1. La lógica temporal lineal proposicional

Los operadores básicos de la lógica temporal lineal proposicional (PLTL) son

- $\Diamond\phi$ (lo podemos encontrar como $F\phi$ también) leído *en algún instante en el futuro $\phi$ es cierta*,
- $\Box\phi$ (lo podemos encontrar como $G\phi$ también) leído *en todos los instantes futuros $\phi$ es cierta*,
- $\bigcirc\phi$ (también $X\phi$) leído *en el siguiente instante de tiempo $\phi$ es cierta*, y
- $\phi\mathcal{U}\psi$ leídos como *es cierta $\phi$ hasta el instante futuro donde $\psi$ es cierta*.

Dado un conjunto de proposiciones atómicas $\Omega$, la sintaxis de la lógica PLTL consiste en el menor conjunto de fórmulas generadas siguiendo las siguientes reglas:

1. toda proposición atómica es una fómula PLTL
2. dadas dos fórmulas PLTL $\phi$ y $\psi$, las fórmulas $\phi \wedge \psi$ y $\neg\phi$ también son fórmulas PLTL
3. dadas dos fórmulas PLTL $\phi$ y $\psi$, las fórmulas $\phi\mathcal{U}\psi$ y $\bigcirc\phi$ también son fórmulas PLTL

Como siempre, podemos definir los demás operadores en función de los anteriores como $\phi \vee \psi$ que abrevia $\neg(\neg\phi \wedge \neg\psi)$, $\phi \rightarrow \psi$ abrevia $\neg\phi \vee \psi$, o $\phi \equiv \psi$ abrevia $(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$. Además, *true* abrevia $\neg\phi \vee \phi$ mientras que *false* abrevia $\neg$*true*. En cuanto a los operadores temporales, $\Diamond\phi$ abrevia *true*$\mathcal{U}\phi$ y $\Box\phi$ abrevia $\neg\Diamond\neg\phi$.

La semántica de las fórmulas PLTL la definiremos basándonos en secuencias de estados en un modelo $M = (S, I, R, L)$ donde, como siempre $S$ es un conjunto de estados, R es una relación entre estados y $L$ es una función que etiqueta los estados con las fórmulas que se satisfacen en cada uno de ellos. Definimos una secuencia temporal como $s = s_0, s_1, \ldots, s_n, \ldots$ donde para cada $s_i$ y $s_{i+1}$ se cumple que existe un par $(s_i, s_{i+1}) \in R$. Diremos también que $s^i$ es el sufijo $s_i, s_{i+1}, s_{i+2}, \ldots$ de $s$.

Una vez definida la nomenclatura que usaremos, definiremos la relación de satisfacción $\models$ con respecto a una secuencia $s$. Esta definición se puede extender de forma obvia a modelos considerando las secuencias cuyo primer estado es un estado inicial del mismo. Abusaremos de notación y escribiremos $\models$ cuando hablemos tanto de satisfacción de secuencias como de satisfacción de modelos.

| | | |
|---|---|---|
| (1) | $s \models \phi$ | sii $\forall\phi$ proposición atómica de la lógica, $\phi \in L(s_0)$ |
| (2) | $s \models \phi \wedge \psi$ | sii $s \models \phi$ y además $s \models \psi$ |
| (3) | $s \models \neg\phi$ | sii $s \not\models \phi$ |
| (4) | $s \models (\phi \mathcal{U} \psi)$ | sii $\exists j$ tal que $s^j \models \psi$ y además $\forall k < j$ se cumple que $s^k \models \phi$ |
| (5) | $s \models \bigcirc\phi$ | sii $s^1 \models \phi$ |

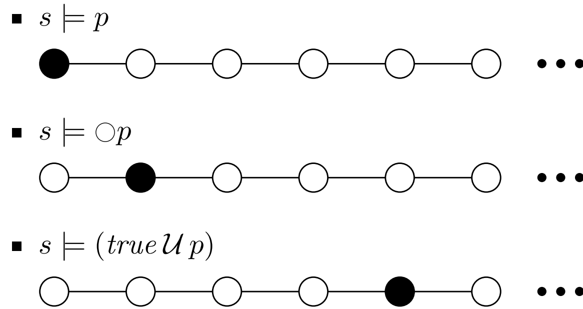
**Ejercicio 1** *Definir formalmente (es decir, usando una notación similar a la de los casos anteriores) la relación de satisfacción de las modalidades* $\Box$ *y* $\Diamond$.

**Ejercicio 2** *Escribir formalmente (es decir, usando la lógica PLTL) las siguientes sentencias:*

- *Si* $\phi$ *es cierta en este instante, entonces en algún momento futuro* $\psi$ *será cierta*

- *Siempre que* $\phi$ *es cierta,* $\psi$ *será cierta en algún momento futuro*

**Ejemplo 2** *En la siguiente figura, podemos ver representado el comportamiento de los distintos operadores y modalidades. Representaremos como una secuencia de puntos la traza* $s$, *y como puntos negros el momento en el que se satisface* $p$.




Por último daremos la caracterización de punto fijo de los operadores temporales:

$$
\begin{aligned}
\models \Diamond\phi &\equiv \phi \vee \bigcirc\Diamond\phi \\
\models \Box\phi &\equiv \phi \wedge \bigcirc\Box\phi \\
\models \phi\mathcal{U}\psi &\equiv \psi \vee (\phi \wedge \bigcirc(\phi\mathcal{U}\psi))
\end{aligned}
$$

## 2.4. La lógica temporal ramificada

En la lógica temporal ramificada, la estructura de tiempo que hay debajo es una estructura de árbol infinito. De hecho asumiremos que cada camino en el árbol es isomorfo a $\mathbb{N}$ y, además, cada nodo del árbol puede tener un

número infinito de sucesores inmediatos. Además todos los nodos del árbol tendrán como mínimo un sucesor.

Esta lógica se interpreta sobre un modelo de Kripke $M = (S, I, R, L)$ donde $S$ es un conjunto de estados, $R$ una relación binaria entre estados y $L$ una función que etiqueta los estados con las fórmulas que se cumplen en cada uno de ellos. Podemos ver esta estructura como un grafo etiquetado dirigido.

### 2.4.1. La lógica temporal ramificada proposicional

Existen dos lógicas principales de este tipo: la *Computational Tree Logic* (CTL) y la versión enriquecida CTL*. Estas lógicas, a parte de los operadores temporales, tienen dos operadores que hacen de cuantificadores sobre los caminos del árbol. $A$ cuantifica universalmente los caminos mientras que $E$ los cuantifica existencialmente.

A continuación daremos la sintaxis de la lógica CTL*. En esta lógica tenemos dos tipos de fórmulas: *state*-fórmulas y *path*-fórmulas cuyos valores de verdad se definen sobre estados o sobre caminos del árbol respectivamente. Las *state*-fórmulas se definen según las siguientes tres reglas:

(S1) las proposiciones atómicas son *state*-fórmulas
(S2) si $\phi$ y $\psi$ son *state*-fórmulas, entonces $\phi \wedge \psi$ y $\neg\phi$ también son *state*-fórmulas
(S3) si $\phi$ es una *path*-fórmula, entonces $A\phi$ y $E\phi$ son *state*-fórmulas

Y las siguientes son las reglas que definen las *path*-fórmulas:

(P1) las *state*-fórmulas son *path*-fórmulas
(P2) si $\phi$ y $\psi$ son *path*-fórmulas, entonces $\phi \wedge \psi$ y $\neg\phi$ también son *path*-fórmulas
(P3) si $\phi$ y $\psi$ son *path*-fórmulas, entonces $\bigcirc\phi$ y $\phi\mathcal{U}\psi$ son también *path*-fórmulas

La sintaxis de la lógica CTL se obtiene imponiendo algunas restricciones a las reglas anteriores de CTL*. Formalmente se sustituyen las reglas P1-P3 por la regla:

(P0) si $\phi$ y $\psi$ son *state*-fórmulas, entonces $\bigcirc\phi$ y $\phi\mathcal{U}\psi$ son *path*-fórmulas.

Informalmente, lo que se hace es obligar a los operadores $\Box$ y $\Diamond$ estar precedidos por un cuantificador, es decir, que tendríamos los siguientes operadores: $A\Box$, $E\Box$, $A\Diamond$ y $E\Diamond$.

Como nota importante diremos que CTL* subsume tanto a PLTL como a CTL, sin embargo, ni PLTL subsume a CTL ni CTL subsume a PLTL.

Necesitamos ahora dar significado a las fórmulas y lo haremos definiendo su semántica. CTL* se interpreta sobre un modelo de Kripke $M =$

$(S, I, R, L)$. Un camino completo (*fullpath*) sobre este modelo es una secuencia *infinita* $s = s_0, s_1, \ldots, s_n, \ldots$ de forma que $\forall i (s_i, s_{i+1}) \in R$. Igual que antes, $s^i$ denotará el sufijo $s_i, s_{i+1}, \ldots$ de la secuencia.

Escribiremos $M, s_0 \models \phi$ para decir que una *state*-fórmula $\phi$ es cierta según el modelo $M$ en el estado $s_0$. De forma similar, escribimos $M, s \models \phi$ para decir que una *path*-fórmula $\phi$ es cierta según el modelo $M$ y para la secuencia $s$.

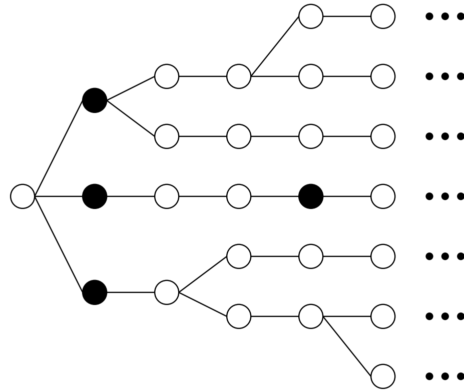
$\models$ se define de forma inductiva a partir de las siguientes reglas:

| | | |
|---|---|---|
| (S1) | $M, s_0 \models \phi$ | sii $\phi \in L(s_0)$ |
| (S2) | $M, s_0 \models \phi \wedge \psi$ | sii $M, s_0 \models \phi$ y además $M, s_0 \models \psi$ |
| (S3) | $M, s_0 \models \neg\phi$ | sii $M, s_0 \not\models \phi$ |
| (S4) | $M, s_0 \models E\phi$ | sii existe un *fullpath* $s = s_0, s_1, \ldots$ en $M$ tal que $M, s \models \phi$ |
| (S5) | $M, s_0 \models A\phi$ | sii para todo *fullpath* $s = s_0, s_1, \ldots$ en $M$ tal que $M, s \models \phi$ |
| (P1) | $M, s \models \phi$ | sii $M, s_0 \models \phi$ |
| (P2) | $M, s \models \phi \wedge \psi$ | sii $M, s \models \phi$ y además $M, s \models \psi$ |
| (P3) | $M, s \models \neg\phi$ | sii $M, s \not\models \phi$ |
| (P4) | $M, s \models \phi \mathcal{U} \psi$ | sii existe un $i$ tal que $M, s^i \models \psi$ y para todo $j$ menor que $i$, $M, s^j \models \phi$ |
| (P5) | $M, s \models \bigcirc\phi$ | sii $M, s^1 \models \phi$ |

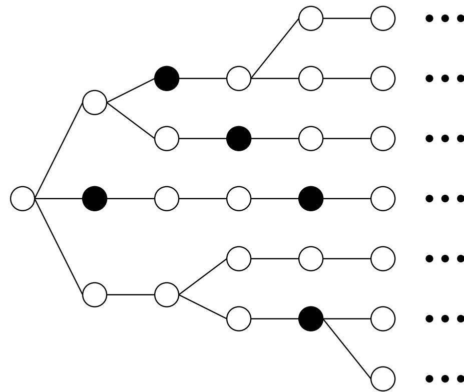
Decimos que una *state*-fórmula (*path*-fórmula) es válida si para todo modelo $M$ y todo estado $s_i$ (camino $s$) de $M$, se cumple que $M, s_i \models \phi$ ($M, s \models \phi$). Además, una *state*-fórmula (*path*-fórmula) es satisfacible si existe un modelo $M$ y existe un estado $s_i$ (camino $s$) de $M$ donde se cumple que $M, s_i \models \phi$ ($M, s \models \phi$). Podemos usar esta semántica para interpretar fórmulas CTL.

**Ejemplo 3** *En las siguientes figuras, podemos ver representado el comportamiento de los distintos operadores y modalidades. Representaremos como puntos negros el momento en el que se satisface $p$ y grises el momento en el que se satisface $q$.*

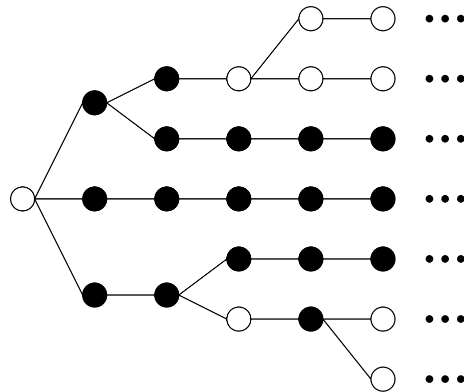
- $s \models A \bigcirc p$



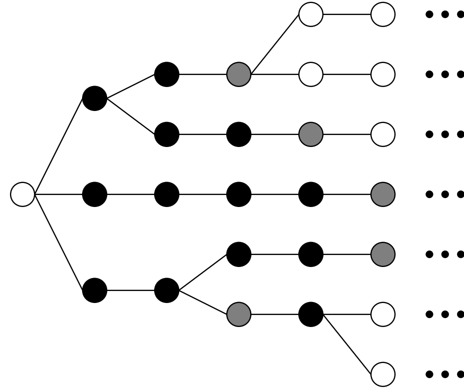
- $s \models E\Diamond p$ y $s \not\models A\Diamond p$



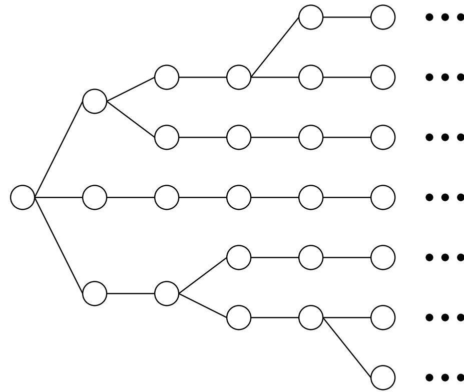
- $s \models A \bigcirc (E\Box p)$



- $s \models A(p\,\mathcal{U}\,q)$



- $s \not\models E(true\,\mathcal{U}\,p)$



# 3. Model Checking Abstracto

En el tema anterior vimos que para abstraer un dominio debemos definir un conjunto de elementos abstractos con una serie de características, una relación entre el conjunto de elementos abstractos y concretos, y debemos definir también una abstracción de las operaciones y expresiones del lenguaje considerado.

## 3.1. La lógica

En este tema vamos a introducir el *model checking* abstracto de propiedades CTL*. Vamos a concretar un poco la notación que usaremos a partir de ahora. Para la cuantificación universal usaremos $A$, para la existencial $E$ y para los operadores temporales $\bigcirc$, $\mathcal{U}$ y $\mathcal{V}$. Este último operador puede verse como el inverso del $\mathcal{U}$:

$$(\psi_1\,\mathcal{V}\,\psi_2) \equiv \neg(\neg\psi_1\,\mathcal{U}\,\neg\psi_2)$$

Es decir, que $\psi_2$ será cierta mientras $\psi_1$ sea falsa, y sólo después de que $\psi_1$ tome el valor cierto, $\psi_2$ podrá tomar el valor falso.

**Ejercicio 3** *¿A qué concepto inuitivo corresponde este nuevo operador?*

En esta versión de la lógica, podemos poner negaciones sólo delante de proposiciones atómicas, por ello necesitamos el operador $\mathcal{V}$. Sin embargo, una formulación equivalente sería eliminar este operador pero permitir la negación delante de fórmulas que no sean proposiciones atómicas.

Sintácticamente, una propiedad estará formada por un conjunto de *proposiciones atómicas* *Prop*. Llamaremos *literales* al conjunto de expresiones

$$Lit = Prop \cup \{\neg p | p \in Prop\}$$

A partir de los literales podemos definir las *state*-fórmulas:

$$\phi ::= p|\phi \wedge \phi|\phi \vee \phi|A\psi|E\psi$$

y el conjunto de *path*-fórmulas:

$$\psi ::= \phi|\psi \wedge \psi|\psi \vee \psi| \bigcirc \psi|\psi \,\mathcal{U}\, \psi|\psi \,\mathcal{V}\, \psi$$

La semántica la hemos definido en la sección anterior, por lo que no volvermos a presentarla. Únicamente repetiremos el hecho de que las *state*-fórmulas deben su nombre al hecho de que la propiedad que se define toma como punto de referencia un punto determinado, mientras que las *path*-fórmulas analizan la propiedad para un camino. Sin embargo, un único punto puede verse también como un camino, por ello las *state*-fórmulas son en realidad también *path*-fórmulas.

Para hacer la especificación de propiedades más sencilla e intuitiva, normalmente se usan una serie de abreviaturas. Nótese que dichas abreviaturas no son aleatorias, sino que se ha probado la equivalencia con respecto a los operadores básicos. Por ejemplo $\neg\psi$ es equivalente a la forma normal negada de la fórmula $\psi$, y para hallar dicha forma normal podemos aplicar las siguientes reglas:

- **R1:** $\neg(\psi_1 \wedge \psi_2) \rightarrow \neg\psi_1 \vee \neg\psi_2$
- **R2:** $\neg(\psi_1 \vee \psi_2) \rightarrow \neg\psi_1 \wedge \neg\psi_2$
- **R3:** $\neg A\psi \rightarrow E\neg\psi$
- **R4:** $\neg E\psi \rightarrow A\neg\psi$
- **R5:** $\neg \bigcirc \psi \;\; \rightarrow \bigcirc\neg\psi$
- **R6:** $\neg(\psi_1 \,\mathcal{U}\, \psi_2) \rightarrow \neg\psi_1 \,\mathcal{V}\, \neg\psi_2$
- **R7:** $\neg(\psi_1 \,\mathcal{V}\, \psi_2) \rightarrow \neg\psi_1 \,\mathcal{U}\, \neg\psi_2$

Otras abreviaturas son las cásicas $true \equiv \psi_1 \vee \neg\psi_1$, $false \equiv \psi_1 \wedge \neg\psi_1$, $\psi_1 \rightarrow \psi_2 \equiv \neg\psi_1 \vee \psi_2$, $\Diamond\psi \equiv true\,\mathcal{U}\,\psi$, $\Box\psi \equiv false\,\mathcal{V}\,\psi \equiv \neg(true\,\mathcal{U}\,\neg\psi)$,

El model checking abstracto puede definirse para dos fragmentos de la lógica CTL* introducida: ∀CTL* y ∃CTL*. Estas dos lógicas se definen como la CTL*, solo que sólo se usará el cuantificador universal (respectivamente el existencial) en las propiedades especificadas. Es fácil identificar cuando una fórmula pertenece a uno de estos fragmentos siempre y cuando nos fijemos en la forma normal negada, ya que de otra forma podríamos confundirlos.

**Ejercicio 4** *¿Pertenece la siguiente fórmula a la lógica ∀CTL*?*

$$A\Diamond p \rightarrow A\Box q$$

**Ejercicio 5** *Tomando como referencia el modelo definido en la sección anterior que describía el comportamiento de un microondas, responder a las siguientes preguntas:*

- *Escribir la fórmula CTL* que especifica que siempre que la puerta del microondas esté abierta y el microondas esté encendido, se producirá un error.*
- *¿El modelo satisface la fórmula* $\Box A(\text{cerrado}\wedge\text{on}) \rightarrow A\Diamond(\text{cerrado}\wedge\text{off})$*?*
- *¿El modelo satisface la fórmula* $\Box A((\text{abierto}\wedge\text{off})\vee(\text{cerrado})\vee((\text{abierto}\wedge\text{on}) \rightarrow A\bigcirc\text{error}))$*?*
- *¿Se satisfaría la condición anterior si hubiera un* arco reflexivo *en el nodo etiquetado como* abierto, on, ¬ error*?*

Como último detalle mencionaremos que si tuviéramos una serie de cuantificadores existenciales y universales seguidos en una fórmula, podemos simplemente sustituir dicha secuencia de cuantificadores por el que aparezca en último lugar (más a la derecha), de forma que obtendremos una fórmula equivalente.

## 3.2. Sistemas de transiciones

Un sistema de transiciones sobre un conjunto $S$ de eestados es un par $(S, R)$ donde $R$ es una relación $R \subseteq S \times S$. Un camino $\pi$ es una secuencia infinita $\pi = s_0 s_1 s_2 \ldots$ de estados, para los que se cumple que para todo $i \in \mathbb{N}, R(s_i, s_{i+1})$. Al elemento i-ésimo del camino lo denotaremos como $\pi(i)$. $\pi^n$ denotará el sufijo de $\pi$ cuyo primer elemento es $\pi(n)$.

Esta es la definición más general de sistema de transición, sin embargo un sistema de transición puede tener algunos atributos adicionales, como el conjunto de estados iniciales $I \subseteq S$. En función de este conjunto $I$, se define la noción de *alcanzabilidad*. Decimos que un camino $\pi$ es un $t$-camino si su

primer estado es $t$. Podemos decir ahora que un estado $s$ es alcanzable si existe un $t$-camino (siendo $t \in I$) que pase por $s$.

Al sistema de transciones se le puede añadir también una *interpretación* que, a cada elemento del conjunto de literales, asocie el conjunto de estados donde dicho literal es válido. A esta función de interpretación le imponemos una restricción, ya que no permitiremos que la versión negada de una proposición atómica esté asociada a un estado donde ya estaba asociada dicha proposición atómica. Sin embargo, pueden existir estados donde ni la proposición atómica $p$ ni el literal $\neg p$ estén. De esta forma permitimos la existencia de estados donde el valor de una determinada proposición sea *desconocido*.

Es sencillo asociar la noción de sistema de transiciones con estos dos atributos a una estructura de *Kripke*. El etiquetado de la estructura da la interpretación de los literales sobre los estados. En la sección anterior dimos la interpretación de la lógica CTL$^*$ con respecto a las estructuras de *Kripke*, así que no la repetiremos aquí.

Por último vamos a definir la noción de simulación entre sistemas de transiciones. Esta noción es importante porque con ella podremos relacionar modelos y asegurar que ciertas propiedades se preservan cuando pasemos de uno a otro. Las simulaciones se definen en función de una relación entre estados de los dos sistemas de transiciones:

**Definición 2** *Dadas dos estructuras de Kripke* $(S_1, I_1, R_1, L_1)$ *y* $(S_2, I_2, R_2, L_2)$*. Sea* $\sigma \subseteq S_1 \times S_2$ *una relación que cumple que para todo* $s \in S_1$ *t* $t \in S_2$*, entonces* $\sigma(s,t)$ *implica que:*

1. $L_1(s) = L_2(t)$,
2. *para todo* $s'$ *tal que* $R(s,s')$*, existe un* $t'$ *tal que* $R_2(t,t')$ *y además* $\sigma(s',t')$.

Ya hemos visto un ejemplo de sistema de transición en este mismo tema, pero ahora vamos a introducir un ejemplo que iremos desarrollando a lo largo del resto del tema para ilustrar la técnica de abstracción.

**Ejemplo 4** *Vamos a suponer que tenemos un sistema que representa un protocolo de acceso mutuamente exclusivo por parte de dos procesos que se ejecutan en paralelo. Para tener una imagen más clara, vamos a suponer que tenemos dos matemáticos sentados a la mesa listos para comer. Los dos matemáticos deben turnarse para comer, de forma que su estado será* pensante *o* comiendo *de forma alternativa y sin límite de tiempo.*

*Para comer, cada matemático tendrá que consultar una variable del sistema (n) de forma que, si la variable es* par*, entonces el primer matemático podrá comer, mientras que si es* impar*, el que comerá será el segundo matemático. En cualquier caso, una vez terminado de comer, los matemáticos asignaran un valor nuevo a la variable* $n$*, cada uno de forma distinta.*

*Inicialmente, los dos matemáticos están pensando y la variable $n$ tiene un valor arbitrario.*

Para aclarar el objetivo final diremos que A partir de ese estado y teniendo en cuenta la descripción dada del sistema, lo que queremos es verificar lo siguiente:

1. que los matemáticos tienen acceso mutuamente excluyente al comedor
2. que los matemáticos no se bloquean, es decir, que si uno está comiendo, tarde o temprano el otro matemático también comerá

Estas dos propiedades serán cruciales para la determinación de si nuestro sistema se comporta de forma correcta o no.

La descripción dada para el ejemplo es intuitiva y clara para nosotros, pero no podemos usarla como entrada de un algoritmo. Por ello, todo problema es necesario codificarlo de forma que sea fácilmente manejable (de forma automática). Normalmente se recurre a los lenguajes de programación, pero también a modelos como los introducidos para el *model checking*.

**Ejemplo 5 (Continuación)** *Las variables $m_0$ y $m_1$ van a representar el estado en el que se encuentra el primer y segundo matemático respectivamente. Estas variables pueden tomar por tanto dos valores* pensando *o* comiendo. *El conjunto de estados del sistema serán configuraciones con tres componentes: el estado del primer matemático, el estado del segundo y el valor de la variable de control $n$. Así pues, los estados $\Sigma$ estarán formadas según*

$$\{pensando, comiendo\}^2 \times \mathbb{N}\backslash\{0\}$$

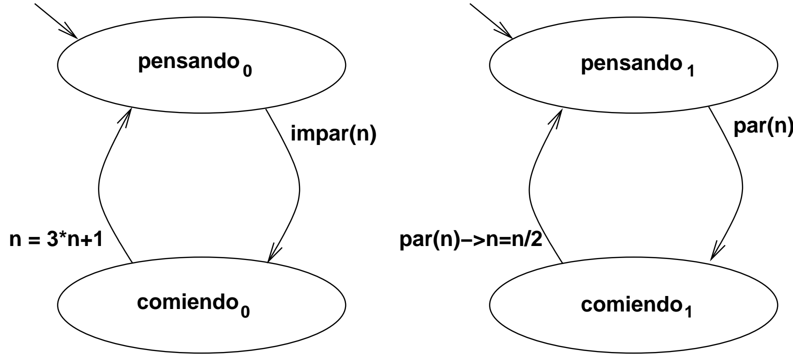
*y representadas de la forma $\langle m_0, m_1, n\rangle$.*

*Para definir las posibles transiciones que pueden darse durante la ejecución del sistema, definimos cuatro posibles acciones. La primera (Acción 1) nos dice que si el primer matemático está pensando y $n$ es impar, entonces el primer matemático pasará a estar comiendo:*

$$m_0 = pensando, odd(n) \rightarrow m_0 := comiendo$$

*De forma similar se definen la otras tres acciones (Acción 2, Acción 3 y Acción 4 respectivamente):*

$$\begin{array}{l} m_0 = comiendo \rightarrow m_0 := pensando, n := 3 * n + 1 \\ m_1 = pensando\, even(n) \rightarrow m_1 := comiendo \\ m_1 = comiendo \rightarrow m_1 := pensando, n := n/2 \end{array}$$




Hemos dado una codificación para el sistema que queremos verificar, pero ahora necesitamos codificar también las propiedades que hemos mencionado antes. Para ello usaremos la lógica temporal CTL$^*$ como ya sabemos.

**Ejemplo 6 (continuación)** *La primera propiedad que queremos especificar dice que* nunca podrán estar los dos matemáticos comiendo al mismo tiempo*. Teniendo en cuenta que A representa todos los posibles caminos, y que $\Box$ significa* siempre*, la propiedad la podemos escribir de la siguiente forma:*

$$\forall\Box\neg(m_0 = comiendo \wedge m_1 = comiendo)$$

*La segunda propiedad mencionada en la descripción se puede modelar con dos afirmaciones:* que siempre ocurre que si un matemático está comiendo, entonces eventualmente, se vaya por el camino que se vaya, el otro matemático comerá *y viceversa.*

$$\forall\Box(m_0 = comiendo \rightarrow \forall\Diamond m_1 = comiendo)$$
$$\forall\Box(m_1 = comiendo \rightarrow \forall\Diamond m_0 = comiendo)$$

Ya tenemos todo el sistema, incluidas las propiedades que queremos verificar, codificado. De esta forma nuestra herramienta de verificación automática podrá manejarlo.

## 3.3. Abstracción del modelo

Para abstraer los modelos podemos garantizar ciertas propiedades siempre y cuando se cumplan algunas condiciones. Por ejemplo, las propiedades de seguridad se preservan entre los modelos abstractos y concretos siempre y cuando el resultado de los cálculos sobre la concreción de los objetos abstractos está incluído en la concreción del resultado abstracto calculado (usando las versiones abstractas de los operadores). La idea fundamental en el *model checking* abstracto es la de construir descripciones de objetos concretos que imiten el efecto de las operaciones concretas con operaciones abstractas sobre las descripciones que sean *seguras* y por tanto adecuadas.

Vamos a empezar definiendo lo que entenderemos como el modelo abstracto, que no será otro que una versión abstracta de una estructura de *Kripke*. La idea es la de definir una función que relacione la estructura de *Kripke* concreta con la versión abstracta: $\xi \subseteq KS \times KS_\alpha$ ($KS$ es el conjunto de estructuras de *Kripke* mientras que $KS_\alpha$ es el conjunto de estructuras de *Kripke* abstractas), pero asegurando que las propiedades CTL$^*$ se preservan, al menos de forma débil:

$$\forall K \in KS, A \in KS_\alpha, (\xi(K, A) \Rightarrow \forall \phi \in CTL^*(K \models \phi \Leftarrow A \models \phi))$$

Para definir la relación $\xi$, vamos a definir una relación $\rho$ entre estados (concretos) de $K$ y estados (abstractos) de $A$. De esta forma, podremos construir el modelo abstracto imitando la semántica concreta de un determinado lenguaje de programación ya que, normalmente, esta semántica se da en términos de transición entre estados concretos (y cada estado concreto puede verse como una evaluación de valores de las variables).

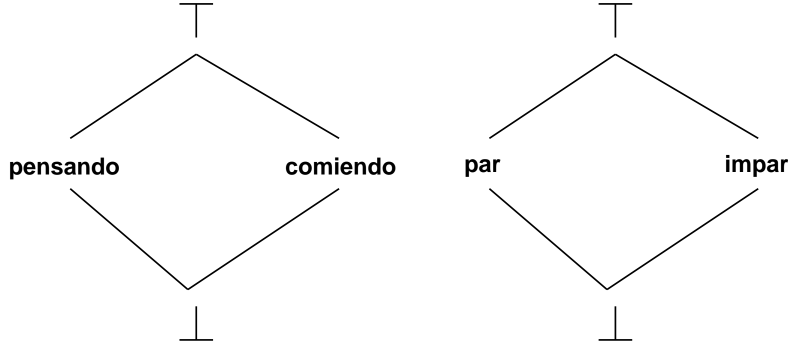
Con este objetivo en mente, transformamos la propiedad descrita anteriormente en términos de estados de las dos estructuras relacionadas:

$$\forall c \in S, a \in S_\alpha, (\rho(c, a) \Rightarrow \forall \phi \in CTL^*((K, c) \models \phi \Leftarrow (A, a) \models \phi))$$

siendo $S$ el conjunto de estados del modelo concreto $K$ y $S_\alpha$ el conjunto de estados del modelo abstracto $A$. Esta relación se integra dentro de un marco con una conexión de Galois $(\alpha, \gamma)$, por lo que asumimos que $S_\alpha$ es un retículo completo con un orden de aproximación $\sqsubseteq$ y que tenemos una inserción de Galois $(\alpha, \gamma)$ de $(\wp(S), \subseteq)$ a $(S_\alpha, \sqsubseteq)$. Recordemos que $a \sqsubseteq b$ si y sólo si $\gamma(a) \subseteq \gamma(b)$. El hecho de que consideremos como universo concreto $\wp(S)$ y no $S$ es un aspecto meramente técnico. En realidad el universo continuará siendo $S$, pero en la relación usamos el conjunto potencia para de esta forma tener de forma automática un retículo completo bajo inclusión de conjuntos como vimos en el tema anterior.

Observemos ahora el ejemplo que hemos introducido. Debido al uso de la variable de control entera, el espacio de estados del sistema se convierte en infinito. Una de las formas de manejar dicho espacio de estados de forma efectiva es mediante la interpretación abstracta. Vamos a definir el dominio abstracto que usaremos para este ejemplo concreto.

**Ejemplo 7 (continuación ejemplo matemáticos)** *Los dos valores posibles del estado de los matemáticos vamos a representarlos igual que en el mundo concreto, es decir mediante los valores* comiendo *y* pensando*. Sin embargo, los valores de la variable de control vamos a agruparlos en dos valores abstractos* par *e* impar*. De esta forma, si conseguimos relacionar bien el mundo concreto con el abstracto, conseguiremos directamente pasar de un universo de estados infinito, a uno finito. Además de los valores par e impar, tendremos un* $\top$ *y* $\bot$*.*




*Para relacionar lo abstracto y lo concreto, se definen las funciones de abstracción y concreción $\alpha$ y $\gamma$:*

$$\alpha(n) \begin{cases} par & \text{si } n \, mod \, 2 = 0 \\ impar & \text{si } n \, mod \, 2 \neq 0 \end{cases}$$

$$\gamma(par) = \{2, 4, 6, 8, \ldots\}$$
$$\gamma(impar) = \{1, 3, 5, 7, \ldots\}$$

Como decíamos antes, ahora tenemos un espacio de estados finito:

$$\Sigma_\alpha = \{pensando, comiendo, \top\}^2 \times \{par, impar, \top\}$$

### 3.3.1. Preservación de resultados

Tradicionalmente, el *model checking* abstracto se ha centrado en la preservación de propiedades de seguridad universales. Sin embargo, es posible construir modelos que preserven propiedades algo más ambiciosas, o al menos con una naturaleza distinta.

Un resultado de preservación relaciona ciertas propiedades entre objetos de alguna forma correspondientes, las propiedades que se satisfacen para un objeto y otro. Por ejemplo, si tenemos dos estructuras isomorfas, satisfarán las mismas propiedades de la lógica de primer orden. La correspondencia entre las estructuras será una relación de equivalencia. Además, la relación entre la satisfacción de fórmulas es una bi-implicación.

En particular, en el *model checking* abstracto podemos tener preservación *débil* o preservación *fuerte*. En la preservación débil toda descripción de un elemento concreto debe proporcionar información fiable acerca del elemento concreto:

$$\forall c \in C, a \in A \ (\alpha(c) = a \Rightarrow \forall \phi \in L(c \models \phi \Leftarrow a \models^\alpha \phi))$$

siendo $L$ la lógica en la que se expresan las propiedades.

Por otro lado, con la preservación fuerte, además de conservar las propiedades que se satisfacen, también se conservan las propiedades que son falsas. Hay dos tipos de conservación fuerte:

$$\forall c \in C, a \in A \ (\alpha(c) = a \Rightarrow \forall \phi \in L(c \models \phi \Leftrightarrow a \models^\alpha \phi))$$

denota una relación *buena*, mientras que

$$\forall c \in C, a \in A \ (\alpha(c) = a \Leftrightarrow \forall \phi \in L(c \models \phi \Leftrightarrow a \models^{\alpha} \phi))$$

denota una relación *adecuada*.

Obviamente, con una conservación fuerte, el nivel de simplificación (abstracción) del modelo dependerá en gran medida del conjunto de propiedades de $L$ ya que sólo las propiedades que no estén en dicho conjunto podrán ser *abstraidas*.

### 3.3.2. Estructura de *Kripke* abstracta

Supongamos que tenemos un conjunto $S_\alpha$ de estados abstractos, junto con una inserción de Galois $(\alpha, \gamma)$ que especifica la conexión entre el mundo concreto y el mundo abstracto. Para definir una *Estructura de Kripke abstracta* necesitamos además:

- Una función $L_\alpha$ que especifique la interpretación de los literales sobre los estados abstractos
- Un conjunto $I_\alpha$ de estados iniciales abstractos
- Una relación de transición $R_\alpha$ entre estados abstractos

A continuación describiremos estos tres puntos con más detalle ya que de ellos depende en parte la precisión de nuestro modelo abstracto. Pero antes vamos a introducir el concepto de concreción de una secuencia en una estructura de *Kripke*.

**Definición 3** *Para una secuencia de estados* $s_\alpha = a_0 a_1 \ldots$ *con* $a_i \in S_\alpha$, *definimos* $\gamma(s_\alpha) = \{c_0 c_1 \ldots | \forall i R(c_i, c_{i+1}) \wedge c_i \in \gamma(a_i)\}$

**Interpretación.** Para satisfacer la propiedad acerca de la preservación de propiedades descrita anteriormente, se tiene que cumplir que para todo literal $p$, $(A, a) \models p \Rightarrow (K, \gamma(a)) \models p$. Es decir, que si el literal se satisface en el modelo abstracto, entonces la concreción correspondiente a dicho modelo satisface también dicho literal. Para poder tener el mayor número de literales que se satisfagan en el modelo abstracto (de esta forma podremos probar más propiedades), definimos la función $L_\alpha$ que da la interpretación sobre el modelo abstracto como:

**Definición 4** *Para* $p \in Lit, L_\alpha(p) = \{a \in S_\alpha | \gamma(a) \subseteq L(p)\}$

En este caso estamos usando la función de etiquetado con la notación alternativa, es decir, relacionando las proposiciones con los estados en vez de los estados con las proposiciones, pero como es bien sabido, ambas representaciones son equivalentes. La intuición tras esta definición es que el conjunto

de estados abstractos en los que un literal $p$ será satisfecho, será aquéllos cuyas (todas) concreciones satisfagan dicho literal $p$ en el modelo concreto.

A partir de esta definición podemos definir la relación $\models_\alpha$ y ver que para todo $a \in S_\alpha$ y $p \in Lit$, $(A, a) \models p \Leftrightarrow (K, \gamma(a)) \models p$.

Como último apunte de este punto haremos notar que, si $\gamma(a)$ contiene estados concretos, algunos que satisfacen $p$ y otros que satisfacen $\neg p$, entonces se dará el caso en el que ni $a \models p$, ni $a \models \neg p$. De hecho, $a \not\models p$ **no implica** que $a \models \neg p$ como ocurre en los marcos lógicos tradicionales. Es más, cuanto más precisa sea una descripción, más literales satisfará, o dicho al contrario, cuanto más abstracta sea una descripción, menos literales satisfará:

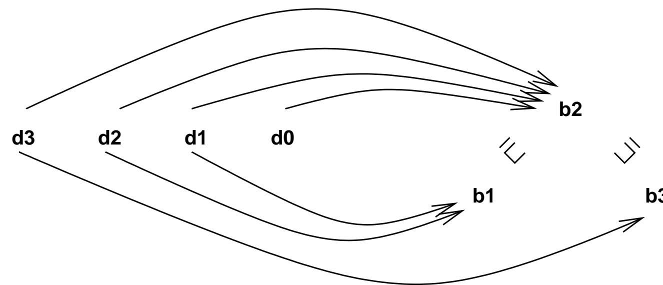
**Lema 1** *Sean $a, a' \in S_\alpha$, si $a' \sqsupseteq a$, entonces para todo $p \in Lit$, $a' \models p \Rightarrow a \models p$.*

**Estados iniciales.** Pasemos ahora a la definición de los estados iniciales del modelo abstracto. Para tener un modelo abstracto adecuado debemos tener en cuenta las condiciones de preservación que hemos mencionado al principio de esta sección. El conjunto de estados iniciales debe ser cuanto más pequeño mejor para mejorar la eficiencia del método, pero siempre deberá incluir al conjunto de estados iniciales del modelo concreto. La situación ideal sería la de definir $I_\alpha$ de tal forma que $I = \cup\{\gamma(a) | a \in I_\alpha\}$, es decir, que el conjunto de concreciones de los estados en $I_\alpha$ coincida con el conjunto de estados iniciales del modelo concreto. Sin embargo, esto en general no es posible pero sí que podemos definir $I_\alpha$ de forma que obtengamos el menor conjunto de estados posible:

$$I_\alpha = \{\alpha(c) | c \in I\}$$

Esta definición **no** es equivalente a $\alpha(I)$ debido a que la primera nos da en general un conjunto más preciso que la segunda. Vamos a ver un ejemplo que ilustra este aspecto:

**Ejemplo 8 (Selección de conjunto inicial)** *En la siguiente figura, asumamos que $I = \{d_3, d_2, d_1\}$.*



*Con la definición dada, tenemos* $I_\alpha = \{b_1, b_3\}$*. La concreción de este conjunto* $I_\alpha$ *es* $\{d_3, d_2, d_1\}$*. Sin embargo* $\alpha(I) = \{b_2\}$ *cuya concreción es* $\{d_3, d_2, d_1, d_0\}$ *y es, por lo tanto, menos preciso que el primer resultado.*

**Relación de transición abstracta** El único punto que nos falta definir de nuestro modelo abstracto es la relación de transición entre estados. Este es un punto crucial y apasionante ya que de él depende en gran medida el éxito o fracaso de la técnica abstracta de *model checking*. En este apartado debemos dar respuesta a la pregunta **¿Cuándo debe existir una transición entre el estado abstracto $a$ y el estado abstracto $b$ de nuestro modelo?**.

Para que las propiedades existenciales se preserven del modelo concreto al abstracto, la relación de transición abstracta debe definirse de forma que la existencia de una transición desde el estado abstracto $a$, implique que para todo estado $c \in \gamma(a)$, exista un sucesor de $c$ en el modelo concreto que satisfaga la propiedad considerada. Por otro lado, la preservación de propiedades universales, el hecho de que todo sucesor de $a$ satisfaga cierta propiedad, debe implicar que dicha propiedad se satisface para todo sucesor de $c \in \gamma(a)$.

Para que estas dos condiciones (preservación de propiedades existenciales y de propiedades universales) se satisfaciera, deberíamos exigir que la relación $R_\alpha$ fuera una *bisimulación*, lo que es un requisito demasiado costoso y que puede implicar que la reducción del espacio de búsqueda sea muy pequeña. Esta condición implicaría de forma automática que el modelo preservarıa las fórmulas CTL$^*$ de forma fuerte.

En lugar de imponer esta restricción tan exigente, lo que se hace es definir **dos** relaciones de transición en el modelo. Una de las relaciones preservará las condiciones existenciales, mientras que la otra preservará las condiciones universales. Por este motivo hemos definido en la parte inicial del tema los fragmentos $\forall$CTL$^*$ y $\exists$CTL$^*$, ya que es interesante distinguir las propiedades según esta clasificación para saber cuál de las dos transiciones del modelo abstracto debemos usar para determinar el valor de verdad de la propiedad. Sin embargo, esta solución es algo más débil que la mencionada anteriormente donde exigíamos bisimulación ya que ya no tenemos preservación fuerte para CTL$^*$. Puede darse el caso de que dada una propiedad $\phi$, ésta no se satisfaga en el modelo abstracto, pero que tampoco $\neg\phi$ se satisfaga en dicho modelo.

En las dos siguientes subsecciones vamos a describir con más detalle las dos relaciones de transición abstractas que definiremos en el modelo abstracto: la relación de transición *vinculada* y la relación de transición *libre*.

### 3.3.3. La relación de transición vinculada

Consideremos un estado abstracto $a \in S_\alpha$ que tiene un sucesor $b \in S_\alpha$ para el que se satisface la propiedad $\phi \in CTL^*$. Es decir, que $a \models \exists \bigcirc \phi$. Como tenemos que garantizar la preservación de la propiedad, tiene que ocurrir que todo estado concreto perteneciente al conjunto $\gamma(a)$, tiene que tener un sucesor para el que se satisfaga $\phi$.

Vamos a definir una serie de relaciones que serán útiles en esta sección:

**Definición 5** *Las relaciones* $R^{\exists\exists}, R^{\forall\exists} \subseteq \wp(A) \times \wp(B)$ *se definen como:*

- $R^{\exists\exists} = \{(X, Y) | \exists x \in X \exists y \in Y.R(X, Y)\}$
- $R^{\forall\exists} = \{(X, Y) | \forall x \in X \exists y \in Y.R(X, Y)\}$

Hay que notar que estas dos relaciones están definidas sobre superconjuntos, es decir, que devolverá un par de conjuntos que pertenecerán a la relación si existe un (para todo) elemento en $X$ que está relacionado mediante $R$ con un elemento de $Y$.

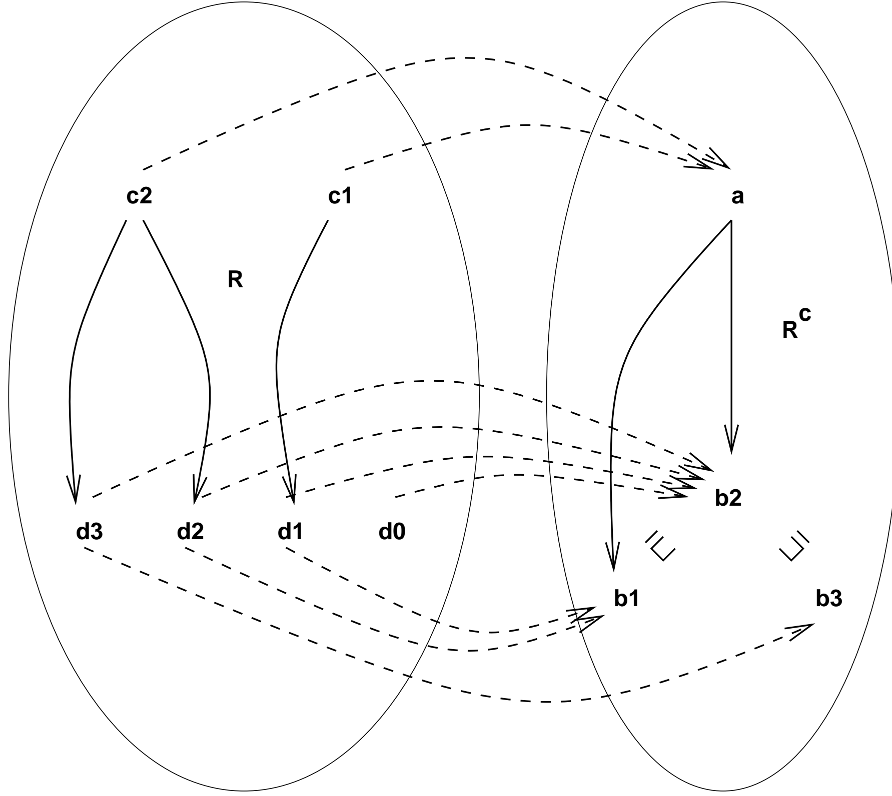
Usando esta definición y volviendo a la definición de la relación de transición vinculada, podemos decir que $b$ será un sucesor de $a$ sólo si $R^{\forall\exists}(\gamma(a), Y)$ para algún $Y \subseteq S$ cuya descripción (abstracción) es $b$, es decir, que $Y \subseteq \gamma(b)$. Con esta definición vemos que estamos garantizando que todo elemento descrito por $a$ estará relacionado con al menos un elemento de $Y$, que resulta ser un conjunto de elementos descritos por $b$, y por tanto que satisfacen la propiedad $\phi$.

Esta condición garantiza seguridad en el sentido de que las propiedades existenciales son conservadas entre ambos modelos. En cualquier caso, nos interesa también que $a$ tenga cuantos más sucesores mejor, por lo que cada uno de ellos debe ser la descripción más precisa posible de $Y$, por lo que elegiremos siempre el conjunto $Y$ mínimo y *la mejor* descripción $b$ de $Y$. Resumiendo, tenemos que

**Definición 6** *La relación de transición abstracta vinculada* $R^C_\alpha$ *se define como*

$$R^C_\alpha(a, b) \Leftrightarrow b \in \{\alpha(Y) | Y \in \min\{Y' | R^{\forall\exists}(\gamma(a), Y')\}\}$$

**Ejemplo 9 (Relación de transición vinculada)** *En la siguiente figura se representa la relación de transición vinculada definida a partir de un determinado mundo concreto. Los arcos dibujados con líneas continuas representan la relación de transición tanto en el mundo concreto como en el abstracto, mientras que los arcos dibujados con un trazo discontinuo representan las abstracciones de cada uno de los elementos concretos.*




*Puede verse como $\gamma(a) = \{c_1, c_2\}$, $\alpha(\{d_3\}) = b3$, $\alpha(\{d_2\}) = \alpha(\{d_1\}) = \alpha(\{d_2, d_1\}) = b1$ y que $\alpha(\{d_3, d_1\}) = \alpha(\{d_0\}) = b_2$. Del estado a podemos definir transiciones vinculadas a $b_1$ y a $b_2$. Desde el punto de vista de la preservación de propiedades, sería suficiente la transición a $b_1$. ¿Por qué?.*

Se cumple que si $a$ es un estado abstracto, y $c \in \gamma(a)$. Si $s_\alpha$ es un camino en la estructura abstracta sobre la relación vinculada $R^C_\alpha$, entonces existe un camino concreto $s$ en $\gamma(s_\alpha)$. Es decir, que si tenemos una traza en el modelo abstracto que sigue un camino con la relación vinculada, uno de los caminos del conjunto de caminos formado por la concreción de dicha traza abstracta se corresponde con una traza en el modelo concreto.

### 3.3.4. La relación de transición libre

Ahora consideremos un estado abstracto $a \in S_\alpha$ tal que todo sucesor suyo $b$ satisfaga $\phi$. Es decir, que $a \models \forall \bigcirc \phi$. Para satisfacer las restricciones sobre preservación, todo sucesor de todo estado concreto de $\gamma(a)$ debería satisfacer $\phi$. Esta cuestión puede girarse de forma que podemos decir que si algún elemento $c \in \gamma(a)$ tiene un sucesor que satisfaga $\phi$, entonces $a$ debe tener un sucesor que satisfaga $\phi$.

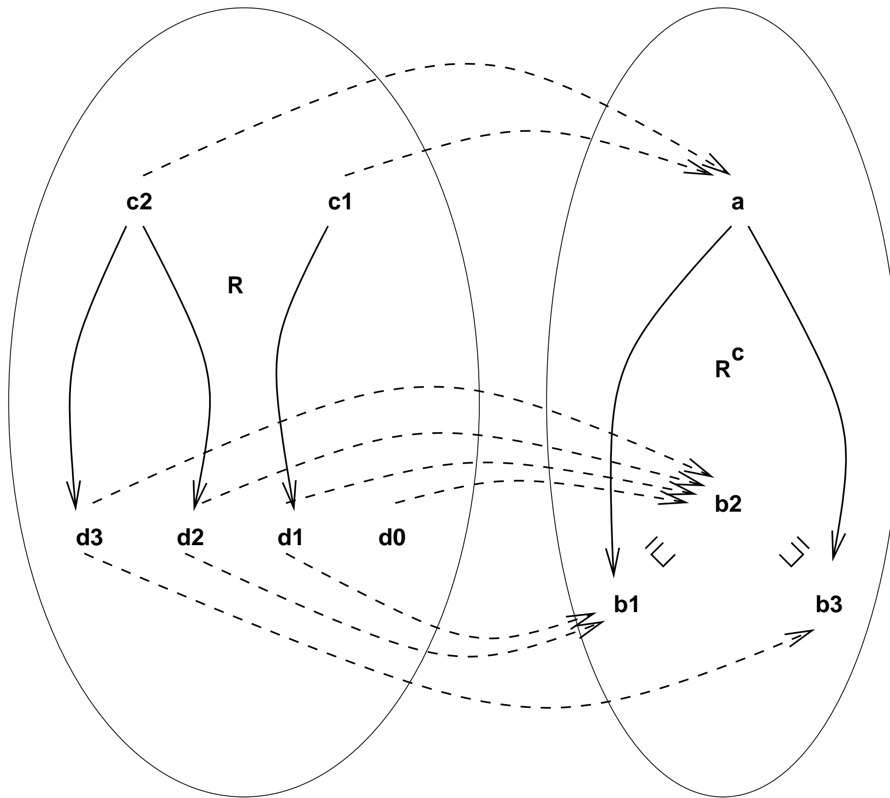
Usando la terminología que hemos empleado para el caso anterior, ppodemos imponer la condición de que $b$ debe ser un sucesor de $a$ si $R^{\exists\exists}(\gamma(a), Y)$

y $b$ es una descripción (abstracción) de $Y$. Esta condición hace que las propiedades universales se preserven. Además, en este caso nos gustaría que $a$ tuviera cuantos menos sucesores mejor pero que a la vez, cada uno de ellos sea una descripción de $Y$ lo más precisa posible. Para obtener este resultado elegiremos el mínimo $Y$ que satisfaga la condición y $b$ la mejor representación de dicho $Y$ mínimo.

**Definición 7 (Relación de transición abstracta libre)** *Decimos que la relación de transición libre se define como*

$$R_\alpha^F(a,b) \Leftrightarrow b \in \{\alpha(Y)|Y \in \min\{Y'|R^{\exists\exists}(\gamma(a),Y')\}\}$$

**Ejemplo 10 (Relación de transición libre)** *En la siguiente figura se representa la relación de transición libre definida a partir de un determinado mundo concreto. Como en el ejemplo anterior, los arcos dibujados con líneas continuas representan la relación de transición tanto en el mundo concreto como en el abstracto, mientras que los arcos dibujados con un trazo discontinuo representan las abstracciones de cada uno de los elementos concretos.*




*En la figura podemos observar que no existe una relación entre el elemento $a$ y $b_2$. Esto es debido a la exigencia de minimalidad de conjunto. Para cualquier estado abstracto $a \in S_\alpha$, el conjunto mínimo para el que*

$R^{\exists\exists}(\gamma(a), Y')$ *se cumple es un conjunto con un único elemento. En el ejemplo, esos* $Y'$ *son* $\{d_3\}$, $\{d_2\}$ *y* $\{d_1\}$, *siendo* $\alpha(Y')$ *respectivamente* $b_3$, $b_1$ *y* $b_1$.

Como para la relación anterior, también aquí tenemos una correspondencia entre caminos concretos y abstractos. Si $a \in S_\alpha$ y $c \in \gamma(a)$ y $s$ es un camino del modelo concreto, entonces existe un camino en la relación libre del modelo abstracto en cuya concreción está incluida la traza $s$.

Para terminar diremos que debido a la condición de minimalidad exigida para estas dos relaciones de transición, en general no tiene por qué satisfacerse que $R_\alpha^C \subseteq R_\alpha^F$ como podemos ver en los dos ejemplos anteriores.

### 3.3.5. Modelo abstracto final

Vamos a integrar los conceptos definidos hasta el momento acerca de la abstracción de modelos en la definición de la estructura de *Kripke* abstracta. Primero definiremos qué es un sistema de transición *mixto*:

**Definición 8 (Sistema de transición mixto)** *Un sistema de transición mixto es una tripla* $(S, F, C)$ *que consiste en un conjunto de estados* $S$ *y dos relaciones de transición* $F$ *y* $C$ *llamadas* liibre *(*free*) y* vinculada *(*constrained*) respectivamente.*

*Un* camino libre *será un camino cuyas transiciones están incluídas en la relación* $F$*, mientras que un* camino vinculado *tendrá sus transiciones en la relación* $C$.

*La interpretación de fórmulas CTL* sobre este tipo de estructura cambia ligeramente, ya que ahora decimos que:*

- $s \models \forall\psi$ *si y sólo si para todo caino libre* $\pi$, $\pi \models \psi$
- $s \models \exists\psi$ *si y sólo si existe un camino vinculado* $\pi$ *tal que* $\pi \models \psi$

Ahora podemos definir la noción de estructura de *Kripke* abstracta:

**Definición 9 (Estructura de Kripke abstracta)** *Una estructura de Kripke abstracta es una quintupla* $(S_\alpha, F, C, J, L_\alpha)$ *que representa un sistema de transición mixto con estados iniciales* $J$ *y función de interpretación* $L$*. La noción de alcanzabilidad se define según la unión* $F \cup C$ *a no ser que se diga lo contrario*

*Dada la estructura de Kripke* $K = (S, R, I, L)$ *y el conjunto de estados* $S_\alpha$, *la función de abstracción* $\alpha^M : KS \to KS_\alpha$ *mapea la estructura* $K$ *a la estructura abstracta* $K_\alpha = (S_\alpha, R_\alpha^F, R_\alpha^C, I_\alpha, L_\alpha)$ *donde* $R_\alpha^C$, $R_\alpha^F$, $I_\alpha$ *y* $L_\alpha$ *se definen como hemos descrito a lo largo de esta sección.*

Con esto, tenemos el resultado de que

**Teorema 1** *Para toda fórmula* $\phi \in CTL^*$, $\alpha^M(K) \models \phi \Rightarrow K \models \phi$

## 3.4. Abstracción de programas

Sabemos cómo generar un modelo abstracto a partir de uno concreto, pero en realidad a nosotros nos interesa construir un modelo abstracto directamente, sin tener que construir previamente el concreto. Para ello recurrimos a la abstracción de semánticas. De esta forma, en vez de ejecutar un programa de forma tradicional, se ejecutará un programa *de forma abstracta*, obteniendo así una traza abstracta. En este punto nos encontramos de nuevo con los puntos fijos. Querremos definir una semántica abstracta que imite de forma más precisa posible el comportamiento de la semántica concreta.

La idea fundamental es la de definir una semántica *no estándar* que imite (o represente) la semántica estándar. La semántica abstracta que podamos definir en cada momento dependerá principalmente del lenguaje de programación que se esté usando. Si la semántica concreta de un programa la representamos por la función $\mathcal{I}(P)$, tendremos que definir una relación $\sigma$ que obtenga la correspondiente semántica concreta $\mathcal{I}_\alpha(P)$. Si hablamos de la semántica de un programa definida por sus puntos fijos, podemos aprovecharnos del siguiente resultado:

**Lema 2** *Sea $(C, \sqsubseteq)$ un cpo, y $f : C \to C$ una función monótona. Sea $(A, \leq)$ un poset, y $f_\alpha : A \to A$ monótona y $(\alpha, \gamma)$ una conexión de Galois de $(C, \sqsubseteq)$ a $(A, \leq)$. Supongamos que*

$$\alpha \circ f \leq f_\alpha \circ \alpha$$

*entonces* $\alpha(\mathrm{lfp}(f)) \leq \mathrm{lfp}(f_\alpha)$

PROOF

$\alpha \circ f \leq f_\alpha \circ \alpha$

$\Rightarrow$ $\{\gamma$ es monótona$\}$

$\gamma \circ \alpha \circ f \circ \gamma \sqsubseteq \gamma \circ f_\alpha \circ \alpha \circ \gamma$

$\Rightarrow$ $\{\gamma \circ \alpha$ es extensiva, $\alpha \circ \gamma$ es reductora$\}$

$f \circ \gamma \sqsubseteq \gamma \circ f_\alpha$

$\Rightarrow$ $\{$orden de instanciación punto a punto$\}$

$f(\gamma(\mathrm{lfp}(f_\alpha))) \sqsubseteq \gamma(f_\alpha(\mathrm{lfp}(f_\alpha)))$

$\equiv$ $\{$por resultado $f(\mathrm{lfp}(f)) = \mathrm{lfp}(f)$ tenemos que $f_\alpha(\mathrm{lfp}(f_\alpha)) = \mathrm{lfp}(f_\alpha)\}$

$f(\gamma(\mathrm{lfp}(f_\alpha))) \sqsubseteq \gamma(\mathrm{lfp}(f_\alpha))$

$\Rightarrow$ $\{$se cumple que $\forall c \in C, f(c) \sqsubseteq c \Rightarrow \mathrm{lfp}(f) \sqsubseteq c\}$

$\mathrm{lfp}(f) \sqsubseteq \gamma(\mathrm{lfp}(f_\alpha))$

$\equiv$ $\{$ $(\alpha, \gamma)$ es una conexión de Galois$\}$

$\alpha(\mathrm{lfp}(f)) \leq \mathrm{lfp}(f_\alpha)$

Para ilustrar la forma de definir la semántica abstracta de un lenguaje de programación vamos a centrarnos en un lenguaje de programación concreto. En particular a un lenguaje donde se expresan las acciones que se

toman cuando determinadas condiciones se satisfacen. Vamos a fijarnos en el ejemplo de los matemáticos que hemos estado usando a lo largo del tema.

Si nos fijamos en nuestro sistema, se realizan tres operaciones con los enteros: la multiplicación por tres, la suma de uno y la división por dos pero vamos a definir el lenguaje de una forma algo más formal.

Un programa será un conjunto de acciones de la forma $c_i(\overline{x}) \rightarrow t_i(\overline{x}, \overline{x}')$, donde $y$ es un valor de un conjunto indexado $J$, $\overline{x}$ representa el vector de variables de un programa, $c_i$ es una condición que depende de los valores de las variables del programa, y $t_i$ representa la actualización de los valores de las variables del sistema. Un programa de esta forma se ejecuta eligiendo de forma no determinista entre una de las acciones para las que la condición $c_i$ se cumple, y actualizando los valores como viene especificado por $t_i$.

Llamaremos Val al conjunto de valores que puede tomar el vector $\overline{x}$ e IVal$\subseteq$Val será el conjunto de valores que pueden tener inicialmente. Así pues, cada $c_i$ es un predicado sobre Val y $t_i$ es una relación sobre Val$\times$Val.

Para este lenguaje, podemos definir la semántica estándar (concreta) como sigue

**Definición 10** *La función de interpretación concreta $\mathcal{I}$ : Lang $\rightarrow$ KS se define como sigue. Sea $P = \{c_i(\overline{x}) \rightarrow t_i(\overline{x}, \overline{x}')|i \in J\}$ en Lang, $I(P)$ es el sistema de transición $(S, I, R)$ donde:*

- $S=$ *Val*
- $I=$ *IVal*
- $R = \{(\overline{v}, \overline{v}') \in \mathrm{Val}^2 | \exists i \in J.c_i(\overline{v}) \wedge t_i(\overline{v}, \overline{v}')\}$.

Ahora asumamos que tenemos un dominio abstracto $\mathrm{Val}_\alpha$ relacionado con el concreto mediante una conexión de Galois $(\alpha, \gamma)$ de $(\wp(\mathrm{Val}, \subseteq)$ a $(\mathrm{Val}_\alpha, \sqsubseteq)$. Definimos entonces dos tipos de interpretaciones abstractas de cada $c_i$ y $t_i$ que se corresponderán con las transiciones vinculadas y libres del sistema.

**Definición 11** *Para $i \in J$, sea $c_i^F$, $c_i^C$ las condiciones sobre $\mathrm{Val}_\alpha$ y $t_i^F$ y $t_i^C$ las transformaciones sobre $\mathrm{Val}_\alpha \times \mathrm{Val}_\alpha$:*

- $c_i^F(a)$ *es una interpretación abstracta libre de $c_i$ si, y sólo si, para todo elemento $a \in \mathrm{Val}_\alpha$,*

$$c_i^F(a) \Leftrightarrow \exists \overline{v} \in \gamma(a).c_i(\overline{v})$$

- $t_i^F(a)$ *es una interpretación abstracta libre de $t_i$ si, y sólo si, para todo par de elementos $a, b \in \mathrm{Val}_\alpha$,*

$$t_i^F(a, b) \Leftrightarrow b \in \{\alpha(Y)|Y \in \min\{Y'|t_i^{\exists\exists}(\gamma(a), Y')\}\}$$

- $c_i^C(a)$ *es una interpretación abstracta vinculada de* $c_i$ *si, y sólo si, para todo elemento* $a \in \mathrm{Val}_\alpha$*,*

$$c_i^C(a) \Leftrightarrow \forall \overline{v} \in \gamma(a).c_i(\overline{v})$$

- $t_i^C(a)$ *es una interpretación abstracta vinculada de* $t_i$ *si, y sólo si, para todo par de elementos* $a, b \in \mathrm{Val}_\alpha$*,*

$$t_i^C(a,b) \Leftrightarrow b \in \{\alpha(Y) | Y \in \min\{Y' | t_i^{\forall\exists}(\gamma(a), Y')\}\}$$

Para acabar, definimos la interpretación abstracta $\mathcal{I}_\alpha(P)$ de la semántica de un programa $P$ como el sistema $\widehat{A^M} = (S_\alpha, \widehat{R_\alpha^F}, \widehat{R_\alpha^C}, I_\alpha)$ donde:

- $S_\alpha = \mathrm{Val}_\alpha$
- $\widehat{R_\alpha^F} = \{(a,b) \in \mathrm{Val}^2 | \exists i \in J.c_i^F(a) \wedge t_i^F(a,b)\}$.
- $\widehat{R_\alpha^C} = \{(a,b) \in \mathrm{Val}^2 | \exists i \in J.c_i^C(a) \wedge t_i^C(a,b)\}$.
- $I_\alpha = \{\alpha(\overline{v}) | \overline{v} \in \mathrm{IVal}\}$

$\widehat{R_\alpha^F}$ y $\widehat{R_\alpha^C}$ son las llamadas *relaciones de transiciones computadas libre y vinculada* respectivamente.

### 3.4.1. Ejemplo de los matemáticos: Final

Resumiendo lo que tenemos hasta ahora, tenemos la especificación del sistema y la definición del dominio abstracto, así como la inserción de Galois. Vamos a definir ahora las relaciones de transición del sistema mediante la definición de los operadores del lenguaje.

Nuestro dominio de estados abstractos es, recordemos

$$\Sigma_\alpha = S_\alpha = \{pensando, comiendo, \top\}^2 \times \{par, impar, \top\}$$

Los estados iniciales del sistema serán

$$I_\alpha = \{\langle pensando, pensando, par\rangle, \langle pensando, pensando, impar\rangle\}$$

Notese que este conjunto de estados se corresponde con la definición formal dada: $I_\alpha = \{\alpha(c) | c \in I\}$.

Las siguientes tablas nos dan la interpretación abstracta libre para el ejemplo. Tenemos que considerar tanto las actualizaciones como las posibles comprobaciones de guardas que se hacen en el programa.

Como ejemplo de interpretación de las tablas, diremos que la entrada *false* en la siguiente tabla, fila $+1^F$, columna *(par,par)* significa que $+1^F$(par,par) es falseo, es decir, que para cualquier conjunto minimal $Y$ tal que $+1^{\exists\exists}(\gamma(par), Y)$, tenemos que $\alpha(Y) \neq par$.

| FREE | (*par*,*par*) | (*par*,*impar*) | (*par*,⊤) | (*impar*,*par*) | (*impar*,*impar*) | (*impar*,⊤) | (⊤,*par*) | (⊤,*impar*) | (⊤,⊤) |
|---|---|---|---|---|---|---|---|---|---|
| $3*^F$ | *true* | *false* | *false* | *false* | *true* | *false* | *true* | *true* | *false* |
| $+1^F$ | *false* | *true* | *false* | *true* | *false* | *false* | *true* | *true* | *false* |
| $/2^F$ | *true* | *true* | *false* | *false* | *false* | *false* | *true* | *true* | *false* |

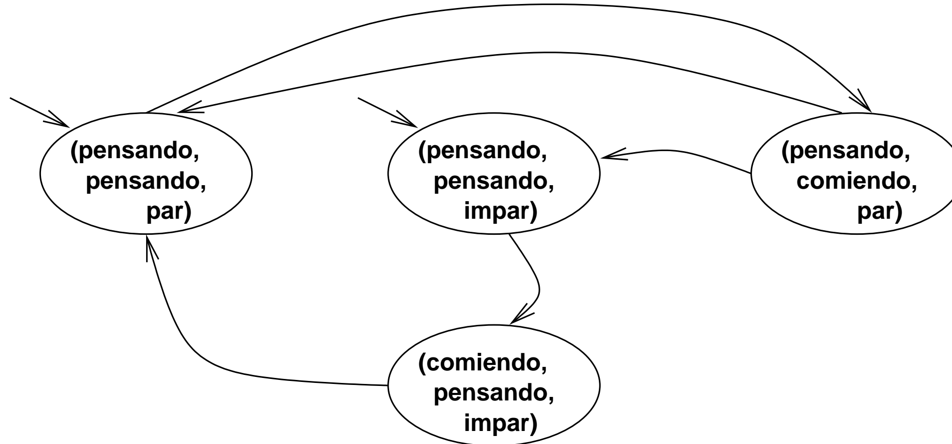
La entrada *true* de la siguiente tabla, fila $par^F$, columna *par* nos indica que $even^F(par)$ es cierta, es decir, que $\exists n \in \gamma(par).par(n)$.

| FREE | *par* | *impar* | ⊤ |
|---|---|---|---|
| $par^F$ | *true* | *false* | *true* |
| $impar^F$ | *false* | *true* | *true* |

| FREE | *pensando* | *comiendo* | ⊤ |
|---|---|---|---|
| $(=pensando)^F$ | *true* | *false* | *true* |
| $(=comiendo)^F$ | *false* | *true* | *true* |

Con este ejemplo vemos también que cuando abstraemos funciones, éstas pueden dejar de ser funciones pasando a ser relaciones como ocurre para $/2^F$: ante una misma entrada podemos obtener dos resultados distintos.

Usando las tablas definidas para las abstracciones libres, podemos dibujar el modelo abstracto libre a partir del programa. Es decir, construiremos la semántica no estándar definida por las funciones de las tablas (no a partir del modelo concreto). La siguiente figura muestra el modelo obtenido:




Para ver la utilidad de las <u>versiones vinculadas</u> de abstracción tenemos que modificar ligeramente el problema. Vamos a añadir un tercer proceso concurrente que puede *resetear* el sustema asignando el valor 100 a la variable $n$. Esta acción puede llevarse a cabo sólo cuando los dos matemáticos estén pensando. Para especificar dicho comportamiento, añadimos al programa la siguiente acción:

$$m_0 = pensando, m_1 = pensando \rightarrow n := 100$$

En este nuevo programa, queremos comprobar que a lo largo de todo camino, en todo estado existe un camino sucesivo que alcanza un estado de *reseteo*. Si denoramos como *reset* la condición $l_0 = pensando \wedge l_1 = pensando \wedge n = 100$, la propiedad en CTL* se escribiría como:

$$\forall\Box\exists\Diamond reset$$

Ahora tenemos que extender el dominio abstracto con el valor 100, de forma que $\gamma(100) = \{100\}$. Como la fórmula que queremos verificar no es del fragmento ∃CTL* ni de ∀CTL*, sino que pertenece a CTL*, necesitaremos un sistema de transición mixto, ya no nos valdrá con la versión libre únicamente.

En las siguientes tablas damos la interpretación necesaria para la construcción del modelo que representa la semántica abstracta. Las tablas de las versiones libres deben ser extendidas para considerar el nuevo valor 100, pero la extensión es trivial, por lo que no incluiremos dicha información.

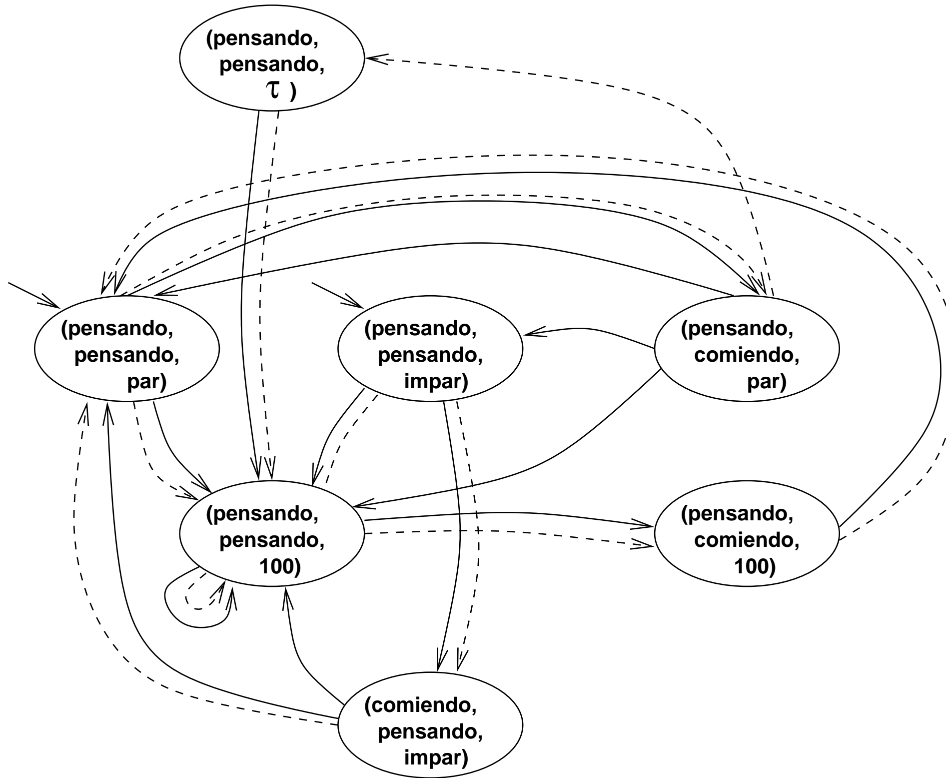
| CONSTR. | *(par,100)* | *(par,par)* | *(par,impar)* | *(par,⊤)* | *(100,100)* | *(100,par)* | *(100,impar)* | *(100,⊤)* |
|---|---|---|---|---|---|---|---|---|
| $/2^C$ | *false* | *false* | *false* | *true* | *false* | *true* | *false* | *false* |

| CONSTR. | *(impar,100)* | *(impar,par)* | *(impar,impar)* | *(impar,⊤)* |
|---|---|---|---|---|
| $3*^C$ | *false* | *false* | *true* | *false* |
| $+1^C$ | *false* | *true* | *false* | *false* |

| CONSTR. | *100* | *par* | *impar* | ⊤ |
|---|---|---|---|---|
| $even^C$ | *true* | *true* | *false* | *false* |
| $odd^C$ | *false* | *false* | *true* | *false* |

| CONSTR. | *pensando* | *comiendo* | ⊤ |
|---|---|---|---|
| $= pensando^C$ | *true* | *false* | *false* |
| $= comiendo^C$ | *false* | *true* | *false* |

La siguiente figura muestra la estructura de *Kripke* abstracta *relevante*, es decir, los estados alcanzables:




## 4. Ejercicios

Responde a las siguientes cuestiones. Puedes incluir información mencionada en clase aunque no esté contenida en el texto. Las preguntas pueden no estar ordenadas.

1. ¿En qué consiste un modelo abstracto según el trabajo de Dams?.

2. ¿Qué es la preservación débil de resultados? ¿y la preservación fuerte?.

3. ¿Qué tiene que cumplir una fórmula para pertenecer a la lógica $\forall$CTL$^*$?.

4. ¿Por qué definimos dos sistemas de transiciones en un modelo abstracto?.

5. Intuitivamente: ¿Qué nos aporta la abstracción de programas con respecto a la abstracción de modelos?.

6. La aproximación presentada, ¿es capaz de responder satisfactoriamente siempre cualquier propiedad CTL$^*$?.

7. En la práctica, ¿cómo relacionados los mundos concreto y abstracto?.

8. ¿Qué diferencia fundamental en cuanto a funcionalidad aportada existe entre el *model checking* abstracto y otras técnicas de optimización de *model checking*?.

# Referencias


[1] D. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. PhD Dissertation, Eindhoven University of Technology, July 1996.

[2] D. Dams, R. Gerth, O. Grumberg *Abstract interpretation of reactive systems* ACM Transactions on Programming Languages and Systems (TOPLAS), 19(2), 1997.

[3] E.M. Clarke, O. Grumberg, D.E. Long *Model Checking and Abstraction*, ACM Transactions on Programming Languages and Systems (TOPLAS), 16(5):1512-1542, 1994.