Introducción al Modelado de Sistemas
Capítulo 2

Camilo Rueda

20 de enero de 2016

1. La herramienta Rodin

Vimos en el capitulo anterior la forma de definir el modelo de un sistema usando el lenguaje de EventB. Un modelo de esta naturaleza se puede escribir convenientemente usando el software *Rodin*. Este tipo de software se denomina en computadión un "marco de trabajo" (en ingls, "framework"). Es decir, un conjunto de facilidades para ayudar a definir de manera precisa el modelo de un sistema en un lenguaje dado. Estos marcos de trabajo consisten de una colección de *herramientas*, cada una de las cuales proporciona al disñador del modelo facilidades de un cierto tipo. Un ejemplo de herramienta es un *editor*, que es un programa que facilita escribir texto en un determinado lenguaje, señalando los posibles errores de escritura que se cometan. Cuando se lanza el software *Rodin*, se obtiene una ventana como la de la figura 1.

La ventana se compone de varias partes. La parte de la izquierda sirve para exhibir una serie de carpetas, una por cada modelo de sistema que se haya escrito. Como no se ha escrito hasta ahora ninguno, aparece en blanco. La flecha roja señala un icono, que es el que se usa para definir nuevos sistemas. En *Rodin* los modelos de los sistemas se denominan *Proyectos*. Al presionar el icono indicado por la flecha roja se abre un diálogo que solicita dar un nombre al proyecto. Si escribimos "bicicleta" y presionamos el botón "finish", obtendremos en la parte izquierda de la ventana una carpeta de nombre "bicicleta". Esta carpeta será la que contenga la definición del modelo.

Enseguida seleccionamos esa carpeta y después presionamos el icono justo a la derecha del indicado por la flecha. Obtenemos un diálogo que pide el nombre de un "componente". Escrbimos allí el nombre "biciCtx". Como vimos antes, hay dos clases de componentes: contextos y máquinas. Seleccionamos el botón "Context" y presionamos luego "finish". Obtenemos en la parte central de la ventana un contexto, que por ahora está vacío (ver figura 2).

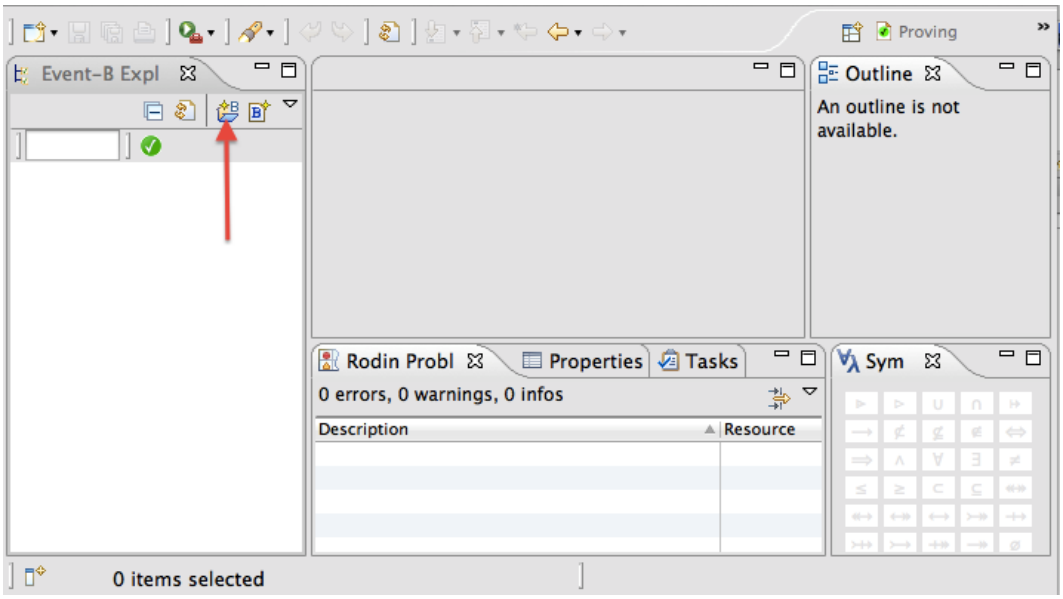

Figura 1: Ventana principal de Rodin: crear un nuevo proyecto

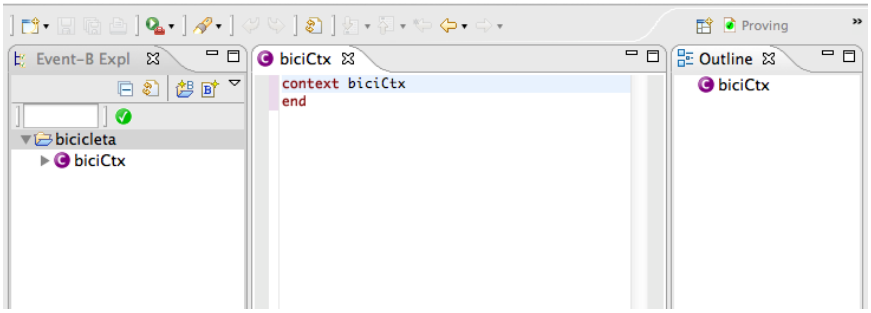

Figura 2: Agregar componente

1.1. Describir el modelo

La ventana central es un editor en el que podemos escribir la descripción del contexto. Escribimos entonces el contexto de la bicicleta que vimos en el capítulo anterior, como se muestra en la figura 3. En la figura puede verse que el texto del contexto aparece en la ventana central y que en la ventana inferior derecha (marcada "Sym") se muestran todos los símbolos del lenguaje EventB. Cuando se escribe el texto de algún componente del sistema se puede presionar alguno de estos símbolos para que aparezca en el texto que se escribe. Se puede notar también que *Rodin* traduce automáticamente la notación

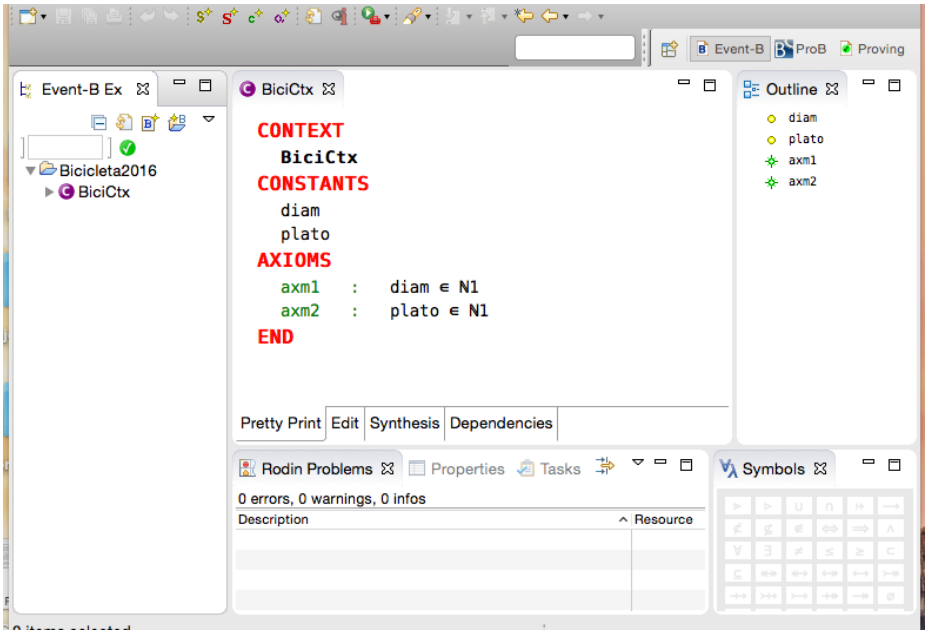

Figura 3: Primer componente, el contexto

"$diam : NAT1$" a su equivalente en notación matemática, "$diam \in \mathbb{N}_1$". La ventana central inferior es importante. Allí se muestran posibles errores en el texto del modelo, que es siempre necesario corregir. En la ventana de la izquierda, si se abre la pestaña, se muestra el nombre de cada componente del modelo. En el ejemplo, aparece el componente "biciCtx", que es el único que se ha definido hasta ahora.

El siguiente paso es crear un nuevo componente para escribir el texto de la máquina (que incluye las variables y los eventos). Como en el caso del contexto, se presiona el icono de creación de un nuevo componente. En el diálogo que aparece se le da un nombre al componente, por ejemplo "biciMaq", se selecciona el botón "machine" y se presiona "finish". El modelo se muestra como en la figura 4. Este es un esqueleto que se debe completar. Lo completamos con la definición de la máquina que se mostró en el capítulo

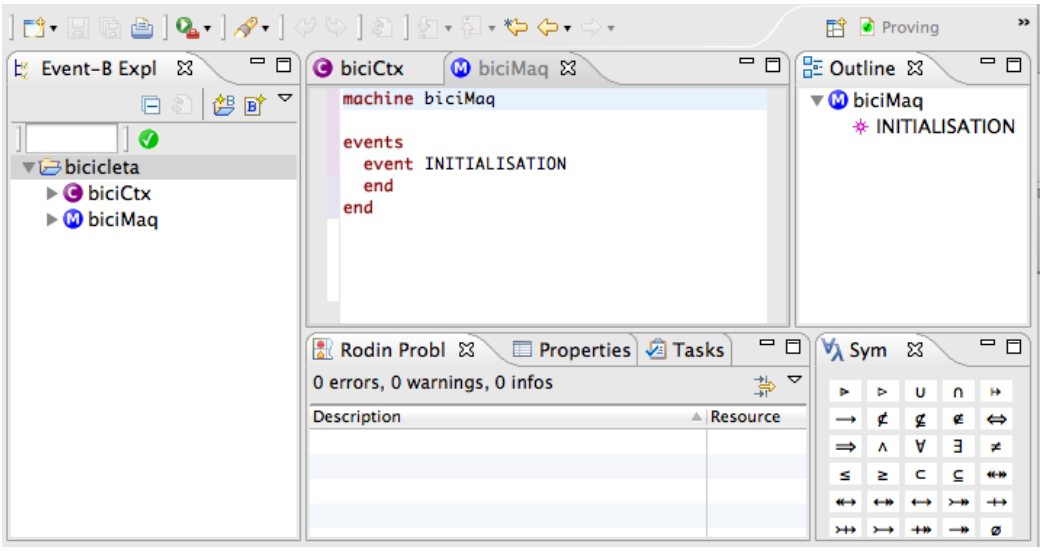

Figura 4: Segundo componente, la máquina

1. Se puede notar que lo primero es agregar en la primera línea la especificación de que la máquina "ve" (en ingles, "sees") el contexto "biciCtx", como se muestra en la figura 5. En el invariante @*inv*3 se ve que la variable cambio se ha definido como un número natural, sin el cero. La razón es que la variable *cambio* aparece en el denominador de una expresión, luego no puede ser igual a cero.

1.2. Comportamiento del modelo

Una vez se construye un modelo de un sistema, una manera simple de experimentarlo es observar cómo evoluciona. Los modelos que se construyen en el lenguaje de eventB son *Sistemas discretos de transición de Estados*. Es decir, son modelos que representan la manera en que van cambiando las observaciones a medida que ocurre cada evento. Un *estado* de un sistema corresponde a una observación. Cada estado representa entonces los valores que tienen las variables del modelo en un momento dado. Cuando ocurre un evento se produce una *transición*. La transición consiste en pasar de un estado a otro.

Los estados y transiciones del sistema se pueden representar en un gráfico en el que cada observación se encierra en un círculo y dos círculos de estos, por ejemplo, C_1 y C_2, se unen con una flecha cuando, después de la ocurrencia de un evento, la observación en C_1 se transforma en la observación de C_2. Cada flecha en el gráfico se decora con el nombre del evento que produce el correspondiente cambio de estado. Para el ejemplo del modelo de la bicicleta, este gráfico podría ser el de la figura 6. En este gráfico, por ejemplo, el evento *pedalear* es el que causa la transición del *estado*1 al *estado*2. En este caso el gráfico es infinito porque no hay restricciones sobre el número de pedalazos que se puede dar.

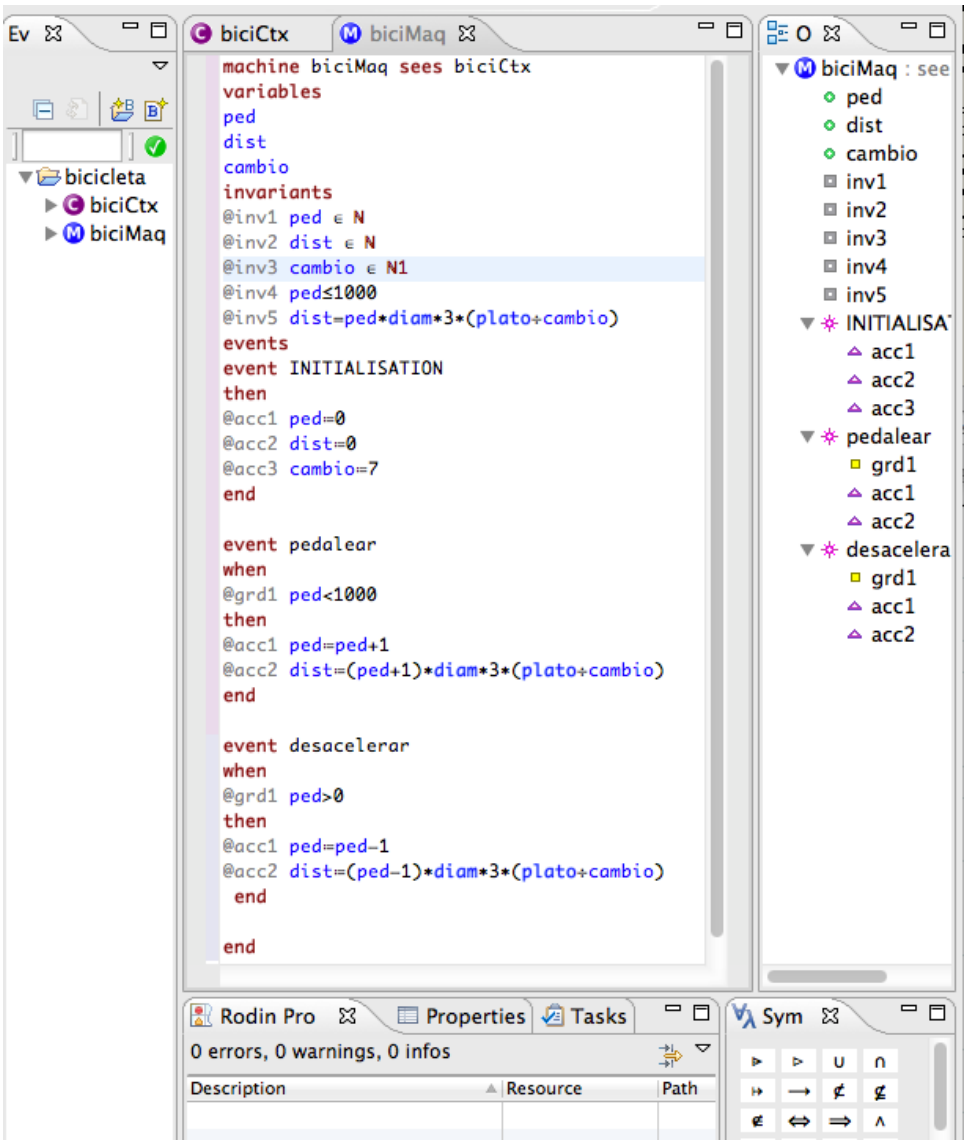

Figura 5: El texto de la máquina

Note que hay ciclos en el gráfico porque, partiendo de un estado, si se pedalea y luego se desacelera, se llega al mismo estado.

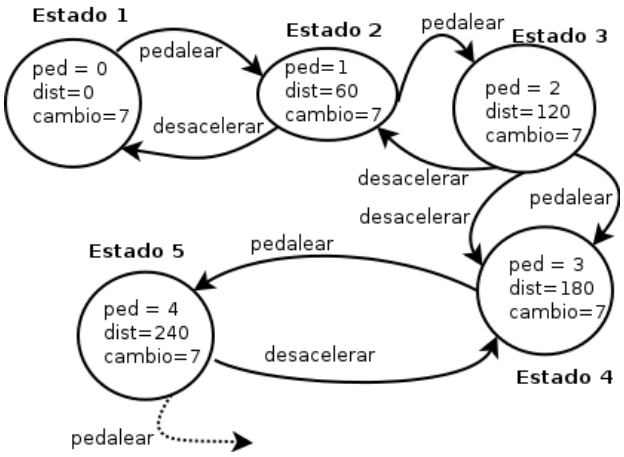

Figura 6: Gráfico de transición de estados

El comportamiento de un sistema está dado por su gráfico de transición de estados. Como este gráfico puede ser muy grande (o infinito), para observar el comportamiento se recorre alguna porción del gráfico, por ejemplo, la que corresponda a la ejecución de la secuencia de eventos $pedalear, pedalear, pedalear, desacelerar$, que recorrería los estados $1, 2, 3, 4, 3$. Hacer este tipo de recorridos sobre el sistema de transición de estados de un sistema se denomina *animar* el modelo.

El software *Rodin* tiene una herramienta de animación que se llama *ProB*. Esta herramienta permite conocer las observaciones sucesivas que se obtienen con la ejecución de uno o más eventos del modelo. Permite también verificar que las propiedades del modelo (axiomas e invariantes) se cumplen en todos los recorridos del gráfico de transición de estados. Para animar una máquina en *Rodin*, se presiona el click derecho (o la tecla "ctrl" y el click, en un Mac) sobre el nombre del componente en la ventana izquierda, y luego se selecciona, en el menú que se obtiene, la opción "Start Animation/ Model Checking", como se ve en la figura 7.

La ventana de animación de *ProB* se muestra en la figura 8. En la ventana superior izquierda aparecen los eventos *pedalear* y *desacelerar*, precedidos de un círculo rojo. Esto significa que esos eventos no están habilitados. Es decir, las condiciones de estos eventos no se cumplen. La razón es que todavía no se saben los valores de las variables del modelo. Cuando se empieza una animación, lo primero que se debe hacer es calcular el estado inicial, es decir, lanzar el evento de inicialización de las variables. Antes de hacer esto, *ProB* solicita escoger algún valor para las constantes, mediante el comando *SETUP_CONTEXT* que aparece en esa ventana. Dando click derecho sobre ese comando aparece un menú con parejas de valores, uno para cada constante. Basta seleccionar alguno para comenzar.

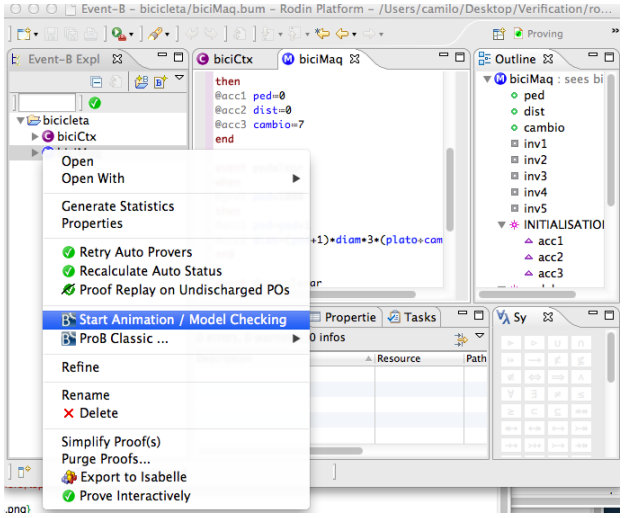

Figura 7: "Animar" una máquina

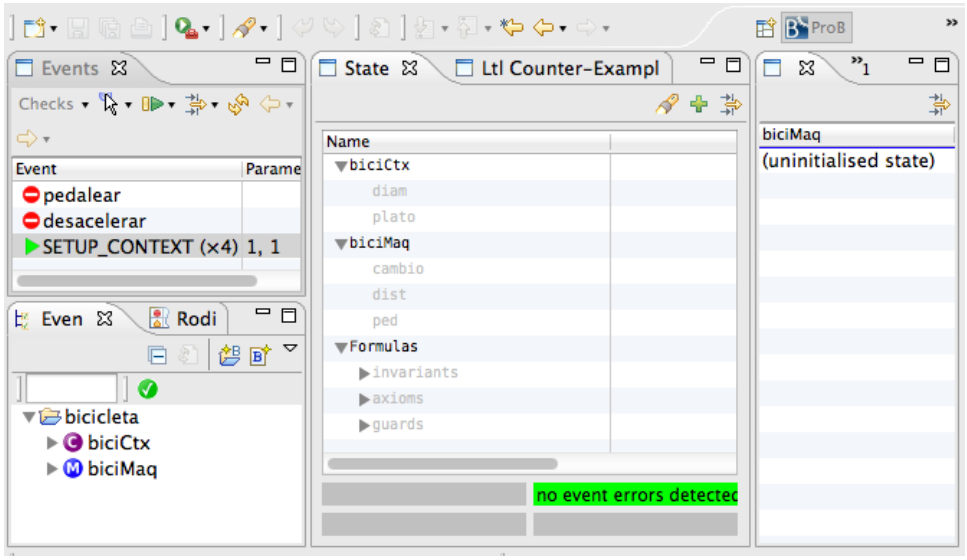

Figura 8: La ventana de *ProB*

Aparece en la ventana el evento de inicialización. Dando doble-click en él, se lanza este evento, lo que ocasiona que las variables tengan los valores que esa inicialización especifica en el modelo.

De ahí en adelante se puede ver que uno o más eventos aparecen con una pestaña verde a la izquierda. Esto quiere decir que están habilitados porque en el estado actual (es decir, con los valores actuales de las variables) sus condiciones se cumplen. Hacer doble-click en un evento dispara su ejecución. La observación que resulta se puede ver en la ventana del centro, en el que aparece cada variable, junto con el valor que tiene después de lanzar el evento y el que tenía antes de lanzarlo. La ventana central inferior es muy importante. Cuando están en verde las dos partes ("invariant ok" y "no event errors detected") significa que las propiedades (invariantes) del modelo se cumplen para la observación actual. Es decir, siempre deben quedar en verde. Si no lo están, hay un problema con el modelo, que debe corregirse. Después de ejecutar una vez el evento *pedalear*, la ventana que se muestra es la de la figura 9. Note que se escogieron valores de 1 y de 8 para *diam* y para *plato*.

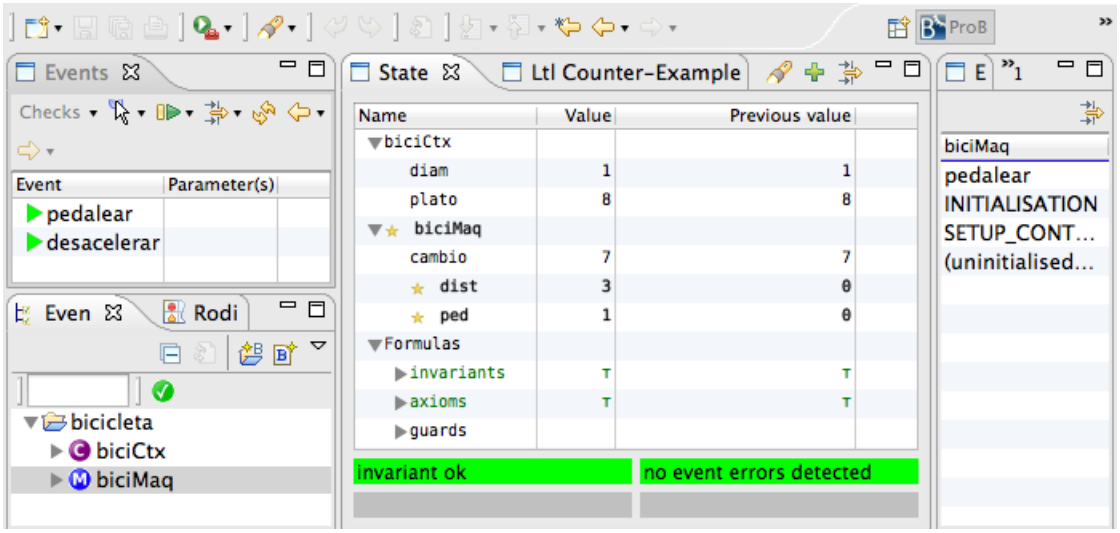

Figura 9: Lanzar el evento *pedalear*

En síntesis, el proceso para construir un modelo correcto de un sistema comprende dos pasos. Primero, definirlo con dos componentes, un contexto y una máquina, y escribirlo en el editor de *Rodin*. Segundo, analizar su comportamiento usando *ProB*.

La verificación que hace *ProB* de las propiedades del modelo consiste en comprobar si en toda observación se cumplen esas propiedades. La primera observación es la inicialización. Los valores iniciales de las variables deben obedecer todas las propiedades definidas en los invariantes. Luego la verificación se realiza cada vez que una observación cambia. Es decir, cada vez que se ejecuta un evento. La acción del evento cambia los valores de las variables. Los nuevos valores deben entonces obedecer tambin las propiedades. Para ilustrar esta

comprobación en *ProB*, haremos deliberadamente un cambio en el evento *pedalear* que vuelva erróneo el modelo:

event *pedalear*
when
$grd1: \quad ped < 1000$
then
$acc1: \quad ped := ped + 2$
$acc2: \quad dist := (ped + 1) \times diam \times 3 \times (plato \div cambio)$
bend

Lo que está mal en el evento anterior es la asignación $ped := ped + 2$, en lugar de la correcta $ped := ped + 1$. Al lanzar este evento en *ProB*, obtenemos lo que se muestra en la figura 10.

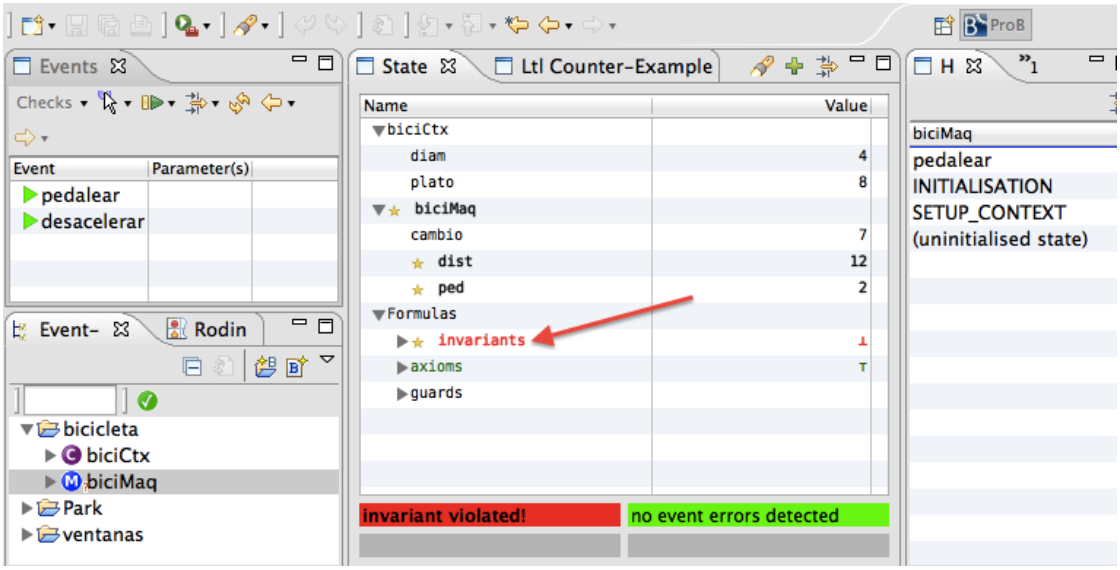

Figura 10: Invariante que no se cumple

Se observa que *ProB* reporta la violación de un invariante. Es decir, que alguna de las propiedades definidas en la sección *invariants* del modelo no se cumple en alguno de los estados. Se puede saber cuál es abriendo la pestaña a la izquierda de la palabra *invariante* señalada por la flecha roja, como se muestra en la figura 11.

El invariante que aparece en rojo es el que es falso. La razón, claro, es que al modificar los pedalazos agregando dos, en lugar de uno, la distancia recorrida ya no concuerda con el número de pedalazos.

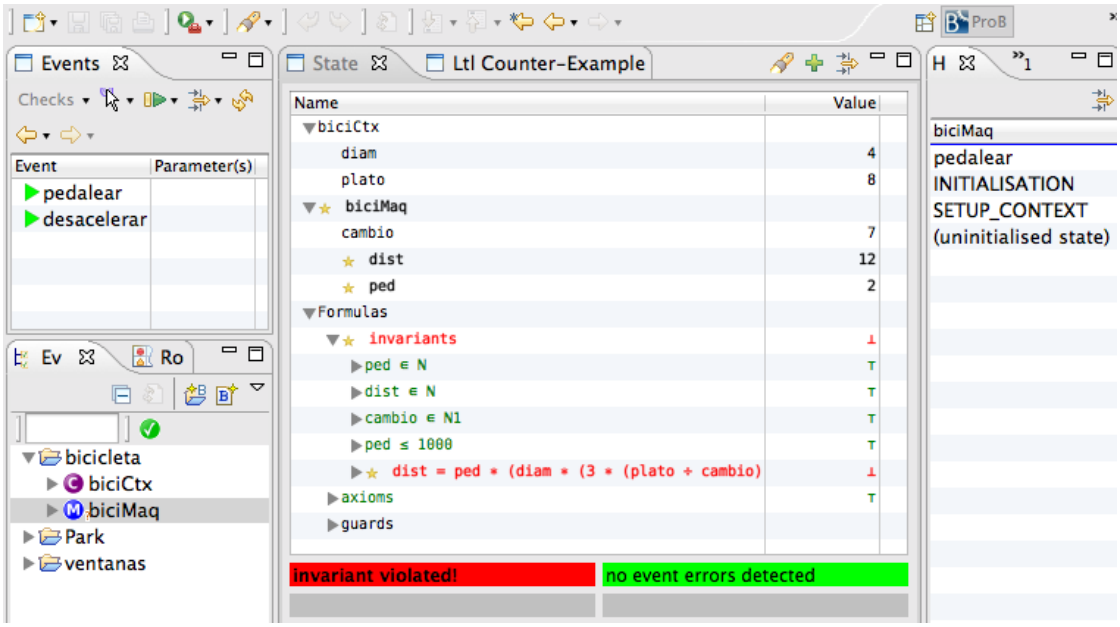

Figura 11: Desplegar el invariante que no se cumple

2. Modelos con observaciones numéricas

Supongamos el sistema de un bus urbano. Hay muchos aspectos que se pueden observar en él. Por ejemplo, podrían observarse las tarjetas de transporte de todas las personas que están dentro del bus, o también las estaciones por las que ya el bus ha pasado. Estas observaciones corresponden a cierta clase de objetos (tarjetas, estaciones). Cada estado sería un listado de estos objetos. Podemos elegir, en cambio, observar solamente cantidades. Por ejemplo, cuántas personas están dentro del bus, o cuántas estaciones faltan en el recorrido.

Quien construye el modelo de un sistema decide qué tipo de observaciones le parece conveniente representar. En modelos simples, las observaciones pueden ser solamente numéricas, como se mencionó para el bus. Si optamos por observar simplemente el número de pasajeros que hay adentro del bus en cada momento, ese número va a estar restringido por el cupo máximo de pasajeros que caben en el bus: no se pueden subir más pasajeros de los que caben.

El modelo comprende entonces un aspecto que es *constante*, es decir, algo que no cambia a medida que se desarrolla el funcionamiento del sistema. Este aspecto es el cupo del bus. El número de pasajeros dentro del bus puede que cambie si alguien se sube o alguien se baja, pero el cupo del bus sigue siendo el mismo. Todos los aspectos constantes de un sistema, como ya vimos, se definen en el *contexto*. El siguiente es el contexto del sistema

del bus:

context *ctx*
constants
cupo
axioms
$ax1: \quad cupo \in \mathbb{N}_1$

La notación $\mathbb{N}_1$ corresponde a los número naturales, sin incluir el cero. El axioma 1 dice entonces que la constante *cupo* es un valor cualquiera de ese conjunto. Por ejemplo, podría ser igual a 42 o a 15, o a algún otro número distinto de cero. No es necesario establecer un valor específico. Cuando se ejecuta el modelo con el programa *ProB*, este escoge automáticamente alguno.

Quien modela el sistema decide qué quiere observar. En nuestro caso queremos observar la cantidad de personas que están montadas en el bus. Para cada observación se define una variable. El valor de esta variable constituye una observación. La siguiente máquina define estas observaciones:

machine *Bus* **sees** *ctx*
variables
numpBus `// número de personas en el bus`
invariant
$inv1: \quad numpBus \in \mathbb{N}$

El tipo de la variable *numpBus* es el conjunto de números naturales, incluyendo el cero. El valor cero se incluye porque en un momento dado puede que no haya nadie subido en el bus.

La parte dinámica del sistema comprende las acciones que se observan y que cambian los valores de las variables. En este caso, se observa que se suben y se bajan pasajeros. Estas acciones constituyen los eventos del sistema:

events	**events**
event *subirse*	**event** *bajarse*
when	*when*
$grd1: \quad numpBus < cupo$	$grd1: \quad numpBus > 0$
then	*then*
$ac1: \quad numpBus := numpBus + 1$	$ac1: \quad numpBus := numpBus - 1$
end	*end*

La guarda (o condición) del evento *subirse* establece que el evento solamente puede lanzarse cuando el número de personas que está dentro del bus sea *estrictamente* inferior al cupo del bus. Por ejemplo, si el cupo es de 40 personas, el evento podría lanzarse cuando adentro del bus haya 20 personas, o cuando haya 39 personas, pero no cuando ya estén 40 personas subidas en el bus. En este último caso no se puede lanzar el evento porque, si ya

hay 40 personas adentro y el evento se lanza, van a quedar 41 personas subidas en el bus, lo que sobrepasa el cupo máximo. La acción 1 del evento *subirse* establece que la variable *numpBus toma ahora un nuevoi valor*, que es igual al que tenía antes más uno. Es decir, si antes de lanzar el evento el valor de la variable *numpBus* era igual a 25, por ejemplo, después de relizar la acción del evento el valor de *numpBus* será igual a 26.

En el evento *bajarse*, la condición stablece que el evento solamente puede lanzarse si el número de personas dentro del bus es estrictamente mayor a cero. Es decir, si hay al menos una persona dentro del bus. Esto es razonable porque, si no hay nadie adentro, difícilmente alguien podrá bajarse. La acción del evento decrementa en uno el valor que tenga en ese momento la variable *numpBus*.