

## 14 Reality Check ®

Se construyen modelos para solucionar problemas. Una vez construídos, existen varias pruebas que se pueden hacer para confrontarlos con la realidad. Las pruebas deben ser explícitas y tomar la forma de exámenes de comportamiento del modelo, o de partes del modelo, bajo diferentes suposiciones, o pueden ser simulaciones mentales implícitas y análisis basados en el entendimiento de los modelos y del proceso de modelado. En ambos casos estas pruebas son muy importantes para asegurarse de que el modelo que se desarrolló puede representar adecuadamente los problemas a los que es aplicado.

Los Reality Check dan una vía directa para plantear pautas que se piensa deben ser verdaderas acerca del modelo para ser útil, y las herramientas para verificar automáticamente la conformidad del modelo a esas pautas. Reality Check es una nueva tecnología que se agrega para validar y defender los modelos que se construyen. Puede focalizar las discusiones a partir de ciertas suposiciones hechas en los modelos hacia convicciones más sólidamente fundadas acerca de la naturaleza de la realidad.

Este capítulo:

- Introduce el concepto de un Reality Check.
- Muestra como crear Constraints (Restricciones) y Test Inputs (Entradas de prueba)
- Muestra como controlar la coherencia de un modelo con las ecuaciones de Reality Check.
- Muestra como construir un modelo usando Reality Check.

Copyright © 1998-2007 Ventana Systems, Inc.  
Traducido al español con autorización  
Copyright de la traducción (c) 2007 Juan Martin Garcia

Causal Tracing, Reality Check, Vensim and Ventana  
son marcas registradas de Ventana Systems, Inc.

## Los modelos y la realidad

Los modelos son representaciones de la realidad, o bien percepciones personales de la realidad. Para validar la utilidad de un modelo, es importante determinar si las cosas que se observan en la realidad también se mantienen verdaderas en el modelo. Esta validación se puede hacer usando métodos formales o informales para comparar las mediciones y los comportamientos del modelo. La comparación se puede hacer observando series temporales de datos, observando si las condiciones corresponden a descripciones cualitativas, controlando la sensibilidad de las suposiciones en el modelo, y desarrollando explicaciones correctas para los comportamientos generados por el modelo y los patrones de comportamiento.

Otro importante componente en la validación de un modelo es la consideración detallada de los supuestos acerca de la estructura. Los usuarios no requerirán información que no está disponible para tomar decisiones. Es necesario reforzar las relaciones causales. Los materiales se deben conservar.

Entre los detalles de la estructura y la abrumadora riqueza de comportamiento, hay muchas cosas que se pueden decir acerca de un modelo y que raramente lo influyen. Si se tuviera que completar la frase “Para que un modelo o submodelo sea correcto cuando \_\_\_\_\_ debería \_\_\_\_\_”, se vería que hay muchas cosas que se pueden hacer a un modelo para encontrar problemas y desarrollar confianza en él.

En la mayoría de los casos, algunas de las condiciones para que un modelo sea correcto son tan importantes que se las verifica. En otros casos, las condiciones no se expresan acerca del modelo completo sino acerca de un pequeño componente del modelo, o una ecuación. En tales casos, como constructor de modelos, se puede diseñar sobre las experiencias propias y el trabajo de otros relacionado al comportamiento de estructuras genéricas y formulaciones específicas.

Sin embargo, la mayoría de las condiciones que necesitan cumplirse para que un modelo sea correcto nunca se verifican. Usando las técnicas tradicionales de modelado, los procedimientos de verificación requieren separar partes del modelo, ejecutarlos con diferentes entradas, cambiar la estructura básica en ubicaciones preseleccionadas, hacer muchas de simulaciones, y revisar los resultados. Aún cuando se haga esto, se efectúa a menudo sobre una versión del modelo que posteriormente es revisada, sin verificar el efecto de las revisiones.

Las ecuaciones de Reality Check proporcionan un lenguaje para indicar lo que es necesario que se cumpla para que un modelo sea correcto, y las herramientas para verificar automáticamente la conformidad con tales requerimientos. Las especificaciones que se hacen a un modelo no están atadas al modelo. Están separadas de las ecuaciones normales del modelo y no interfieren con su funcionamiento normal. Presionando un botón es posible ver en que puntos un modelo viola o no las restricciones que impone la realidad.

## **El dominio de la experiencia**

Aunque las ecuaciones de Reality Check en Vensim se escriben como una extensión del lenguaje de modelado, las herramientas y la experiencia requeridas para escribir ecuaciones de Reality Check son diferentes que las requeridas para escribir el modelo. Las ecuaciones de Reality Check son afirmaciones acerca de la naturaleza del comportamiento en la realidad. No requieren la creación de una estructura capaz de generar un comportamiento particular. Las ecuaciones de Reality Check crean condiciones de comportamiento y luego verifican si la estructura del modelo ofrece una respuesta de comportamiento apropiada.

Debido a que las ecuaciones de Reality Check se formulan en un entorno de comportamientos, las personas más apropiadas para formularlas son aquellas que mayor conocimiento tienen del comportamiento, en general, expertos en el tema en estudio. Por este motivo, las ecuaciones de Reality Check permiten mucho más que encontrar errores sintácticos en el modelo. Las ecuaciones de Reality Check permiten al usuario de un modelo tener confianza en la calidad de los resultados que obtiene.

## Definir las ecuaciones de Reality Check

Las ecuaciones de Reality Check se escriben de la misma manera que las ecuaciones en Vensim. Se pueden usar las herramientas de esquema (Sketch Tool) para definir entradas a las ecuaciones de Reality Check, o simplemente escribir las ecuaciones directamente en el Editor de Ecuaciones o en el editor de Texto. Estructuralmente las ecuaciones de Reality Check no son entradas a ninguna ecuación del modelo, pero usan variables del modelo en sus definiciones. Cuando se efectúa un ejercicio de Reality Check, pueden cambiar el valor de las variables del modelo tanto como las ecuaciones usadas para computarlas. Sin embargo, se enfatiza que las ecuaciones de Reality Check no son proposiciones acerca de la estructura causal sino acerca del comportamiento-“Si pasa esto, entonces debe ocurrir esto”-.

Las convenciones acerca de nombres apropiados para ecuaciones Reality Check son diferentes de aquellas para variables de modelo. Las variables de modelo deberían tener nombres con significado más o menos obvio – *Mano de obra, productividad, capacidad, determinación, propensión al ahorro* y similares. Por otra parte, las ecuaciones de Reality Check deberían ser frases que describen la naturaleza de la verificación – *sin trabajadores no hay producción, lluvia implica inundación*. La mejor guía para definir nombres en Reality Check es pensar en ellas como verdaderas o falsas, y nombrar el Reality Check con la proposición que debería verificarse cuando resulta verdadera.

Hay dos tipos de ecuaciones que pueden ser definidas en Vensim para hacer uso de las funciones de Reality Check: **Constraints (Restricciones)** y **Test Inputs (Entradas de prueba)**. Las Restricciones hacen proposiciones acerca de las consecuencias que podrían resultar de un determinado conjunto de condiciones. Se llaman así porque especifican la manera en que las Entradas de Prueba podrían restringir el comportamiento. La violación de una Restricción indica un problema en el modelo. Las Entradas de Prueba son una manera de especificar condiciones o circunstancias bajo las cuales surgen las Restricciones. Dado que las Entradas de Prueba pueden ser usadas con las Restricciones, se describen primero.

## Test Inputs

Los Tests Inputs permiten definir condiciones alternativas cambiando las ecuaciones para una variable en el modelo. Su forma básica es:

$$\text{nombre :TEST INPUT: } variable = \text{expresión}$$

Donde *nombre* es el nombre de un Test Input. Lo que aparece a la izquierda de un :TEST INPUT: es exactamente el mismo formato de ecuación que normalmente se usa para definir una variable auxiliar y sólo puede contener variables del modelo. La ecuación que escribe está también restringida porque no se pueden usar funciones dinámicas (tal como SMOOTH), funciones de definición (como ACTIVE INITIAL) ni usar Macros. Si se necesitara este tipo de funcionalidad, se pueden crear variables extras en el modelo para usar en Test Inputs.

Los Test Inputs solo se pueden usar en la parte condicional de las ecuaciones de Restricción. La razón más importante para definir Test Inputs es dar un nombre a la experiencia que se está desarrollando. Esto se puede hacer más fácilmente leyendo la Restricción. Si no se definen Test Inputs, se pueden definir Constraints usando la parte *variable = expresión* de la ecuación de Test Input. Se aplican los mismos formatos de restricción de las ecuaciones.

### Test Inputs dinámicos

Adicionalmente a una expresión alternativa para un Test Input, es a menudo deseable forzar un cambio en una variable después de un período de tiempo de simulación. Por ejemplo, se puede desear que la producción caiga en forma de rampa a 0 en el lapso de 10 a 12, pero antes del tiempo 10 dejar la producción tal como era anteriormente. Este tipo de cambio es usual para escribir Reality Checks completos para el estudio de la respuesta del modelo a Test Inputs de interés.

Para crear Test Inputs con un cierto perfil de tiempo hay una serie de funciones que comienzan con **RC — RC COMPARE, RC DECAY, RC GROW, RC RAMP y RC STEP** (detalles en el Capítulo 4 del Reference Manual). Todas se comportan de una manera similar. Por ejemplo:

```
TIProduccion a cero :TEST INPUT:
```

```
produccion = RC RAMP(produccion,0,2,10)
```

Este Test Input causará que la *produccion* caiga de su valor a tiempo 10 hasta cero a tiempo 12.

Todas las funciones RC...toman dos argumentos opcionales: un tiempo de comienzo y una duración. Si se omite duración Test Input continúa en estado de cambio hasta el final de la simulación. Si se especifica una duración el Test Input continuará cambiando hasta el tiempo especificado, y luego la variable revierte hasta su valor normalmente computado.

Si el tiempo de comienzo (10 en el presente ejemplo) se omite, y el modelo contiene la constante RC START TIME, el cambio comienza a este tiempo específico Si RC START TIME no está en el modelo, el cambio empezará a INITIAL TIME + TIME STEP. Usando RC START TIME de esta forma es conveniente porque permite cambiar globalmente el tiempo al cual los cambios producen efecto y permite eliminar argumentos adicionales en las funciones RC. Tener Test Inputs que comienzan durante la simulación es útil porque previene la interferencia entre el comportamiento del modelo y su verificación, y permite ejecutar simulaciones de prueba con valores relativos diferentes para las variables.

## Constraints (Restricciones)

Las Constraints toman la forma:

nombre :*THE CONDITION*: condición :*IMPLIES*: consecuencia

:*THE CONDITION*: y :*IMPLIES*: son palabras clave especiales en Vensim. *Condicion* y *consecuencia* son expresiones lógicas que se describen abajo. El nombre de una Constraint debe usar letras y números tal como otras variables en Vensim. Las Constraints no necesitan unidades de medida, aunque si se está usando el Editor de Texto se debe poner el símbolo ~ como un separador. Se pueden adicionar comentarios a las Constraints tal como se hace con otras variables en Vensim.

### **Expresiones lógicas**

Las Constraints usan una condición y una consecuencia que son definidas como expresiones lógicas. Un ejemplo de esto podría ser:

*sin capital no hay producción* :*THE CONDITION*: *Capital = 0*  
:*IMPLIES*: *producción = 0*

Cuando se prueban las ecuaciones de Reality Check, Vensim fuerza una condición a ser cierta aunque el modelo genere valores que sugieren que podría o no ser cierta, y controla la veracidad de la consecuencia. Si la condición es cierta, pero la consecuencia no lo es, Vensim muestra el problema como un error del Reality Check. Vensim también hace verificaciones pasivas, como se describe a continuación. Las expresiones lógicas puede ser más complicadas que la anterior. Se construyen usando comparaciones =, >, <, and <> junto a :OR:, :AND: y :NOT:. Una expresión lógica podría ser:

*Población > 8E9* :AND: (*disponibilidad de alimentos < .75* :OR:  
*Polución > polución crítica*)

Aquí se comparan varias situaciones, y esta expresión será cierta si *Población > 8E9* y también (*disponibilidad de alimentos < .75* o *Polución > polución crítica*, o ambas.

Las expresiones lógicas pueden hacerse difíciles de entender y se recomienda no combinar demasiadas cosas en una condición. En el segmento consecuencias, es a menudo usual tener muchos puntos combinados con :AND: (para verificar varias consecuencias partiendo de una condición simple), pero raramente son útiles estructuras más complicadas.

El segmento correspondiente a la condición de una Constraint está restringido a la comparación de variables con variables y variables con números. La única excepción es

que se puede usar *variable = expresión* ., o un Test Input con nombre como componente de una de las condiciones lógicas. Esto es:

```
pop lt cc :THE CONDITION : Población < Capacidad de carga * 1.1  
:IMPLIES: muertes por sobrepoblación < 1000
```

es errónea porque toma la forma de *variable < expresión*, donde:

```
pop lt cc :THE CONDITION : Población = Capacidad de carga * 1.1  
:IMPLIES: muertes por sobrepoblación < 1000
```

usa *variable = expresión* y es una expresión legítima. Las expresiones incluidas directamente en las condiciones de Constraint en esta forma pueden usar TIME TRANSITION y deben ajustarse a las reglas para formar Test Inputs.

Los Test Inputs se deben usar como condición de una Constraint, como en:

```
pop lt cc :THE CONDITION : pop at cc plus 10  
:IMPLIES: muertes por sobrepoblación < 1000
```

Donde *pop at cc plus 10* es un Test Input. Observar que los Test Inputs son tratados como variables lógicas que toma un valor de verdad si están activos, o falso si no lo están. Todos los componentes de una Constraint están limitados a usar funciones simoles (MIN, MAX, SUM, etc). Para otras funciones, no hay restricciones en las expresiones lógicas del segmento consecuencias de la definición de una Constraint.

No es usual verificar igualdad en las consecuencias porque las pruebas de igualdad son muy propensas a fallar aún cuando no hay nada erróneo en el modelo. Esto es debido a que conceptos equivalentes por definición, pero computados de diferentes maneras, suelen ser levemente diferentes en valor numérico, lo cual será evidenciado durante un test de igualdad.

### **Pruebas Dinámicas en las Consecuencias**

Las ecuaciones Reality Check que tienen Test Inputs con funciones del tipo RC...normalmente usan una función RC...CHECK en el segmento consecuencia. Las funciones RC...CHECK trabajan de una manera análoga a las funciones RC...de los Test Inputs. Mientras que los Test Inputs cambian el valor de una variable, el segmento consecuencia de una Constraint hace una comparación del valor de la variable. Las funciones RC...CHECK toman un argumento más que la correspondiente función RC... Este argumento es el período de gracia y permite un retraso después que se produce el Test Input y antes de que las consecuencias sean verificadas. Por ejemplo:

```
TI Producción a cero :TEST INPUT: producción =
```

*RC STEP(production,0)*

*RC Sin producción no hay despachos :THE CONDITION: TI Producción a cero :IMPLIES: ventas <= RC RAMP CHECK(.5, ventas,.0001)*

El Test Input hace que producción caiga a cero a RC START TIME. Después de un período de gracia de 0.5 se verifica si las ventas son iguales o menores a 0.0001 por el valor que tenían a RC START TIME. El uso de 0.0001 en lugar de 0 previene violaciones que podrían ocurrir con formulaciones continuas y es una buena práctica.

El período de gracia es el primer argumento de todas las RC....CHECK excepto para RC COMPARE CHECK, el cual toma primero el nombre literal de un archivo.

### **:CROSSING:**

En el segmento de las consecuencias de un Reality Check se puede usar :CROSSING: y :AT LEAST ONCE: con > y < para buscar relaciones del tipo “por arriba/debajo, como en:

... :IMPLIES:

*Inventario > :CROSSING: RC STEP CHECK(0,Inventario,1)*

Esto verificará que a RC START TIME, *Inventario* es primero mayor que su valor de base y luego se hace menor y permanece menor. Si hubiera dos :CROSSING: en una línea como en:

*Inventario > :CROSSING: :CROSSING: RC STEP CHECK(0,Inventario,1)*

Inventario necesita comenzar mayor, luego se hace menor, luego vuelve a hacerse mayor y permanece así.

Si se usa más de un :CROSSING:, se puede finalizar la secuencia con :IGNORE: para indicar que una vez realizado el número requerido de cruzamientos, no preocupa si se producen nuevos cruces. Por ejemplo:

*Inventario > :CROSSING: :IGNORE: RC STEP CHECK(0,Inventario,1)*

Esto dice que Inventario necesita comenzar mayor, luego hacerse menor, luego no importa lo que ocurra.

### **:AT LEAST ONCE:**

Análogo a :CROSSING: , la palabra clave :AT LEAST ONCE: simplemente requiere que una condición sea verdad una vez después de RC START TIME. Por ejemplo:

*Inventario* > :AT LEAST ONCE: RC STEP CHECK(0,*Inventario*,1)

dice que *Inventario* debe exceder su valor a RC START TIME por lo menos una vez durante el resto de la simulación. Debería estar siempre por encima, o cruzar de abajo a arriba. Si el valor esta por encima al menos una vez, este puede cruzar nuevamente abajo y la condición permanecerá verdadera.

### **Condiciones vacías (Empty Conditions)**

La parte condicional de una Constraint puede estar vacía, como en:

*débito limitado* :THE CONDITION: :IMPLIES: *débito* < 4E6

Esta ecuación establece que, sin importar lo que ocurra nunca habrá un débito mayor a cuatro millones. Observar que Vensim no trata de verificar todas las condiciones posibles del modelo si encuentra una condición vacía. Las Constraints con condiciones vacías se verifican pasivamente para cualquier período en el que se esté usando la función Reality Chef, y detectará un débito alto. Para el ejemplo simple que se muestra aquí, usando 4 E6 como el valor máximo que el débito debería tomar en su ecuación también resultará en un mensaje cuando el débito excede 4 E6 en la medida en que las advertencias no estén suprimidas. La Condición Vacía puede ser usada para evaluar expresiones mucho más generales.

### **Wildcard Tests (comodines) en las Consecuencias**

Además de verificar una variable, se puede verificar todas las variables para ver si satisfacen una condición. Para hacer esto usar un \* en lugar del nombre de una variable. Por ejemplo se puede escribir la restricción:

*Todo tranquilo* :THE CONDITION: FINAL TIME=101 :IMPLIES:

\* < 1E9 :AND: \* >= -1E3

Lo cual verificará que todas las variables están en el rango -1000 a 1000 000 000. La condición FINAL TIME = 101, que simplemente ejecuta la simulación un año extra, se usa en lugar de la Condición Vacía porque esta es una verificación que consume tiempo y las Constraints con condiciones vacías se verifican pasivamente cada vez que se ejecuta cualquier Reality Check.

## **Simulación y Reality Check**

Antes de comentar la mecánica de ejecutar ecuaciones de Reality Check, es usual describir muy brevemente que ocurre dentro del modelo. Las ecuaciones de Reality Check involucran una intervención sistemática en la estructura básica del modelo. Son cualitativamente diferentes de los análisis de sensibilidad en que estos no son vías de influencia bien definidas. Los Test Inputs y las Constraints pueden generar la necesidad de hacer cambios en casi cualquier punto de un modelo.

A efectos de cumplir los cambios indicados en la ejecución de las ecuaciones de Reality Check, Vensim reestructura el modelo, agregando ecuaciones y modificando la secuencia en que las ecuaciones se computan. Después de completar las ecuaciones de Reality Check, Vensim retorna el modelo a su estructura original. Esto significa que efectuar el seguimiento causal (causal tracing) en una simulación hecha con uno o más Test Inputs activos puede dar resultados sorprendentes y aparentemente incorrectos.

En algunos casos, la reestructuración y reordenamiento pueden dejar al modelo “deformado”. El más común de los problemas es que el modelo podría contener ecuaciones simultáneas y por lo tanto, no se pueda simular. Si este es el caso, Vensim mostrará el problema y no completará la simulación. Debido a la naturaleza vinculada al comportamiento de las ecuaciones Reality Check, la existencia de ecuaciones simultáneas no representa necesariamente un problema conceptual, pero impide la simulación. De ser posible, se deben reformular los Test Inputs que causan problemas para evitar ecuaciones simultáneas.

NOTA: las ecuaciones de Reality Check se ejecutan siempre usando simulaciones interpretadas, no compiladas. Esto es porque requieren continuas reestructuraciones de las ecuaciones.

## Verificaciones Active Constraint

Durante las verificaciones activas con Constraints, Vensim fuerza la parte condicional de una Constraint a ser verdadera, cambiando los valores de las variables o la estructura del modelo cuando sea necesario.

- Si la condición es una igualdad o un Test Input, Vensim agrega la ecuación a la ecuación existente en el modelo (recordar que la ecuación debe referenciarse al valor originalmente computado para la variable). El nuevo valor de la variable se utiliza entonces cada vez que esta se usa.
- Si la condición usa una desigualdad, Vensim primero verifica para ver si la desigualdad es verdadera.
- Si es verdadera, el valor de la variable no cambia.
- Si no es verdadera, Vensim la hace verdadera asignando el valor tal como si fuera una condición de igualdad.

Dado que forzó las condiciones para que fueran verdad, Vensim verifica si las consecuencias también son verdaderas.

## **Verificaciones Passive Constraint**

Si efectúa la verificación pasiva de una Constraint, Vensim simplemente evalúa ambas mitades de la ecuación Constraint como si fueran una expresión lógica. Si la consecuencia es falsa mientras la condición es verdadera, muestra un error. Siempre que se ejecute un Reality Check, Vensim verifica la conformidad pasiva de todas las Constraints que no son explícitamente activas . Esta verificación no se hace durante las simulaciones normales.

## **Informe de errores**

La violación de Constraints se muestran de la misma manera que cuando se excede una tabla Lookup. La primera vez que se viola una Constraint se muestra un error. Luego se envía un mensaje cuando la Constraint no es más violada, indicando un retorno a la conformidad. Se muestra la siguiente violación, y así siguiendo.

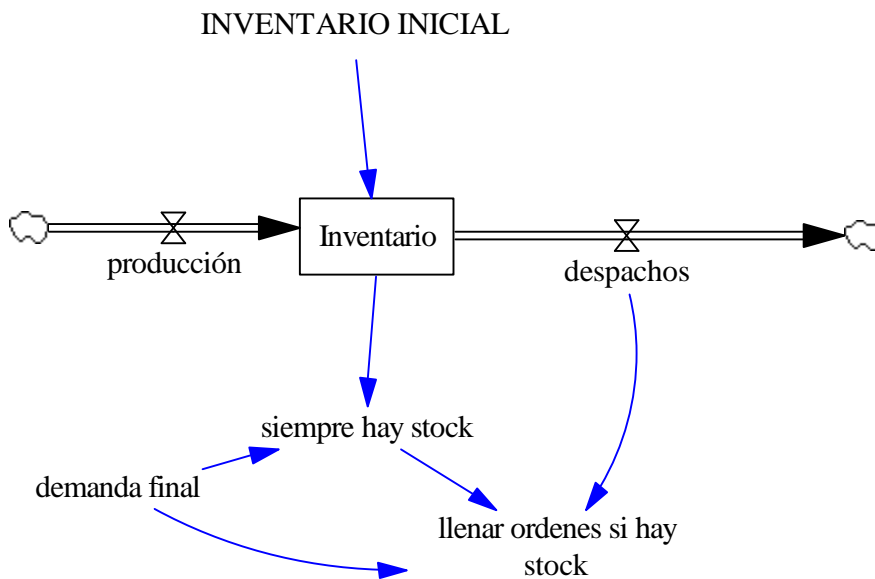
## Escribiendo las ecuaciones de Reality Check

Las ecuaciones de Reality Check se escriben de la misma manera que una ecuación en un modelo normal. En el Editor de Texto simplemente se escriben. En el Editor de Esquema, se pueden ingresar en diagramas mostrando todos los elementos a ser verificados como causas de los Test Inputs y Constraints. Los Test Inputs y Constraints no son parte de la estructura causal del modelo. Además, el lado derecho de las ecuaciones en Test Inputs no se muestran como causas de la variable del Test Input. Así la ecuación:

```
Inventario = INTEG(producción-despachos , INVENTARIO_INICIAL) ~~|
```

```
siempre hay stock :TEST INPUT: Inventario = 3*demanda final ~~|  
llenar ordenes si hay stock :THE CONDITION: siempre hay stock  
:IMPLIES: despachos >= demanda final ~~|
```

aparecerá en el diagrama como:



Es común que se desee mantener las ecuaciones y la estructura causal usadas en la definición de Reality Check separadas de las ecuaciones normales del modelo. Una vía conveniente para hacerlo es ponerlas en grupos separadas e incluirlas en vistas separadas.

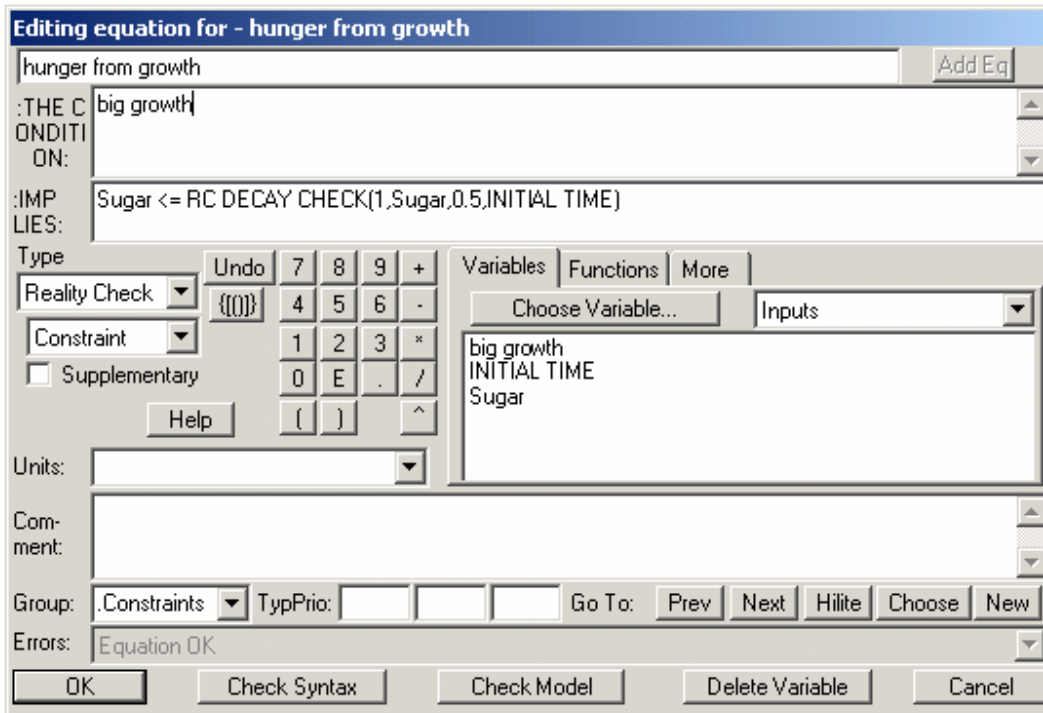
En el diagrama la mezcla de diferentes tipos de nombres puede ser confusa. Cuando se buscan flechas entrando a *llenar ordenes si hay stock*, no es qué causas determinan *llenar ordenes si hay stock*, sino qué necesitamos saber para determinar si es verdad.

Dada la dificultad para entender los diagramas que sólo muestran las relaciones causales y los flujos, se desea omitir esta complejidad adicional al diagrama de trabajo. Es probablemente más fácil colocar las ecuaciones de Reality Check en vistas separadas así no se las confunde con la estructura del modelo.

También puede ser muy conveniente adoptar una convención para nombrar las proposiciones de Reality Check. Por ejemplo se puede empezar todos los Test Inputs con TI y todas las Constraints con RC.

## Editor de ecuaciones


Se escriben las ecuaciones de Reality Check igual que las otras ecuaciones. Así, se crea una variable, se abre el Editor de Ecuaciones, se selecciona el tipo Reality Check y el Subtipo Constraint o Test Input y se escribe la ecuación.



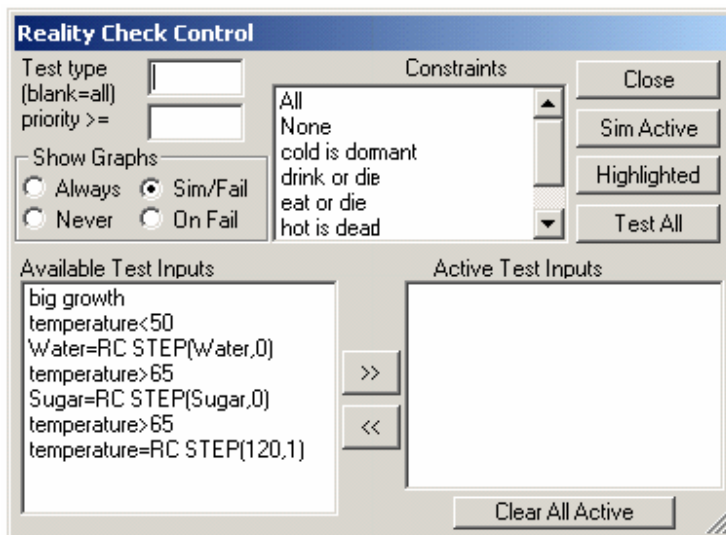
Cuando se selecciona el tipo Reality Check, se verá que la etiqueta Range cambia a TypPrio (este aparece cerca del fondo, justo a la derecha de Group). Este es un mecanismo para filtrar las proposiciones Reality Check en el diálogo Reality Check Control y será discutido a continuación. Se puede entrar un número tanto para Type como para Priority. Los valores que puede tomar Priority son arbitrarios, pero 1-10 son prioridades comunes. Type debería tomar un valor entero entre 0 y 64. Estos parámetros no están disponibles en PLE o PLE Plus.

## Ejecutar Reality Check

La aplicación Reality Check se ejecuta estableciendo primero una simulación y luego verificando las ecuaciones de Reality Check que fueron ingresadas.

Se puede comenzar un Reality Check desde la barra de Tareas o desde el diálogo Simulation Control (ver Capítulo 8 del Referente Manual para mayores detalles). Se pueden hacer cambios y ajustar otras opciones del mismo modo que se hace en una simulación normal. Después de haberlo hecho pulsar en **Reality Check**  en el diálogo Simulation Control o en el botón **Reality Check** de la barra de Tareas.

NOTA: Cualquier cambio que se haya efectuado a los valores constants o datos a ser usados previo a comenzar un Reality Check se mantendrá a través de toda la sesión. Cuando se comienza un Reality Check, aparece el diálogo de Control:



**Test type** (no en PLE o PLE Plus) permite especificar que tipo de Reality Check se desea ejecutar. Si está en blanco se verifican todos los tipos. El tipo de Reality Check se indica en el campo **TypPrio** del Editor de Ecuaciones. En Text Editor Type, **Priority** se encierra entre corchetes [ ] en el campo (~[type,priority]). Este campo sólo es aplicable cuando se usa el botón **Test All** para comenzar.

**Priority >=** (no en PLE o PLE Plus) restringirá la verificación a aquellas Constantes cuya prioridad es mayor o igual a la especificada. Este campo sólo es aplicable cuando se usa el botón **Test All** para comenzar.

NOTA: Si las **Constraints** no tienen una prioridad especificada, son tratadas con la máxima prioridad. Si no tienen un tipo especificado, se equiparan a todos los tipos. Para mostrar un gráfico de una variable que está siendo verificada en el segmento Consecuencias contra el comportamiento que esa variable debe cumplir, se usa **Show**

**Graphs.** Esto se hace usando una selección especial en **Graph Tool** y puede ser de mucha ayuda para entender la manera en que ocurrió una falla.

- **Always**, si está tildada, genera un gráfico para cada Constraint verificada.
- **Never**, si está tildada, suprime la salida de un gráfico.
- **Sim/Fail**, si está tildada, produce la aparición de un gráfico siempre que falla una Reality Check , y también cuando se hace una simulación simple usando **Sim Active** o los botones **Highlighted** .
- **On Fail**, si está tildada, hace que el gráfico solo aparezca si Reality Check falla. Cuando se usa el botón Test All es lo mismo que **Sim/Fail**. Para **Sim Active** y botones **Highlighted** suprime el gráfico a menos que realmente se produzca una falla.

**Constraints** es una lista de las Constraints que fueron ingresadas. Pulsando en una de ellas resaltara los correspondientes Test Inputs. Se puede entonces activar uno o más de estos Test Inputs.

**Test Inputs** es una lista de los Test Inputs en los modelos. Esta lista incluye todo lo que se ha definido explícitamente como Test Input, y todas las comparaciones en las expresiones lógicas que constituyen los componentes condicionales de las ecuaciones Constraint . Las comparaciones se muestran directamente y no es necesario dar un nombre.

>> mueve los items resaltados en la lista de Test Inputs a la lista Active Test Inputs .

<< borra los items resaltados en la lista Active Test Inputs .

Pulsar en un elemento de la lista lo resalta. Control-pulsar cambia el estado agregando o borrando la selección de elementos resaltados. Pulsar dos veces mueve el elemento a la lista Test Inputs .

**Active Test Inputs** muestra una lista de los Test Inputs, explícitos e implícitos, que están activos. Se usan si se pulsa el botón Simulate . Pulsar en un elemento lo resalta, y desmarca cualquier otra cosa resaltada. Control-pulsar cambia el estado de un elemento resaltado. Pulsar dos veces borra un elemento de la lista.

**Clear All Active** borra todos los elementos de la lista Active Test Inputs.

**Sim Active** simula el modelo usando los Test Inputs activos en la lista. Todas las otras Constraints se verifican pasivamente. Si la simulación se completa bien, los resultados son almacenados como una simulación normal, y pueden ser revisados con las herramientas de trabajo. Se debe ser cuidadoso en observar que el seguimiento de causas no siempre da los resultados esperados dado que la estructura del modelo ha sido modificada durante la simulación.

**Highlighted** simula el modelo usando el elemento que esta resaltado en la lista de Constraint . Debido a la estructura lógica de la parte condicional de una Constraint, esta realmente requiere más que una simulación. La simulación será almacenada igual que la simulación normal.

**Test All** verifica todas las Constraints en el modelo a un tiempo formando la colección de Test Inputs necesaria para activar cada Constraint. Debido a la estructura lógica de la parte condicional de una ecuación Constraint, puede tomar más de una simulación verificar una Constraint. Esta verificación puede consumir tiempo; los errores se muestran en una ventana separada. Ejecutar **Test All** no almacena ninguna simulación, simplemente muestra los resultados de Reality Check.

**Close** cierra el diálogo Reality Check Control.

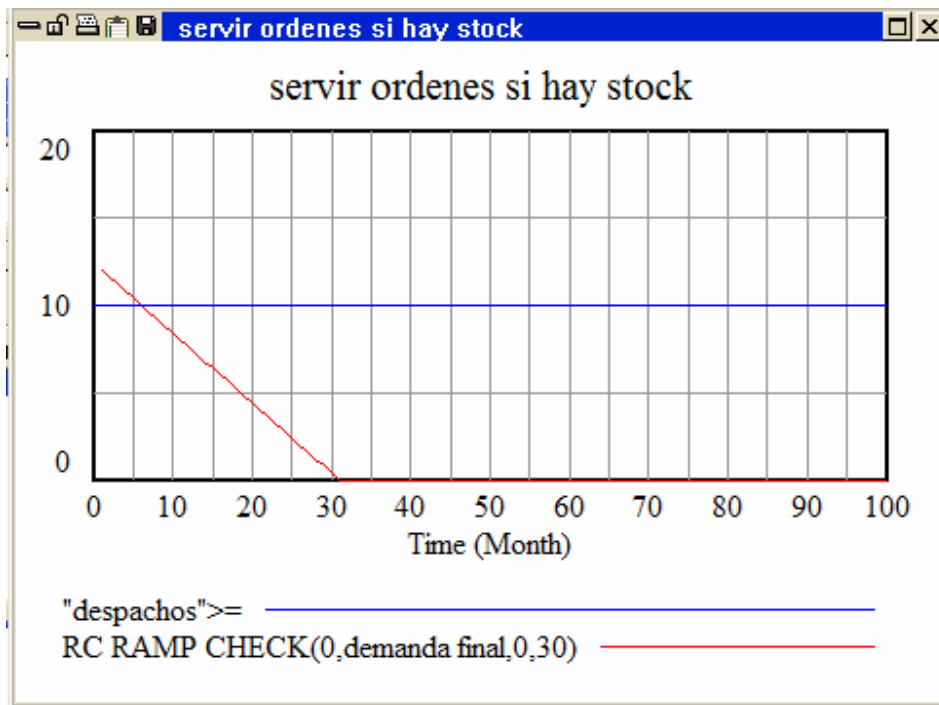
## La herramienta Reality Check

Se puede agregar la herramienta Reality Check al conjunto de herramientas de Análisis, (no aplica en PLE o PLE Plus). Esta herramienta es un atajo para comentar Reality Check, resaltando una Constraint y pulsando en **Highlighted**. La herramienta trabaja en la Constraint que está seleccionada dentro del esquema de Trabajo. Si la variable de trabajo no es una Constraint se muestra un mensaje de error.

## Resultados de un Reality Check

Los resultados de un Reality Check se muestran en una ventana de texto. La ventana muestra que Constraints fueron verificadas y si alguna Constraint fue violada. Se abre una nueva ventana cada vez que se pulsa en **Sim Active**, **Highlighted** o **Test All**. A continuación se dan ejemplos.

Aparecen uno o más gráficos. Estos gráficos tienen por objeto mostrar el segmento de consecuencias de una Constraint:



Aquí la línea azul muestra lo que hace *despachos*, mientras la línea roja muestra a que está siendo comparado.

## Repasar los resultados de una Simulation

Cada vez que se pulsa en **Sim Active** o **Highlighted** se efectúa una nueva simulación. Esta simulación recibe el nombre especificada en el recuadro Runname (en la barra de Herramientas o en el diálogo Simulation Control). Se puede, mientras está abierto el Control de Reality Check, mirar los resultados de una simulación tal como se haría con cualquier otra simulación. Se puede entonces hacer otra simulación y revisar estos resultados. Si se desea comparar dos simulaciones Reality Check, se puede cerrar el Reality Check control y luego comenzar de nuevo poniendo un nombre diferente en el recuadro **Runname**.

NOTA: Si se cambian las constantes en el diálogo Simulation Control o en la pantalla usando Set Up a Simulation de la barra de tareas, esos cambios se conservarán durante la sesión de Reality Check.

## Reality Check y el crecimiento de las levaduras

Para demostrar los mecanismos para escribir y usar las ecuaciones de Reality Check, es útil hacerlo a través de un ejemplo muy simple. Se desea modelar el crecimiento de levadura en un recipiente con agua. El agua se mantiene a temperatura constante que se puede fijar, y tiene una cantidad fija de azúcar al comienzo.

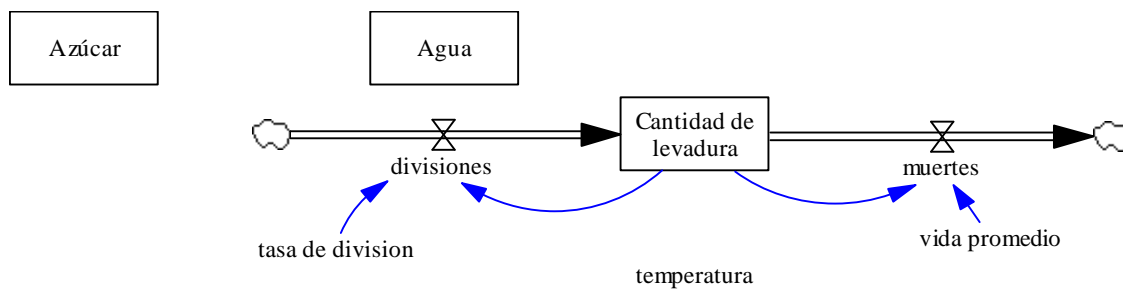
Primero se listará algunas de las ecuaciones de Reality Check que se deben cumplir:

- Si la temperatura baja de 50 grados, el crecimiento de levadura debería detenerse, y la levadura queda latente.
- Si la temperatura supera los 100 grados la levadura muere.
- Si no hay azúcar y las levaduras no están latentes, mueren.
- Si no hay agua y las levaduras no están latentes, mueren.
- Si la levadura continúa creciendo, consumirá todo el azúcar.
- Si la levadura continua creciendo, consumirá todo el agua.

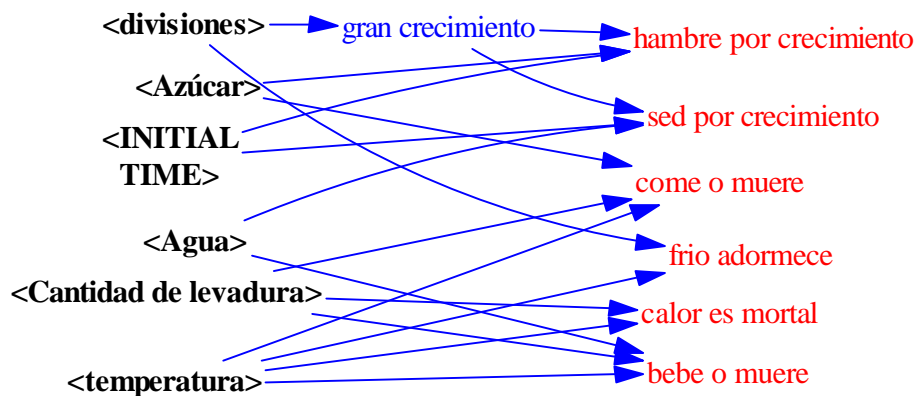
La otra cosa que sabemos es que las levaduras se reproducen por división y cuando las condiciones son adecuadas se pueden reproducir con un tiempo de duplicación de aproximadamente 10 minutos.

## Ecuaciones de Test Input y Constraint

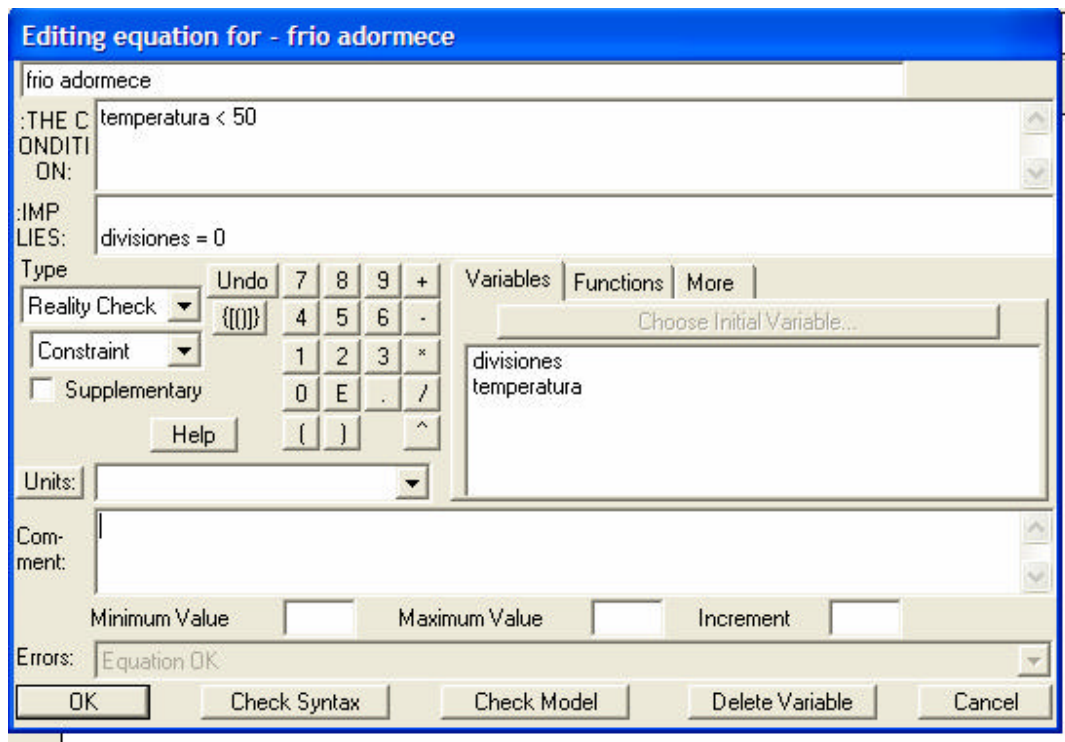
Comenzaremos el proceso de modelado definiendo las variables que necesitamos para efectuar los Reality Check y luego entrando las ecuaciones correspondientes. Las ecuaciones de Reality Check que fueron comentadas hablan acerca de cantidad de levadura, cantidad de agua, cantidad de azúcar, el número de divisiones de la levadura y la temperatura. Identificando los niveles, se ubican en el esquema del modelo de la siguiente manera.



Por claridad, se desea construir las ecuaciones de Reality Check en una segunda vista. Si se está usando Vensim PLE necesitará ubicarlas en la misma vista que la estructura del modelo. La primera etapa es ubicar todos los elementos del modelo como variables **shadow**. El uso de este tipo de variables permite que la estructura del modelo cambie sin requerir modificaciones del diagrama de Reality Check.



No hay reglas fijas para estructura los diagramas de Reality Check. La experiencia muestra que hacer una columna para las variables normales, Test Inputs y Constraints es la manera más fácil. Puede ser útil un código de colores (por ejemplo azul para Test Inputs y rojo por Constraints). Las flechas pueden quedar un poco desordenadas de esta forma, pero si se organiza la información para Reality Check por las variables que son afectadas, no es normalmente un gran problema. Se pueden ingresar las Constraints y Test Inputs en el Editor de Ecuaciones:



Se necesitará seleccionar **Type Reality Check** y **subtype Constraint**. Las condiciones y consecuencias se dividen en dos ventanas separadas.  
Las ecuaciones de Reality Check son:

```
frio adormece :THE CONDITION: temperatura < 50 :IMPLIES:
```

```
divisiones = 0
```

```
calor es mortal :THE CONDITION: temperatura = RC STEP(120,1)
```

```
:IMPLIES: Cantidad de levadura <= RC DECAY CHECK(1,Cantidad de levadura,2)
```

Para calor es mortal hemos usado un escalón en la temperatura y luego se uso una función RC DECAY CHECK para verificar la Cantidad de levadura. Para este modelo es adecuada una caída para mirar el comportamiento de Cantidad de levadura puesto que estamos focalizando en la situación de muerte.

```
come o muere :THE CONDITION: azúcar = RC STEP(azucar,0)
```

```
:AND: temperatura > 65 :IMPLIES: Cantidad de levadura <= RC DECAY
```

CHECK(1,Cantidad de levadura,15)

Aqui la Constraint require que la Cantidad de levadura decline hacia cero con un tiempo promedio de muerte de 15.

Bebe o muere :THE CONDITION: Agua = RC STEP(Agua,0)

:AND: temperatura > 65 :IMPLIES: Cantidad de levadura <= RC DECAY

CHECK(1,Cantidad de levadura,1)

gran crecimiento :TEST INPUT: divisiones = 1e+022

hambre por crecimiento :THE CONDITION: gran crecimiento :IMPLIES  
:azúcar <= RC DECAY CHECK (1,azúcar,0.5,INITIAL TIME)

sed por crecimiento :THE CONDITION: gran crecimiento :IMPLIES:  
Agua <= RC DECAY CHECK(1,Agua,0.5,INITIAL TIME)

Notar que para las dos últimas Constraints la función RC DECAY CHECK comienza verificando a INITIAL TIME ya que el Test Input comienza desde el inicio de la simulación. Si el Test Input ha usado una función RC STEP para comenzar durante la simulación, el argumento INITIAL TIME puede obviarse en los dos últimos RC DECAY CHECK .

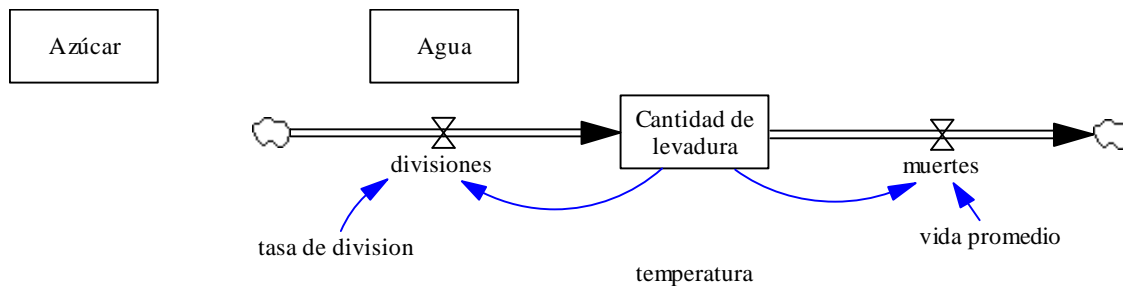
Finalmente se tiene:

RC START TIME = 10

El tiempo es arbitrario. Podría ocurrir que comenzando temprano hubiera unas pocas levaduras y cantidades de azúcar, mientras que empezando más tarde habría más levaduras y menos azúcar. Cuando un modelo pasa el Reality Check es una buena idea cambiar RC START TIME y re verificar.

## Un modelo inicial

Hay una breve lista de cosas que serán necesarias en el modelo para hacer posible la utilización de las ecuaciones de Reality Check. Contando con una comprensión muy básica del crecimiento exponencial, se puede llenar el esquema ya preparado para construir un modelo. Se comienza con el más simple de los modelos (*levadura1\_guia.mdl*)



Agua = 100  
 Units: ml  
 La cantidad de agua en el recipiente

Azúcar = 100  
 Units: g  
 La cantidad de azúcar en el recipiente.

bebe o muere: THE CONDITION: Agua = RC STEP(Agua, 0) :AND:  
 temperatura > 65: IMPLIES: Cantidad de levadura <= RC DECAY  
 CHECK(1, Cantidad de levadura, 1)  
 Units: \*\*undefined\*\*

calor es mortal: THE CONDITION: temperatura = RC  
 STEP(120, 1): IMPLIES: Cantidad de levadura <= RC DECAY CHECK  
 (1, Cantidad de levadura, 2)  
 Units: \*\*undefined\*\*

Cantidad de levadura = INTEG ( divisiones - muertes, 100 )  
 Units: Celulas  
 El número de levaduras.

come o muere: THE CONDITION: Azúcar = RC STEP(Azúcar, 0) :AND:  
 temperatura > 65: IMPLIES: Cantidad de levadura  
 <= RC DECAY CHECK(1, Cantidad de levadura, 15)  
 Units: \*\*undefined\*\*

divisiones = Cantidad de levadura \* tasa de division  
 Units: Celulas/Minuto

*FINAL TIME = 300*

*Units: Minute*

*frio adormece:THE CONDITION:temperatura < 50:IMPLIES:*

*divisiones = 0*

*Units: \*\*undefined\*\**

*gran crecimiento:TEST INPUT:divisiones = 1e+02 2*

*Units: \*\*undefined\*\**

*hambre por crecimiento:THE CONDITION:*

*gran crecimiento:IMPLIES:Azúcar <= RCDECAY CHECK  
(1,Azúcar,0.5,INITIAL TIME)*

*Units: \*\*undefined\*\**

*INITIAL TIME = 0*

*Units: Minute*

*The initial time for the simulation.*

*Muertes = Cantidad de levadura/vida promedio*

*Units: Celulas/Minuto*

*La velocidad de muerte de levaduras.*

*RC START TIME = 10*

*Units: Minute*

*SAVEPER = 1*

*Units: Minute*

*La frecuencia a la cual se almacenan los datos de salida.*

*sed por crecimiento:THE CONDITION: gran  
crecimiento:IMPLIES:Agua <= RC DECAY CHECK(1,Agua,0.5,INITIAL  
TIME)*

*Units: \*\*undefined\*\**

*tasa de division = 0.08*

*Units: 1/Minuto*

*La tasa de division de las levaduras.*

*Temperature = 85*

*Units: grados fahrenheit*

*La temperatura del agua en el recipiente.*

*TIME STEP = 1*

*Units: Minute*

*El intervalo de simulación.*

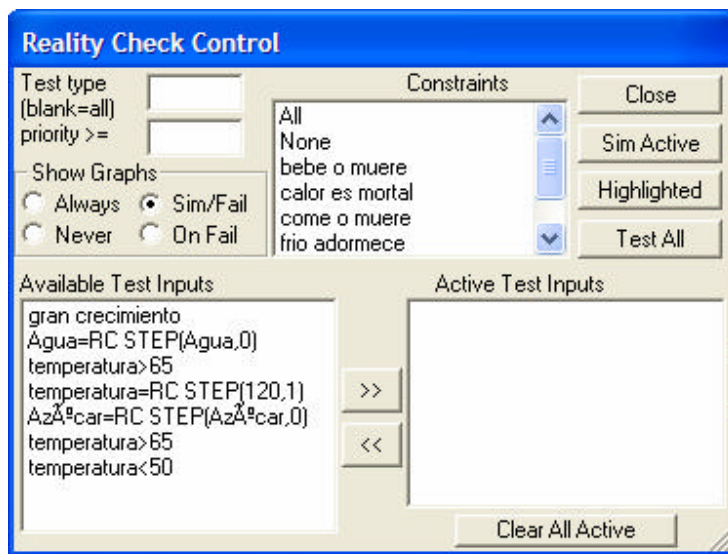
*vida promedio = 250*

*Units: Minuto*

*El tiempo promedio que una levadura sobrevive.*

En este modelo no estamos haciendo uso de *Azúcar*, *Agua* o *temperatura*, los cuales se han fijado a constantes utilizando el recuadro descolgable **Type** del editor de ecuaciones.

A continuación, se ejecuta el Reality Check de este modelo. Pulsar en el botón **Reality Check** de la barra de herramientas, o abrir el diálogo Simulation Control y pulsar en el botón **Reality Check**.



En el diálogo de control de Reality Check se ve una lista de Constraints para el modelo. Debajo de ésta está la lista de Test Inputs. Uno, *gran crecimiento*, fue explícitamente nombrado, mientras los otros son simplemente derivados de la parte condicional de las diferentes Constraints.

Pulsando en Test All comienza una serie de simulaciones. Se muestran las siguientes violaciones:

```

Constraint checking
Starting testing of Constraint- bebe o muere
Test inputs :
  Agua=RC STEP(Agua,0)
  temperatura>65
... testing - bebe o muere
The constraint -bebe o muere- violated at time 11
-----

Starting testing of Constraint- calor es mortal
Test inputs :
  temperatura=RC STEP(120,1)
... testing - calor es mortal
The constraint -calor es mortal- violated at time 11
-----

Starting testing of Constraint- come o muere
Test inputs :

```

El informe resumido es:

Starting testing of Constraint- frío adormece

Test inputs : temperatura<50

Starting to test the constraint –frío adormece

The constraint –frío adormece- violated at time 0

-----

The constraint –bebe o muere- violated at time 11

-----

The constraint –come o muere- violated at time 11

-----

The constraint –calor es mortal- violated at time 11

-----

The constraint –hambre por crecimiento- violated at time 1

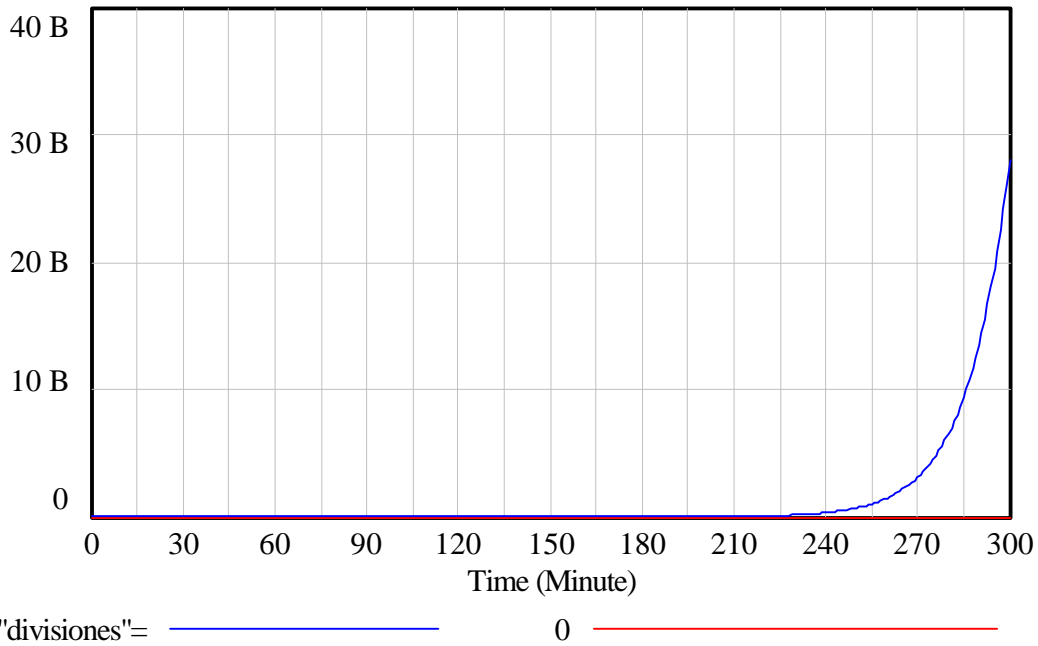
-----

The constraint –sed por crecimiento- violated at time 1

-----

Todas las Constraints han sido violadas. No es una gran sorpresa ya que los elementos involucrados en las Constraints no fueron conectados en el modelo. Este modelo sólo representa un mecanismo básico de crecimiento. No se ha dado atención al control o contención a las cuales están relacionadas las Constraints. Adicionalmente a la ventana de error habrá 6 gráficos tales como:

### frio adormece



La línea superior muestra lo que hacen las divisiones, mientras que la línea inferior muestra a que se debe.

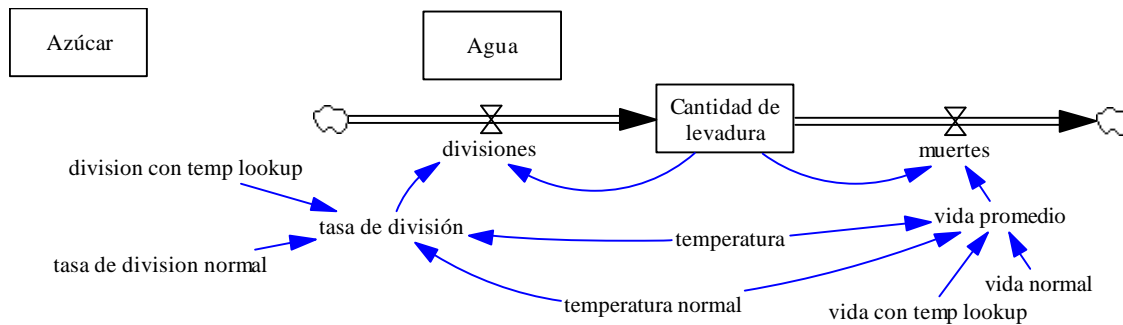
La ventana de reporte finalice con un resumen de lo que ha pasado.

```
*****
0 successes and 6 failures testing 6 Reality Check equations
The Reality Check Index as run is 0
Closeness score is 0.0% on 6 measurements
```

La primera línea es un resumen estadístico. La segunda línea muestra el Reality Check Index. Este se define como el número de éxitos dividido por el producto del número de variables dinámicas en el modelo con el total de variables en el modelo. Dado que por cada par de variables hay un potencial de uno o más ecuaciones Reality Check este índice es algo que debería estar siempre presente para un modelo con un conjunto completo de Reality Checks. Finalmente se muestra un grado de robustez (closeness score). Este grado es un promedio de la robustez del Reality Check. Si un Reality Check tiene éxito, este grado es 1. Si falla es 1 menos el promedio del error absoluto promedio dividido por la cantidad de variación en la variable que está siendo verificada. Por lo tanto un Reality Check que sólo falla poco tiene un grado cercano a 1, por lo que Closeness score es una medida continua cuanto, en promedio, son violadas las Constraints.

## Temperatura, divisiones y muertes

Dos condiciones relacionan la temperatura al crecimiento. Si la temperatura es baja todo pasa a estado de latencia. Si la temperatura es alta las levaduras mueren. Se reemplazan las ecuaciones para *divisiones* y *muerte* con este concepto en mente, y se crea *Levadura2\_guia.mdl*



```

divisiones = Cantidad de levadura * tasa de division
division rate = tasa de division normal * division con temp
lookup ( temperatura / norm temperatura )
tasa de division normal = 0.08
temperatura normal = 80
division con temp lookup ( (0,0),(0.8,0),(1,1),(2,1) )
muertes = Cantidad de levadura / vida promedio
vida promedio = vida normal * vida con lookup ( temperatura /
temperatura normal )
vida normal = 250
vida con temp lookup ((0,1),(1,1),(1.25,0.02),(2,0.001) )

```

Ahora verificar las Constraints mostrará es respetada. Las Constraints son violadas porque las levaduras no se mueren lo suficientemente rápido cuando la temperatura se incrementa. En este caso, se puede reestructurar el modelo para alcanzar la Constraint, o suavizar levemente la Constraint, dependiendo en que es más realista. Una manera de aliviar la restricción es cambiar el tiempo de transición para permitir más tiempo para la muerte de las levaduras (se cambia el argumento RC DECAY CHECK de 2 a 5):

```

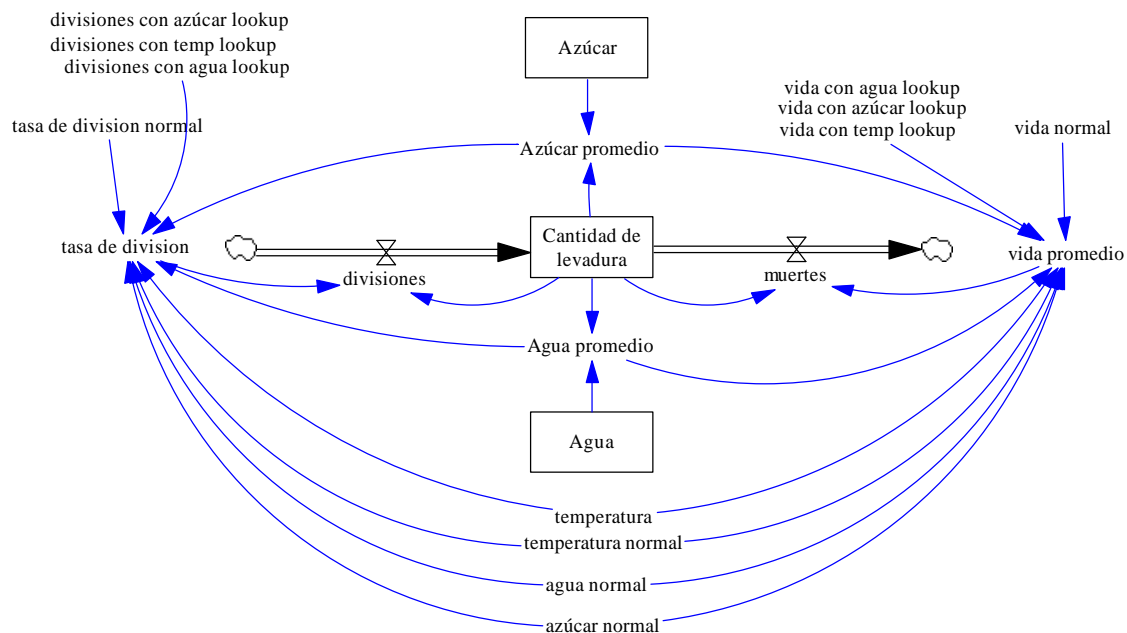
Calor es mortal :THE CONDITION: temperatura = RC STEP(120,1)
:IMPLIES: Cantidad de levadura <= RC DECAY CHECK(1,Cantidad de
levadura,5)

```

Ahora verificar las Constraints mostrará que frío adormece y calor es mortal son ambas respetadas. Este tipo de interacción, pactando la rigurosidad de las Constraints y haciendo ajustes a las ecuaciones es válido. Proporciona un medio de focalizar la atención sobre las Constraints que pueden ser violadas.

## Influencia del Agua y el Azúcar en las Divisiones

Las dos siguientes ecuaciones Constraint establecen que el agua y el azúcar son necesarios para sobrevivir. Se modeló el impacto de falta de agua y azúcar tanto en la tasa de división como la vida promedio de las levaduras. El nuevo modelo (*levadura3\_guia.mdl*) se ve:



Las ecuaciones están disponibles con el modelo. Observar que los Test Inputs elegidos causan condiciones extremas para la existencia en Agua y Azúcar, y la forma en que el modelo está formulado hace que las muertes crezcan explosivamente. Usando integración por Euler, el sistema muestra oscilaciones con la Cantidad de Levadura haciéndose negativa. Para evitar esto puede usarse otras técnicas de integración, o cambiar las ecuaciones para muertes.

$muertes = MIN(Cantidad\ de\ levadura / TIME\ STEP, Cantidad\ de\ levadura / vida\ promedio)$

Esta formulación evita dinámicas no esperadas, pero también pueden fijar la Cantidad de levadura en cero. Esto significa que Azúcar promedio y Agua promedio necesitan ser computadas como:

$Azúcar\ promedio = ZIDZ(Azúcar, Cantidad\ de\ levadura)$

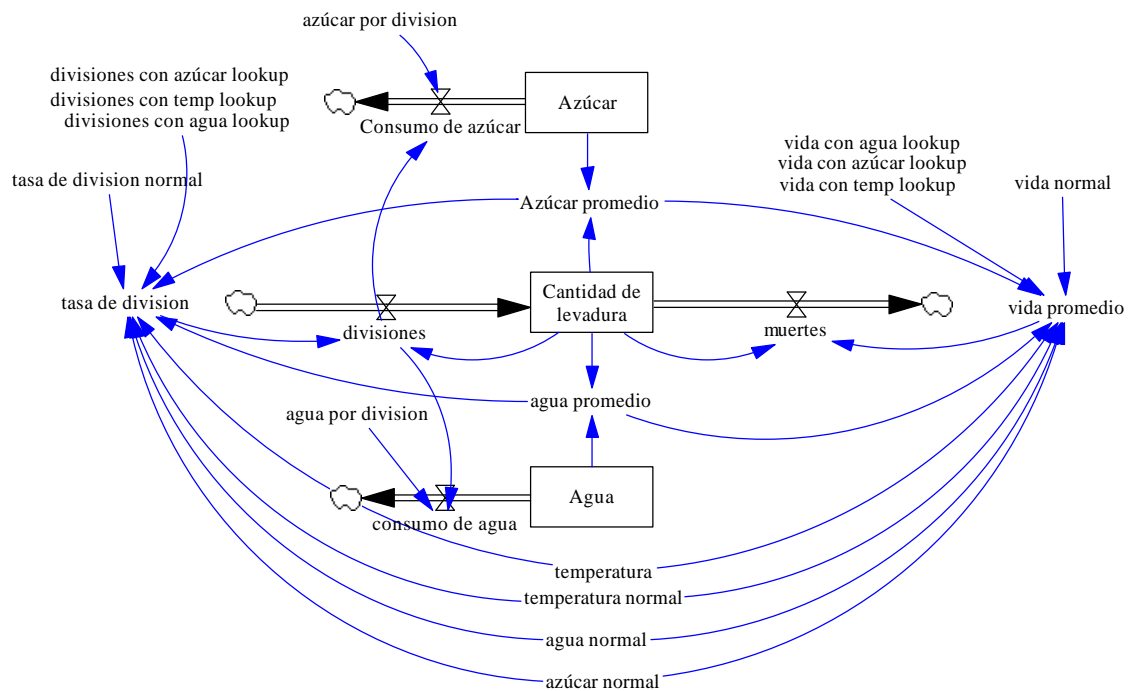
$Agua\ promedio = ZIDZ(Agua, Cantidad\ de\ levadura)$

Para prevenir errores numéricos cuando Cantidad de levadura es 0.

Con la estructura cambiada, el modelo supera con éxito cuatro de las seis Constraints.

## Agua y Azúcar influenciadas por las Divisiones

Aún no se logró ajustar el modelo a las dos últimas Constraints. El problema es que no hemos hecho la conexión entre lo que las levaduras están haciendo y cuanta agua y azúcar hay. La manera más simple de hacerlo es considerar el proceso de división como un consumidor de recursos de agua y azúcar. Así tenemos el modelo *levadura4\_guia.mdl*



Este modelo cumple todas las Constraints que se escribieron. Pero, ¿es un buen modelo? En este caso la respuesta probablemente es “no todavía”. Para una representación detallada y segura del crecimiento de las levaduras necesitaríamos algunos datos experimentales, y calibrar el modelo para efectos de temperatura y falta de recursos. El punto es que este modelo no viola las Reality Checks mas obvias que surgen del sentido común.

Se han mostrado varios modelos diferentes. ¿Qué pasó con las ecuaciones Reality Check? Excepto por los pequeños cambios efectuados a *calor es mortal*, no hay cambios y tienen exactamente el mismo diagrama.

## **Conclusión**

El uso de Reality Check sirve a dos propósitos. Primero, es un registro escrito de aspectos que asumimos deben ser verdaderos para que el modelo tenga sentido. Estos aspectos quedan normalmente sin documentar, aún cuando debieran ser el producto más importante del ejercicio de modelado. Cuestiones muy simples, como la noción de que si la levadura se mantiene en crecimiento podría quedarse sin agua, son muy importantes para entender el sistema.

Otro propósito importante acerca de este ejemplo es que ilustra una manera de construir modelos que es efectiva y eficiente. Se comienza desde un esquema básico y luego se agrega la estructura necesaria para cumplir las restricciones. Esto proporciona un rigor desde el principio del proceso de creación del modelo que puede incrementar ampliamente tanto la velocidad como la calidad del trabajo.

# Recursos de Vensim en español

## CURSOS ONLINE



### **Curso de Especialización en Dinámica de Sistemas**

<http://www.dinamica-de-sistemas.com/cursos/sis.htm>



### **Curso de Creación de Modelos en Ecología**

<http://www.dinamica-de-sistemas.com/cursos/ecologia.htm>

## LIBROS



### **Teoría y ejercicios prácticos de Dinámica de Sistemas**

<http://www.dinamica-de-sistemas.com/libros/sistemas.htm>



### **Sysware, la toma de decisiones en un mundo complejo**

<http://www.dinamica-de-sistemas.com/libros/sysware.htm>

Información: JMG@GRN.ES