Clase teórica 11

Verificación de la correctitud parcial

Repaso

- Definimos la **correctitud parcial** de un programa S con respecto a una especificación (p, q):

  Para todo estado σ: $(\sigma \models p \wedge val(\pi(S, \sigma)) = \sigma' \neq \perp) \rightarrow \sigma' \models q$.

  Es decir, a partir de todo estado σ que satisface la precondición p, **si el programa S termina (o no diverge)** lo hace en un estado σ´ que satisface la postcondición q.

  Se expresa con la expresión **semántica** $\models \{p\}\ S\ \{q\}$. Por ejemplo: $\models \{x = 0\}\ x := x + 1\ \{x = 1\}$

  {p} S {q} se conoce como **fórmula o terna de Hoare** de correctitud parcial.

- En esta clase describimos el método **axiomático** de verificación de correctitud parcial H para los programas con *while* introducidos en la clase pasada.

  H tiene axiomas y reglas que permiten probar **sintácticamente** la correctitud parcial de un programa S con respecto a una especificación (p, q).

  Se utiliza la expresión **sintáctica** $\vdash_H \{p\}\ S\ \{q\}$. Por ejemplo: $\vdash_H \{x = 0\}\ x := x + 1\ \{x = 1\}$

Método H

1. Axioma del skip (SKIP) $\{p\}\ skip\ \{p\}$

2. Axioma de la asignación (ASI) $\{p[x|e]\}\ x := e\ \{p\}$

   p[x|e] expresa la sustitución en p de todas las ocurrencias libres de la variable x por la expresión e.

3. Regla de la secuencia (SEC)

$$\frac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1\ ;\ S_2\ \{q\}}$$

4. Regla del condicional (COND)

$$\frac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi } \{q\}}$$

5. Regla de la repetición (REP)

$$\frac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \text{while } B \text{ do } S \text{ od } \{p \wedge \neg B\}}$$

6. Regla de consecuencia (CONS)

$$\frac{p \rightarrow p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1 \rightarrow q}{\{p\}\ S\ \{q\}}$$

- **Llama la atención la forma del axioma de la asignación (ASI).**

  El axioma establece {p[x|e]} x := e {p}.

  Se lee así: si luego de *x := e* vale p en términos de x, entonces antes de *x := e* valía p en términos de *e*.

  Por ejemplo, veamos cómo se completa la fórmula **{?}** x := x + 1 {x ≥ 0}:

  > **{x ≥ 0[x|x + 1]}** x := x + 1 {x ≥ 0}
  > **{x + 1 ≥ 0}** x := x + 1 {x ≥ 0}
  > Es decir, si luego de *x := x + 1* vale x ≥ 0, entonces antes de *x := x + 1* valía x + 1 ≥ 0.

  El axioma se lee de atrás para adelante, lo que impone una forma de probar de la postcondición a la precondición.

  Sería más natural la forma de axioma hacia adelante **{true} x := e {x = e}.**

  Pero esta fórmula de correctitud no siempre es verdadera. Por ejemplo:

  > **{true} x := x + 1 {x = x + 1}** es una fórmula falsa.
  > El contenido de la x de la derecha es previo a la asignación, y el de la x de la izquierda es posterior.

  Existe un axioma correcto hacia adelante, más complicado y no suele usarse: {p} x := e {∃z: (p[x|z] ∧ x = e[x|z])}.

- En la **regla de la secuencia (SEC),** el predicado (o aserción) r actúa como **nexo** y luego se descarta, no se propaga. Se permite usar la siguiente generalización:

$$\frac{\{p\}\ S_1\ \{r_1\},\ \{r_1\}\ S_2\ \{r_2\},\ \ldots,\ \{r_{n-1}\}\ S_n\ \{q\}}{\{p\}\ S_1\ ;\ S_2\ ;\ \ldots\ ;\ S_n\ \{q\}}$$

$$\frac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1\ ;\ S_2\ \{q\}}$$

- La **regla del condicional (COND)** formula un modo de verificar una selección condicional fijando un **único punto de entrada** y un **único punto de salida,** correspondientes a p y q, respectivamente.

$$\frac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \text{if B then } S_1 \text{ else } S_2 \text{ fi } \{q\}}$$

- La **regla de la repetición (REP)** se basa en un **invariante** p.

  Si p vale al comienzo del *while*,
  y mientras vale B el cuerpo S preserva p,
  entonces por un razonamiento **inductivo** p vale al finalizar el *while*.

  Claramente, REP **no asegura que el *while* termine.**

$$\frac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \text{while B do S od } \{p \wedge \neg B\}}$$

- La **regla de consecuencia (CONS)** permite **reforzar precondiciones** y **debilitar postcondiciones.** P.ej.:

de:	$\{x > 0\}\ S\ \{x = 0\}$	de:	$\{true\}\ S\ \{x = y + 1\}$	$\dfrac{p \rightarrow p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1 \rightarrow q}{\{p\}\ S\ \{q\}}$
y:	$x > 5 \rightarrow x > 0$	y:	$x = y + 1 \rightarrow x > y$	
se deduce:	$\{x > 5\}\ S\ \{x = 0\}$	se deduce:	$\{true\}\ S\ \{x > y\}$	

La regla no depende del lenguaje de programación sino del dominio semántico. Es una regla **semántica** más que sintáctica. Permite manipular **todos los axiomas del dominio semántico** en las pruebas (en nuestro caso los axiomas de los números enteros, porque acotamos el estudio a dicho dominio).

- El método H es **composicional**:

Dado un programa S con subprogramas S_1, …, S_n, que valga la fórmula $\{p\}\ S\ \{q\}$ depende sólo de que valgan ciertas fórmulas $\{p_1\}\ S_1\ \{q_1\}$, …, $\{p_n\}\ S_n\ \{q_n\}$, sin importar el contenido de los S_i (noción de **caja negra**).

P.ej., dado $S :: S_1 ; S_2$,
de $\{p\}\ S_1\ \{r\}$ y $\{r\}\ S_2\ \{q\}$
se deduce $\{p\}\ S_1 ; S_2\ \{q\}$,
independientemente de los contenidos de S_1 y S_2.

Si se tiene $\{r\}\ S_3\ \{q\}$ también se deduce $\{p\}\ S_1 ; S_3\ \{q\}$. S_2 y S_3 son intercambiables por ser funcionalmente equivalentes en relación a (r, q).

Ejemplo 1. Prueba de un programa que intercambia los valores de dos variables

- Dado S_{swap} :: z := x ; x := y ; y := z, se va a probar:

$$\{x = X \wedge y = Y\}\ S_{swap}\ \{y = X \wedge x = Y\}$$

Recordar que antes probamos esto **semánticamente**, usando directamente la semántica operacional del lenguaje de programación. Ahora vamos a probarlo **sintácticamente**, mediante axiomas y reglas.

Por la forma del programa S_{swap}, recurrimos al axioma ASI tres veces, una por cada asignación, y al final completamos la prueba utilizando la (generalización de la) regla SEC:

1. $\{z = X \wedge x = Y\}\ y := z\ \{y = X \wedge x = Y\}$ (ASI)
2. $\{z = X \wedge y = Y\}\ x := y\ \{z = X \wedge x = Y\}$ (ASI)
3. $\{x = X \wedge y = Y\}\ z := x\ \{z = X \wedge y = Y\}$ (ASI)
4. $\{x = X \wedge y = Y\}\ z := x\ ;\ x := y\ ;\ y := z\ \{y = X \wedge x = Y\}$ (1,2,3,SEC)

$$\{p[x|e]\}\ x := e\ \{p\}$$

$$\frac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1\ ;\ S_2\ \{q\}}$$

- Notar en el ejemplo cómo el axioma ASI establece una forma de prueba de la postcondición a la precondición.

- Por la **sensatez** de H, que probaremos en otra clase, se cumple:

$$\models \{x = X \wedge y = Y\}\ z := x\ ;\ x := y\ ;\ y := z\ \{y = X \wedge x = Y\}$$

- Obviamente, **semánticamente** también se cumple la misma fórmula pero permutando los operandos de la precondición, es decir:

$$\models \{y = Y \wedge x = X\}\ z := x\ ;\ x := y\ ;\ y := z\ \{y = X \wedge x = Y\}$$

- Para probar esta última fórmula **sintácticamente** tenemos que recurrir a la regla CONS, agregando a la prueba anterior, que terminaba en:

4. $\{x = X \wedge y = Y\}\ z := x\ ;\ x := y\ ;\ y := z\ \{y = X \wedge x = Y\}$ (1,2,3,SEC)

los siguientes dos pasos:

5. $(y = Y \wedge x = X) \rightarrow (x = X \wedge y = Y)$ (MAT)

6. $\{y = Y \wedge x = X\}\ z := x\ ;\ x := y\ ;\ y := z\ \{y = X \wedge x = Y\}$ (4,5,CONS)

- La aserción del paso 5 es un axioma de los números enteros, por eso se justifica el paso con el indicador MAT (por matemáticas).

- Así, hemos probado mediante el método H también la fórmula de correctitud:

$$\{y = Y \wedge x = X\}\ S_{swap}\ \{y = X \wedge x = Y\}$$

- Si quisiéramos **instanciarla**, por ejemplo en:

$$\{y = 2 \wedge x = 1\}\ S_{swap}\ \{y = 1 \wedge x = 2\}$$

debemos recurrir a una nueva regla, la **regla de instanciación (INST):**

$$\frac{f(X)}{f(c)}$$

tal que f es una fórmula de correctitud, X es una variable lógica, y c está en el dominio de X.

INST es una regla universal, se la usa en todos los métodos a pesar de que no se la suele mencionar.

Ejemplo 2. Prueba de un programa que calcula el valor absoluto

- El siguiente programa calcula en una variable *y* el valor absoluto de una variable x:

$$S_{va} :: \text{ if } x > 0 \text{ then } y := x \text{ else } y := -x \text{ fi}$$

Vamos a probar $\{true\}\ S_{va}\ \{y \geq 0\}$

Ejercicio: ¿La especificación mostrada es correcta?

Considerando las dos asignaciones del programa, se propone que los primeros pasos de la prueba sean:

1. $\{x \geq 0\}\ y := x\ \{y \geq 0\}$ (ASI)

2. $\{-x \geq 0\}\ y := -x\ \{y \geq 0\}$ (ASI)

Para poder aplicar la regla COND, se necesitan dos fórmulas $p \wedge B$ y $p \wedge \neg B$, que en este caso tendrían la forma $p \wedge x > 0$ y $p \wedge \neg(x > 0)$.

$$\frac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi } \{q\}}$$

Probamos con p = true, quedando entonces $true \wedge x > 0$, y $true \wedge \neg(x > 0)$, que de alguna manera deberemos relacionarlas con $x \geq 0$ y $-x \geq 0$ (continúa en el slide siguiente).

Entonces, partimos de:

1. $\{x \geq 0\}\ y := x\ \{y \geq 0\}$ (ASI)
2. $\{-x \geq 0\}\ y := -x\ \{y \geq 0\}$ (ASI)

hacemos:

3. $(\text{true} \wedge x > 0) \rightarrow x \geq 0$ (MAT)
4. $(\text{true} \wedge \neg(x > 0)) \rightarrow -x \geq 0$ (MAT)

y completamos la prueba de la siguiente manera:

5. $\{\text{true} \wedge x > 0\}\ y := x\ \{y \geq 0\}$ (1,3,CONS)
6. $\{\text{true} \wedge \neg(x > 0)\}\ y := -x\ \{y \geq 0\}$ (2,4,CONS)
7. $\{\text{true}\}$ if $x > 0$ then $y := x$ else $y := -x$ fi $\{y \geq 0\}$ (5,6,COND)

$$\frac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi } \{q\}}$$

- Respondiendo a la pregunta del slide anterior, efectivamente (true, $y \geq 0$) **no** es una especificación correcta de un programa que calcula el valor absoluto. P.ej., el programa:

$$S :: y := 1$$

satisface (true, $y \geq 0$) y no es el programa buscado. La variable *y* al final no tiene por qué tener el valor absoluto del contenido de la variable x al inicio. P.ej., se cumple: $\{x = 10\}\ y := 1\ \{y \neq |x|\}$. Una especificación correcta de un programa de valor absoluto sería:

$$(x = X, y = |X|)$$

Queda como ejercicio verificar el programa anterior S_{va} con respecto a esta especificación.

Axiomas y reglas adicionales del método H

- Por la **completitud** de H (se prueba en otra clase), agregarle axiomas y reglas al método es **redundante**. De todos modos, esta práctica es usual en los sistemas deductivos, facilita y acorta las pruebas. Algunos ejemplos clásicos de axiomas y reglas adicionales de H son:

- Axioma de invariancia (INV) $\{p\}\ S\ \{p\}$

  Cuando las variables de p y las variables que modifica S son disjuntas.

- Regla de la disyunción (OR)

$$\frac{\{p\}\ S\ \{q\},\ \{r\}\ S\ \{q\}}{\{p \vee r\}\ S\ \{q\}}$$

  Util para una **verificación por casos,** con distintas precondiciones e iguales postcondiciones.

- Regla de la conjunción (AND)

$$\frac{\{p_1\}\ S\ \{q_1\},\ \{p_2\}\ S\ \{q_2\}}{\{p_1 \wedge p_2\}\ S\ \{q_1 \wedge q_2\}}$$

  Util para una **verificación por casos,** con distintas precondiciones y postcondiciones.

- El axioma INV suele emplearse en combinación con la regla AND, para producir la **Regla de invariancia (RINV):**

$$\frac{\{p\}\ S\ \{q\}}{\{r \wedge p\}\ S\ \{r \wedge q\}}$$

tal que ninguna variable libre de r es modificable por S (se cumple {r} S {r}).

- Dada la Regla de la Disyunción (OR):

$$\frac{\{p\}\ S\ \{q\}\ ,\ \{r\}\ S\ \{q\}}{\{p \vee r\}\ S\ \{q\}}$$

una forma particular de la regla de uso habitual es:

$$\frac{\{p \wedge s\}\ S\ \{q\}\ ,\ \{p \wedge \neg s\}\ S\ \{q\}}{\{p\}\ S\ \{q\}}$$

útil cuando la prueba de {p} S {q} se facilita reforzando la precondición con **dos aserciones complementarias**.

Anexo de la clase teórica 11

Verificación de la correctitud parcial

Recordatorio de la prueba semántica del programa de swap

S_{swap} :: z := x ; x := y ; y := z.

Estado inicial σ_0, con $\sigma_0(x) = 1$ y $\sigma_0(y) = 2$.

$\pi(S_{swap}, \sigma_0) = (z := x ; x := y ; y := z, \sigma_0[x|1][y|2]) \rightarrow$

$(x := y ; y := z, \sigma_0[x|1][y|2][z|1]) \rightarrow$

$(y := z, \sigma_0[y|2][z|1][x|2]) \rightarrow$

$(E, \sigma_0[z|1][x|2][y|1])$

Entonces, $val(\pi(S_{swap}, \sigma_0)) = \sigma_1$, con $\sigma_1(x) = 2$ y $\sigma_1(y) = 1$.

Generalizando, si $\sigma_0(x) = X$ y $\sigma_0(y) = Y$, entonces $val(\pi(S_{swap}, \sigma_0)) = \sigma_1$, con $\sigma_1(x) = Y$ y $\sigma_1(y) = X$.

Proof outlines (esquemas de prueba)

Presentación alternativa de una prueba: se intercalan los pasos de la prueba entre las instrucciones del programa.

Se obtiene una prueba más estructurada, que documenta adecuadamente el programa.

En la verificación de los programas concurrentes las *proof outlines* son **imprescindibles**.

Por ejemplo, la siguiente es una *proof outline* de correctitud parcial de un programa que calcula el **factorial**:

```
{x > 0}
S_fac :: a := 1 ; y := 1 ;
         while a < x do
                  a := a + 1 ; y := y . a
         od
{y = x!}
```

```
{x > 0}
a := 1 ;
y := 1 ;
{inv: y = a! ∧ a ≤ x}
while a < x do
{y = a! ∧ a < x}
a := a + 1 ;
y := y . a
od
{(y = a! ∧ a ≤ x) ∧ ¬(a < x)}
{y = x!}
```

Es común presentar sólo las aserciones más importantes, mínimamente la **precondición**, la **postcondición** y los **invariantes**.

Acerca de los invariantes

- Toda aserción usada en una prueba es en realidad un **invariante**. Volviendo a la *proof outline* anterior: cualquiera sea el estado inicial, toda aserción siempre se cumple en el lugar donde se establece:

{x > 0}
a := 1 ; y := 1 ;
{y = a! $\wedge$ a $\leq$ x}
while a < x do
{y = a! $\wedge$ a < x}
a := a + 1 ; y := y . a
od
{y = x!}

- El uso de un invariante para la prueba de un *while* determina una prueba por **inducción**: es una aserción que vale al comienzo del *while* (base inductiva) y que toda iteración preserva (paso inductivo). De esta manera se prueba que el invariante se cumple **a lo largo de toda la computación** del *while*.

- La correctitud parcial pertenece a la familia de las propiedades ***safety***. Son propiedades que se prueban por **inducción**. Se asocian al enunciado: “Algo malo no puede suceder”. Otros ejemplos de propiedades *safety* son la *ausencia de deadlock* y la *exclusión mutua* o *ausencia de interferencia*, en los programas concurrentes.

Acerca de la completitud de los métodos de prueba

- La completitud se asocia a la problemática de la **expresividad** de las especificaciones.

  Dada la fórmula {p} S_1 ; S_2 {q}, ¿se puede encontrar una aserción intermedia r entre S_1 y S_2 que represente el conjunto de estados que existe entre ámbos?

  Dada la formula {r} while B do S od {q}, ¿se puede encontrar un invariante del *while*?

  La expresividad del lenguaje de especificación **depende** del lenguaje de programación y de la interpretación semántica de las variables. En nuestro caso, que trabajamos con el lenguaje de la lógica de predicados, los programas con *while* y los números enteros, **se cumple la expresividad**.

- Otra causal de incompletitud es la interpretación semántica de las variables.

  Trabajando con variables enteras, por el Teorema de Incompletitud de Gödel se sabe que hay enunciados de los enteros que no se pueden probar con ninguna axiomática.

  De esta manera, el método H debe entenderse como que incluye al conjunto de **todos los axiomas de los números enteros**. Su completitud no es absoluta, es **relativa** (de todos modos, de lo que se trata es de probar programas, no enunciados de los números enteros).

Clase práctica 11

Verificación de la correctitud parcial

Ejemplo 1. Prueba de correctitud parcial de un programa que efectúa la división entera entre dos números enteros

Probaremos: $\{x \geq 0 \wedge y > 0\}\ S_{div}\ \{x = y \,.\, c + r \wedge r < y \wedge r \geq 0\}$, con:

S_{div} :: c := 0 ; r := x ; while r ≥ y do r := r – y ; c := c + 1 od

$$\frac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \text{while } B \text{ do } S \text{ od } \{p \wedge \neg B\}}$$

Recordatorio de la regla REP

La dificultad de la verificación de un programa radica fundamentalmente en encontrar aserciones en distintas locaciones, en particular los **invariantes** de los *while*, para asociarlas con las premisas de alguna regla y obtener la conclusión deseada (de todos modos recordar que estudiamos pruebas a posteriori para simplificar la exposición, siendo en cambio la buena práctica **construir un programa en simultáneo con su verificación**).

Proponemos como invariante del *while* la aserción:

$$p = (x = y \,.\, c + r \wedge r \geq 0)$$

obtenida por una **generalización** de la postcondición del *while*. Notar que cuando el programa termina se cumple r < y, y así de la conjunción de esta condición y el invariante se alcanza la postcondición buscada.

Podemos estructurar la prueba de la siguiente manera, que se desarrolla en el slide siguiente:

a) $\{x \geq 0 \wedge y > 0\}$ c := 0 ; r := x $\{p\}$
b) $\{p\}$ while r ≥ y do r := r – y ; c := c + 1 od $\{p \wedge \neg(r \geq y)\}$
c) Finalmente, aplicando SEC a (a) y (b), y como $(p \wedge \neg(r \geq y)) \rightarrow x = y \,.\, c + r \wedge r < y \wedge r \geq 0$, por CONS se llega a:

$$\{x \geq 0 \wedge y > 0\}\ S_{div}\ \{x = y \,.\, c + r \wedge r < y \wedge r \geq 0\}$$

Dada la precondición $\{x \geq 0 \wedge y > 0\}$ y la postcondición $\{x = y \,.\, c + r \wedge r < y \wedge r \geq 0\}$, usamos el invariante $\{x = y \,.\, c + r \wedge r \geq 0\}$:

Prueba de (a)

1. $\{x = y \,.\, c + x \wedge x \geq 0\}\ r := x\ \{x = y \,.\, c + r \wedge r \geq 0\}$	(ASI)
2. $\{x = y \,.\, 0 + x \wedge x \geq 0\}\ c := 0\ \{x = y \,.\, c + x \wedge x \geq 0\}$	(ASI)
3. $\{x = y \,.\, 0 + x \wedge x \geq 0\}\ c := 0\ ;\ r := x\ \{x = y \,.\, c + r \wedge r \geq 0\}$	(1, 2, SEC)
4. $(x \geq 0 \wedge y > 0) \rightarrow (x = y \,.\, 0 + x \wedge x \geq 0)$	(MAT)
5. $\{x \geq 0 \wedge y > 0\}\ c := 0\ ;\ r := x\ \{x = y \,.\, c + r \wedge r \geq 0\}$	(3, 4, CONS)

Prueba de (b)

6. $\{x = y \,.\, (c + 1) + r \wedge r \geq 0\}\ c := c + 1\ \{x = y \,.\, c + r \wedge r \geq 0\}$	(ASI)
7. $\{x = y \,.\, (c + 1) + (r - y) \wedge r - y \geq 0\}\ r := r - y\ \{x = y \,.\, (c + 1) + r \wedge r \geq 0\}$	(ASI)
8. $\{x = y \,.\, (c + 1) + (r - y) \wedge r - y \geq 0\}\ r := r - y\ ;\ c := c + 1\ \{x = y \,.\, c + r \wedge r \geq 0\}$	(6, 7, SEC)
9. $(x = y \,.\, c + r \wedge r \geq 0 \wedge r \geq y) \rightarrow (x = y \,.\, (c + 1) + (r - y) \wedge r - y \geq 0)$	(MAT)
10. $\{x = y \,.\, c + r \wedge r \geq 0 \wedge r \geq y\}\ r := r - y\ ;\ c := c + 1\ \{x = y \,.\, c + r \wedge r \geq 0\}$	(8, 9, CONS)
11. $\{x = y \,.\, c + r \wedge r \geq 0\}$ while $r \geq y$ do $r := r - y\ ;\ c := c + 1$ od $\{x = y \,.\, c + r \wedge r \geq 0 \wedge \neg(r \geq y)\}$	(10, REP)

Prueba de (c)

12. $\{x \geq 0 \wedge y > 0\}\ c := 0\ ;\ r := x\ ;$ while $r \geq y$ do $r := r - y\ ;\ c := c + 1$ od $\{\ x = y \,.\, c + r \wedge r \geq 0 \wedge \neg(r \geq y)\}$	(5, 11, SEC)
13. $(x = y \,.\, c + r \wedge r \geq 0 \wedge \neg(r \geq y)) \rightarrow (x = y \,.\, c + r \wedge r < y \wedge r \geq 0)$	(MAT)
14. $\{x \geq 0 \wedge y > 0\}\ c := 0\ ;\ r := x\ ;$ while $r \geq y$ do $r := r - y\ ;\ c := c + 1$ od $\{x = y \,.\, c + r \wedge r < y \wedge r \geq 0\}$	(12, 13, CONS)

También se puede probar $\{x \geq 0 \wedge y \geq 0\}\ S_{div}\ \{x = y \,.\, c + r \wedge r < y \wedge r \geq 0\}$, permitiendo que el divisor *y* sea 0. La prueba queda como ejercicio.

Ejemplo 2. Prueba de un programa que calcula el factorial de un número natural

- Vamos a probar:

$$\{x > 0\}$$
$$S_{fac} :: a := 1 ; y := 1 ; \text{while } a < x \text{ do } a := a + 1 ; y := y \,.\, a \text{ od}$$
$$\{y = x!\}$$

Es decir, dado $x > 0$, el programa S_{fac}, si termina, obtiene $y = x!$

Ejercicio: ¿Es correcta la especificación ($x > 0$, $y = x!$) para un programa que calcule el factorial?

- Se propone como **invariante** del while:

$$p = (y = a! \wedge a \leq x)$$

La idea es que se vaya calculando en *y* el factorial de *a*, hasta que a = x. En efecto, notar que el invariante propuesto es similar a la postcondición, la generaliza, sustituyendo x por *a*. Cuando a = x, entonces y = x!.

- La prueba se puede estructurar de la siguiente manera:

a) $\{x > 0\}\ a := 1 ; y := 1\ \{y = a! \wedge a \leq x\}$ — Se cumple el invariante por primera vez

b) $\{y = a! \wedge a \leq x\}$
while a < x do a := a + 1 ; y := y . a od
$\{y = x!\}$ — Aplicación de REP y CONS

c) $\{x > 0\}\ S_{fac}\ \{y = x!\}$ — Aplicación de SEC sobre (a) y (b)

$$\frac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \text{while } B \text{ do } S \text{ od } \{p \wedge \neg B\}}$$

Recordatorio de la regla REP

Seguimos del slide anterior.
Prueba de $\{x > 0\}\ S_{fac}$ :: a := 1 ; y := 1 ; while **a < x** do a := a + 1 ; y := y . a od $\{y = x!\}$
Utilizamos el invariante $p = (y = a! \wedge a \leq x)$ para la aplicación de la regla REP:

Prueba de (a)

1. $\{1 = a! \wedge a \leq x\}$ y := 1 $\{y = a! \wedge a \leq x\}$	(ASI)
2. $\{1 = 1! \wedge 1 \leq x\}$ a := 1 $\{1 = a! \wedge a \leq x\}$	(ASI)
3. $\{x > 0\}$ a := 1; y := 1 $\{y = a! \wedge a \leq x\}$	(1,2,SEC,CONS)

Hemos abreviado pasos. En el paso 3 se usa CONS considerando $(x > 0) \longrightarrow (1 = 1! \wedge 1 \leq x)$.

Prueba de (b)

4. $\{y . a = a! \wedge a \leq x\}$ y := y . a $\{y = a! \wedge a \leq x\}$	(ASI)
5. $\{y . (a + 1) = (a + 1)! \wedge (a + 1) \leq x\}$ a := a + 1 $\{y . a = a! \wedge a \leq x\}$	(ASI)
6. $\{y = a! \wedge a \leq x \wedge$ **a < x**$\}$ a := a + 1; y := y . a $\{y = a! \wedge a \leq x\}$	(4,5,SEC,CONS)
7. $\{y = a! \wedge a \leq x\}$ while **a < x** do a := a + 1; y := y . a od $\{y = a! \wedge a \leq x \wedge$ **¬(a < x)**$\}$	(6,REP)
8. $\{y = a! \wedge a \leq x\}$ while a < x do a := a + 1; y := y . a od $\{y = x!\}$	(7,CONS)

En el paso 6 se usa CONS considerando $(y = a! \wedge a \leq x \wedge a < x) \longrightarrow (y . (a + 1) = (a + 1)! \wedge (a + 1) \leq x)$.
En el paso 8 se usa CONS considerando $(y = a! \wedge a \leq x \wedge \neg(a < x)) \longrightarrow (y = x!)$.

Prueba de (c)

9. $\{x > 0\}\ S_{fac}\ \{y = x!\}$	(3,8,SEC)