Clase teórica 12

Correctitud total
Sensatez y completitud

Repaso e introducción

- Hemos presentado el método H para la prueba de correctitud parcial de un programa S con respecto a una especificación (p, q), es decir {p} S {q}:

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

- En esta clase vamos a presentar el **método H*** para la prueba de **correctitud total**, es decir **⟨p⟩ S ⟨q⟩:**

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

- Como el while es la única instrucción que puede provocar no terminación (o divergencia), en realidad H* consiste en la **extensión** de H con una regla más que asegure la terminación de un while.

- Las dos reglas se separan por basarse en técnicas de prueba distintas. La recomendación es probar la fórmula ⟨p⟩ S ⟨q⟩ probando por separado **{p} S {q} y ⟨p⟩ S ⟨true⟩.**

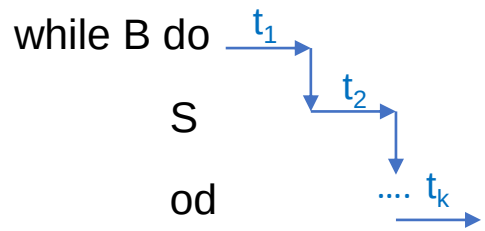
- En la segunda parte de la clase vamos a probar la **sensatez** y la **completitud** de los métodos.

Método H*

- H* difiere de H sólo en la regla asociada al while. Cuenta con los axiomas SKIP y ASI y las reglas SEC, COND y CONS de H. Para distinguirlos, utilizamos * en los nombres, y los delimitadores ⟨ ⟩ en lugar de { }). Por ejemplo: **SKIP*: ⟨p⟩ skip ⟨p⟩, ASI*: ⟨p[x|e]⟩ x := e ⟨p⟩,** etc.

- La regla de repetición de H*, que llamamos **REP*,** se basa como antes en un invariante p, pero ahora también en un **variante** t, función entera que se define en términos de las variables de programa (**representa en todo momento la cantidad máxima de iteraciones del while**). Su forma es:

$$\frac{\langle p \wedge B\rangle\ S\ \langle p\rangle\ ,\ \mathbf{\langle p \wedge B \wedge t = Z\rangle\ S\ \langle t < Z\rangle\ ,\ p \rightarrow t \geq 0}}{\langle p\rangle \text{ while } B \text{ do } S \text{ od } \langle p \wedge \neg B\rangle}$$



El valor de t decrece de iteración en iteración. Como t es una función entera positiva, indefectiblemente la cadena descendente de los t_i es finita.

Es decir, se agregan dos premisas a la regla REP del método H.
La variable Z es una variable lógica, no aparece en p, B, t ni S,
y su objetivo es fijar el valor de t antes de la ejecución de S.

Por la segunda premisa, **t se decrementa en cada iteración.**

Por la tercera premisa, **t arranca ≥ 0, y se mantiene ≥ 0 después de cada iteración.**

De esta manera, **el while debe terminar** (como antes, con postcondición p ∧ ¬B). El fundamento es que no hay cadenas descendentes infinitas en el dominio de los números naturales con respecto a la relación <. Dicho dominio se anota así: **(N, <).** Es un ejemplo de **conjunto bien fundado**: conjunto con una relación de orden estricto sin cadenas descendentes infinitas.

Ejemplo. Prueba de terminación del programa del factorial.

- Con el método H hemos probado:

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

  utilizando el invariante **$p = (y = a! \wedge a \leq x)$**

Regla REP*

$$\frac{\langle p \wedge B\rangle\ S\ \langle p\rangle\ ,\ \langle p \wedge B \wedge t = Z\rangle\ S\ \langle t < Z\rangle\ ,\ p \rightarrow t \geq 0}{\langle p\rangle\ \text{while } B \text{ do } S \text{ od } \langle p \wedge \neg B\rangle}$$

- Con el método H* probaremos:

  $\langle x > 0\rangle\ S_{fac} :: a := 1 ; y := 1 ; \text{while } a < x \text{ do } a := a + 1 ; y := y \cdot a \text{ od } \langle true\rangle$

  y de esta manera habremos probado:

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

- Se propone:

Invariante **$p = (a \leq x)$**	Al comienzo del while, $a \leq x$, y a la salida, $a = x$. Notar que este invariante es más simple que el utilizado antes.
Variante **$t = x - a$**	Al comienzo del while, $t \geq 0$. A lo largo de sus iteraciones, t se decrementa en 1. Y al final se cumple $t = 0$.

- La prueba se desarrolla en el slide siguiente:

Inicializaciones:
1. $\langle x > 0 \rangle$ a := 1 ; y := 1 $\langle a \leq x \rangle$ (ASI*, SEC*, CONS*)

Repetición:
Se tiene: $p = (a \leq x)$, $t = x - a$
Hay que probar por REP* tres premisas:

Regla REP*

$$\frac{\langle p \wedge B \rangle\ S\ \langle p \rangle\ ,\ \langle p \wedge B \wedge t = Z \rangle\ S\ \langle t < Z \rangle\ ,\ p \rightarrow t \geq 0}{\langle p \rangle\ \text{while } B \text{ do } S \text{ od } \langle p \wedge \neg B \rangle}$$

Premisa 1: $\langle p \wedge B \rangle\ S\ \langle p \rangle$
2. $\langle a \leq x \wedge a < x \rangle$ a := a + 1 ; y := y . a $\langle a \leq x \rangle$ (ASI*, SEC*, CONS*)

Premisa 2: $\langle p \wedge B \wedge t = Z \rangle\ S\ \langle t < Z \rangle$
3. $\langle a \leq x \wedge a < x \wedge x - a = Z \rangle$ a := a + 1 ; y := y . a $\langle x - a < Z \rangle$ (ASI*, SEC*, CONS*)

Premisa 3: $p \rightarrow t \geq 0$
4. $a \leq x \rightarrow x - a \geq 0$ (MAT)

Conclusión: $\langle p \rangle$ while B do S od $\langle p \wedge \neg B \rangle$
5. $\langle a \leq x \rangle$ while a < x do a := a + 1 ; y := y . a od $\langle a \leq x \wedge \neg(a < x) \rangle$ (2, 3, 4, REP*)

Programa completo:
6. $\langle x > 0 \rangle$ a :=1 ; y := 1 ; while a < x do a := a + 1 ; y := y . a od $\langle true \rangle$ (1, 5, SEC*, CONS*)

Programa

```
a :=1 ;
y := 1 ;
while a < x
do
a := a + 1 ;
y := y . a
od
```

Proof outline

```
⟨x > 0⟩
a :=1 ; y := 1 ;
⟨inv: a ≤ x, var: x - a⟩
while a < x
do
a := a + 1 ; y := y . a
od
⟨a ≤ x ∧ ¬(a < x)⟩
⟨true⟩
```

En realidad, se probó $\langle x > 0 \rangle\ S_{fac}\ \langle a = x \rangle$. Pero la postcondición en este caso es irrelevante, el objetivo fue asegurar la terminación del while. La postcondición y = x! se alcanzó en la prueba de correctitud parcial.

Sensatez y completitud de los métodos H y H*

- Se cumplen ambas propiedades en ambos métodos:

1. Sensatez: para todo programa S y para toda especificación (p, q):

$$\mathbf{Tr} \vdash_{H} \{p\}\ S\ \{q\} \rightarrow \models \{p\}\ S\ \{q\}$$
$$\mathbf{Tr} \vdash_{H^*} \langle p\rangle\ S\ \langle q\rangle \rightarrow \models \langle p\rangle\ S\ \langle q\rangle$$

Se suele anteponer la expresión Tr (por true) para indicar que el método incluye todos los axiomas del dominio semántico.

Las fórmulas de correctitud probadas por H y H* son verdaderas.

2. Completitud: para todo programa S y para toda especificación (p, q):

$$\models \{p\}\ S\ \{q\} \rightarrow \mathbf{Tr} \vdash_{H} \{p\}\ S\ \{q\}$$
$$\models \langle p\rangle\ S\ \langle q\rangle \rightarrow \mathbf{Tr} \vdash_{H^*} \langle p\rangle\ S\ \langle q\rangle$$

Las fórmulas de correctitud verdaderas pueden ser probadas por H y H*.

- En lo que sigue presentamos parte de la prueba del cumplimiento de dichas propiedades.

- Recurrimos fundamentalmente a la técnica de **inducción**. Antes de avanzar con la prueba, pasamos a repasar la técnica brevemente.

- **Inducción Matemática**

  Sea P una propiedad a probar en el dominio N de los números naturales. Si:

  (a) *base inductiva*: se cumple P(0),

  (b) *paso inductivo*: para todo k de N se cumple P(k) ⟶ P(k + 1),

  entonces para todo n de N se cumple P(n).

  P(k) en (b) se conoce como *hipótesis inductiva*. Hay una *variante fuerte* de la inducción matemática, que tiene como paso inductivo: (P(i) ∧ P(i + 1) ∧ … ∧ P(k – 1) ∧ P(k)) ⟶ P(k + 1), para algún i tal que $0 \leq i < k$.

  Ejercicio: probar que para todo n de N se cumple: $0 + 1 + 2 + 3 + \ldots + n = \frac{n.(n + 1)}{2}$

- **Inducción Estructural**

  Es una generalización de la anterior, útil para probar propiedades y también para definir conjuntos. Se aplica a cualquier dominio, con una relación determinada <. Puede haber más de un minimal, una base inductiva y un paso inductivo.

  Ejemplo. Definición inductiva de una expresión aritmética e, del tipo: ((1 + (0 + x)) **.** (x **.** 1))

  (a) *base inductiva*: 0, 1, x, son expresiones aritméticas.

  (b) *paso inductivo*: si e_1 y e_2 son expresiones aritméticas, también lo son $(e_1 + e_2)$ y $(e_1 \,.\, e_2)$.

  Ejercicio: probar que la cantidad C de constantes y variables de una expresión aritmética *e* supera en 1 a la cantidad O de operadores de *e*, es decir: C(e) = 1 + O(e). P. ej., en la expresión anterior: C(e) = 1 + O(e) = 5.

Sensatez del método H

Para todo S y todo par p y q, se cumple: $\vdash_H \{p\}\ S\ \{q\} \rightarrow \models \{p\}\ S\ \{q\}$.

- Se prueba por inducción matemática (fuerte), considerando la **longitud de la prueba** (1 o más pasos).
  (a) *Base induc*tiva: los **axiomas** son verdaderos (las pruebas miden 1).
  (b) *Paso inductivo*: las **reglas** son sensatas, preservan la verdad (las pruebas miden 2 o más).

- Ejemplo de (a). Prueba de que el axioma **SKIP** es verdadero.

  - Dado $\vdash \{p\}$ skip $\{p\}$, hay que probar **$\models \{p\}$ skip $\{p\}$.**
  - Por la semántica de PLW: (skip, σ) → (E, σ).
  - Sea **σ |= p.** Luego del skip, se cumple **σ |= p.**

- Ejemplo de (b). Prueba de que la regla **SEC** es sensata.

  - Dado $\vdash \{p\}\ S_1 ; S_2\ \{q\}$, hay que probar $\models \{p\}\ S_1 ; S_2\ \{q\}$.
  - $\vdash \{p\}\ S_1 ; S_2\ \{q\}$ se obtiene de $\vdash \{p\}\ S_1\ \{r\}$ y $\vdash \{r\}\ S_2\ \{q\}$ (**pruebas más cortas**).
  - Hipótesis inductiva: **$\models \{p\}\ S_1\ \{r\}$** y **$\models \{r\}\ S_2\ \{q\}$.** Veamos que **$\models \{p\}\ S_1 ; S_2\ \{q\}$.**
  - Sea **$\sigma_1 \models p$,** y asumamos que $S_1 ; S_2$ termina desde σ_1 (si no temina, la terna se cumple trivialmente).
  - Por la semántica de PLW y la hipótesis inductiva: $(S_1 ; S_2, \sigma_1) \rightarrow^* (S_2, \sigma_2) \rightarrow^* (E, \sigma_3)$, con **$\sigma_3 \models q$,** que es lo que queríamos demostrar.

Queda como ejercicio completar la prueba (axioma ASI y reglas COND, REP y CONS).

Completitud del método H

Para todo S y todo par p y q, se cumple: $\models \{p\}\ S\ \{q\} \rightarrow \vdash_H \{p\}\ S\ \{q\}$.

- Se prueba por inducción estructural, considerando las **5 formas de los programas PLW**.
  (a) *Base induc*tiva: considera los **programas atómicos** skip y x := e.
  (b) *Paso inductivo*: considera los **programas compuestos** con la forma de secuencia, if then else y while.

- Ejemplo de (a). Prueba de: **$\models \{p\}$ skip $\{q\} \rightarrow \vdash \{p\}$ skip $\{q\}$.**

  - Se tiene $\models \{p\}$ skip $\{q\}$.
  - Como por la semántica del skip vale $\models \{p\}$ skip $\{p\}$, entonces necesariamente $p \rightarrow q$.
  - Así se obtiene **$\vdash \{p\}$ skip $\{q\}$:**
    1. $\{p\}$ skip $\{p\}$ (SKIP)
    2. $p \rightarrow q$ (MAT)
    3. $\{p\}$ skip $\{q\}$ (CONS, 1, 2)

- Ejemplo de (b). Prueba de: **$\models \{p\}$ if B then S_1 else S_2 fi $\{q\} \rightarrow \vdash \{p\}$ if B then S_1 else S_2 fi $\{q\}$.**

  - Se tiene $\models \{p\}$ if B then S_1 else S_2 fi $\{q\}$.
  - Por la semántica del if then else vale $\models \{p \wedge B\}\ S_1\ \{q\}$ y $\models \{p \wedge \neg B\}\ S_2\ \{q\}$ (**componentes más simples**).
  - Hipótesis inductiva: $\vdash \{p \wedge B\}\ S_1\ \{q\}$ y $\vdash \{p \wedge \neg B\}\ S_2\ \{q\}$.
  - Así se obtiene **$\vdash \{p\}$ if B then S_1 else S_2 fi $\{q\}$,** aplicando la regla COND.

- Otro ejemplo de (b). Prueba de: **$\models \{p\}\ S_1 ; S_2\ \{q\} \rightarrow \vdash \{p\}\ S_1 ; S_2\ \{q\}$.**

  - Se tiene $\models \{p\}\ S_1 ; S_2$ fi $\{q\}$.
  - Por la semántica de la secuencia, debe existir r tal que $\models \{p\}\ S_1\ \{r\}$ y $\models \{r\}\ S_2\ \{q\}$.
  - Hipótesis inductiva: $\vdash \{p\}\ S_1\ \{r\}$ y $\vdash \{r\}\ S_2\ \{q\}$.
  - Así se obtiene **$\vdash \{p\}\ S_1 ; S_2\ \{q\}$,** aplicando la regla SEC.

¿Pero cómo se puede asegurar que r se puede expresar?

- Se puede asegurar porque se demuestra que el lenguaje de predicados es **expresable** con respecto al lenguaje de programación utilizado y la interpretación de los números enteros: dada cualquier precondición p y cualquier programa S, siempre se puede expresar la postcondición q (lo que se denota con **post(p, S)**).

- El mismo problema aparece en la prueba de **$\models \{p\}$ while B do S od $\{q\} \rightarrow \vdash \{p\}$ while B do S od $\{q\}$,** a la hora de expresar el invariante del while. La expresividad provista por el método definido lo resuelve.

- Por eso se utiliza el término **completitud relativa** (o ***en el sentido de Cook***): la completitud de un método de prueba depende de los lenguajes utilizados y la interpretación de las variables.

- Otra posible causa de incompletitud es la **incompletitud relacionada con la interpretación de las variables** (como es nuestro caso, probado por el teorema de incompletitud de Gödel). Este aspecto es insoslayable, por eso lo que se hace es asumir que se cuenta con el conjunto de todos los axiomas de la interpretación.

Sensatez del método H*

- Sólo falta probar el caso de la regla REP*: **$\vdash \langle p \rangle$ while B do S od $\langle p \wedge \neg B \rangle \rightarrow \models \langle p \rangle$ while B do S od $\langle p \wedge \neg B \rangle$:**

La prueba $\vdash \langle p \rangle$ while B do S od $\langle p \wedge \neg B \rangle$ se obtiene de:
$\vdash \langle p \wedge B \rangle$ S $\langle p \rangle$, $\vdash \langle p \wedge B \wedge t = Z \rangle$ S $\langle t < Z \rangle$, $p \rightarrow t \geq 0$.

Hipótesis inductiva:
$\models \langle p \wedge B \rangle$ S $\langle p \rangle$, $\models \langle p \wedge B \wedge t = Z \rangle$ S $\langle t < Z \rangle$, $\models p \rightarrow t \geq 0$.

Regla REP*

$$\frac{\langle p \wedge B \rangle \; S \; \langle p \rangle \;,\; \langle p \wedge B \wedge t = Z \rangle \; S \; \langle t < Z \rangle \;,\; p \rightarrow t \geq 0}{\langle p \rangle \text{ while B do S od } \langle p \wedge \neg B \rangle}$$

Veamos que se cumple **$\models \langle p \rangle$ while B do S od $\langle p \wedge \neg B \rangle$.**

- Sea **$\sigma_0 \models p$.** Probaremos que el while termina, desde σ_0, en un estado σ_k tal que $\sigma_k \models p \wedge \neg B$.

- Supongamos por el absurdo que desde σ_0 el while no termina. Llegaremos a una contradicción.

- Si el while no termina desde σ_0, la computación infinita correspondiente tiene la forma siguiente:
(while B do S od, σ_0) $\rightarrow^*$ (while B do S od, σ_1) $\rightarrow^*$ … $\rightarrow^*$ (while B do S od, σ_i) $\rightarrow^*$ …
con $\sigma_i \models ((p \wedge B) \wedge t \geq 0)$, porque $\models \langle p \wedge B \rangle$ S $\langle p \rangle$ y $\models p \rightarrow t \geq 0$.

- Por lo tanto, también es infinita la cadena: **$\sigma_0(t), \sigma_1(t), \ldots, \sigma_i(t), \ldots$**

- Como $\models \langle p \wedge B \wedge t = Z \rangle$ S $\langle t < Z \rangle$, se cumple que $\sigma_i(t) > \sigma_{i+1}(t)$ para todo $i \geq 0$. Y por lo tanto la cadena infinita de los $\sigma_i(t)$ es descendente, absurdo porque t es una función entera y siempre tiene un valor positivo.

- Así, **el while termina desde σ_0,** y el estado final σ_k cumple **$\sigma_k \models p \wedge \neg B$.**

Completitud del método H*

- Nuevamente, sólo hay que considerar la regla REP*:

$$\models \langle p \rangle \textbf{ while B do S odS } \langle q \rangle \rightarrow \vdash_{H^*} \langle p \rangle \textbf{ while B do S od } \langle q \rangle$$

- Ya consideramos la expresividad del invariante. Falta considerar la expresividad del **variante**.

  En este caso, lo que se requiere es que el lenguaje de especificación permita expresar con una función entera **la cantidad de iteraciones de un while**, y de esta manera de **cualquier función computable**, lo que se justifica por lo siguiente:

  - Dado un programa S :: while B do T od, y un estado inicial σ,
    sea **iter(S, σ)** una función parcial que define el número de iteraciones de S a partir de σ.

  ¿Por qué iter es una función parcial?

  - Notar que la función iter(S, σ) es **computable.** Sea:

$$\textbf{S}_x \textbf{ :: x := 0 ; while B do x := x + 1 ; T od}$$

  - Si S termina a partir de σ en un estado σ´, **iter(S, σ) = σ´(x)** contiene la cantidad de iteraciones del while.

Anexo de la clase teórica 12

Correctitud total
Sensatez y completitud

Propiedades safety y liveness

- Ya vimos que 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 son la *ausencia de deadlock* y la *exclusión mutua*, en los programas concurrentes.

- La terminación o no divergencia, en cambio, pertenece a la familia de las propiedades **liveness**.

  Son propiedades que no se prueban por inducción, sino que se basan en una función variante definida en un **orden bien fundado.**

  Se asocian al enunciado: “Algo bueno va a suceder”.

  Otro ejemplo de propiedad *liveness* es la *ausencia de inanición (non starvation),* en los programas concurrentes.

Tipos de sensatez y completitud

- La sensatez de H se cumple **independientemente de la interpretación semántica considerada**. Notar que en ningún momento de las pruebas recurrimos al dominio de las variables de programa (en nuestro caso los números enteros), salvo cuando utilizamos sus axiomas (que podrían ser de otro dominio semántico). Por ejemplo, en el programa de división visto antes, la prueba también aplica si las variables son de tipo real.

- En efecto, H tiene **sensatez total**, lo que se formula de la siguiente manera:

$$\mathbf{Tr_I \vdash_H \{p\}\ S\ \{q\} \rightarrow \models_I \{p\}\ S\ \{q\},\ para\ toda\ interpretación\ I}$$

  Tr_I tiene todos los axiomas correspondientes a la interpretación I.

  Que valga $\models_I \{p\}\ S\ \{q\}$ significa que $\{p\}\ S\ \{q\}$ es verdadera considerando I.

- En cambio, la sensatez de H* se conoce como **aritmética**. Tal como está planteada la regla REP*:

  - El dominio semántico en el que se interpretan los programas **debe extenderse con el de los números naturales**, en el que varía el variante t.
  - Correspondientemente, el lenguaje de especificación **debe extenderse con el de la aritmética**.

  En el slide siguiente mostramos un ejemplo de prueba de terminación con variables de tipo real:

Ejemplo de prueba de terminación en el dominio de los números reales

- Apartándonos por un momento del dominio semántico de los números enteros, sea el siguiente programa S en el que las variables x y Ɛ son de tipo **real,** inicialmente mayores que 0:

```
S :: while x > Ɛ
        do
        x := x / 2
        od
```

- Claramente se cumple: **⟨x = X ∧ X > 0 ∧ Ɛ > 0⟩ S ⟨true⟩.**

- Los distintos valores positivos de x constituyen iteración tras iteración una secuencia decreciente estricta.

- También en este caso se puede usar REP*, definiendo un adecuado variante t en el dominio de los números naturales.

- Una posible función t podría ser la siguiente:

$$t = \text{if } x > Ɛ \text{ then } [\log_2 X/Ɛ] - \log_2 X/x \text{ else } 0 \text{ fi}$$

Ejercicio: calcular los distintos valores de t a partir de X = 10 y Ɛ = 0,1.

Clase práctica 12

Correctitud total
Sensatez y completitud

Ejemplo 1. Prueba de terminación del programa de división entera por restas sucesivas

Ya hemos probado mediante el método H:

$\{x \geq 0 \wedge y > 0\}$
S_{div} :: c := 0 ; r := x ; while r ≥ y do r := r – y ; c := c + 1 od
$\{x = c.y + r \wedge 0 \leq r < y\}$

Notar que también se puede probar con $y \geq 0$ (ejercicio).

Ahora probaremos la terminación de S_{div} empleando REP*. Naturalmente acá necesariamente la precondición debe incluir y > 0. Vamos a verificar:

$$\langle x \geq 0 \wedge y > 0 \rangle \; S_{div} \; \langle true \rangle$$

Se propone como **invariante** del while la aserción: **p = (x = c . y + r $\wedge$ r ≥ 0 $\wedge$ y > 0).** En este caso el invariante no es más simple que el de la prueba de correctitud parcial.

Y se propone como **variante** del while la función: **t = r.** Claramente t siempre es positiva y se decrementa en cada iteración.

Ver la prueba en el slide siguiente:

La prueba es la siguiente (para simplificar la escritura utilizamos directamente p en lugar de la aserción $x = c.y + r \wedge r \geq 0 \wedge y > 0$):

Especificación y programa S_{div}

$\{x \geq 0 \wedge y > 0\}$

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

$\{x = c.y + r \wedge 0 \leq r < y\}$

regla REP*

$$\frac{\langle p \wedge B\rangle\ S\ \langle p\rangle\ ,\ \langle p \wedge B \wedge t = Z\rangle\ S\ \langle t < Z\rangle\ ,\ p \rightarrow t \geq 0}{\langle p\rangle \text{ while } B \text{ do } S \text{ od } \langle p \wedge \neg B\rangle}$$

Prueba del fragmento inicial del programa:

1. $\langle x \geq 0 \wedge y > 0\rangle$ c := 0 ; r := x $\langle p\rangle$ (ASI*, SEC*, CONS*)

Prueba de las tres premisas de REP*:

2. $\langle p \wedge r \geq y\rangle$ r := r – y ; c := c + 1 $\langle p\rangle$ (ASI*, SEC*, CONS*)
3. $\langle p \wedge r \geq y \wedge r = Z\rangle$ r := r – y ; c := c + 1 $\langle r < Z\rangle$ (ASI*, SEC*, CONS*)
4. $p \rightarrow r \geq 0$ (MAT)

Se establece la conclusión de REP* y la fórmula final pretendida:

5. $\langle p\rangle$ while r ≥ y do r := r – y ; c := c + 1 od $\langle p \wedge \neg(r \geq y)\rangle$ (2, 3, 4, REP*)
6. $\langle x \geq 0 \wedge y > 0\rangle$ S_{div} $\langle$true$\rangle$ (1, 5, SEC*, CONS*)

Ejemplo 2. Sensatez de la Regla de la Disyunción (OR)

La regla OR establece, dadas aserciones p, q, r, y un programa S:

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

Probaremos la sensatez de la regla OR por inducción sobre la longitud de la prueba. Hay que probar:

$\vdash \{p \vee r\}\ S\ \{q\} \rightarrow \models \{p \vee r\}\ S\ \{q\}$, habiendo aplicado OR

- $\vdash \{p \vee r\}\ S\ \{q\}$ proviene de $\vdash \{p\}\ S\ \{q\}$ y $\vdash \{r\}\ S\ \{q\}$, aplicando OR.
- Hipótesis inductiva: (1) $\models \{p\}\ S\ \{q\}$, (2) $\models \{r\}\ S\ \{q\}$.
- Veamos que se cumple **$\models \{p \vee r\}\ S\ \{q\}$**:
- Sea **$\sigma \models p \vee r$**, y asumamos que S termina desde σ en un estado σ´.
- Hay dos posibilidades, lo que completa la prueba:

  Si $\sigma \models p$, entonces por (1) vale **$\sigma' \models q$.**

  Si $\sigma \models r$, entonces por (2) vale **$\sigma' \models q$.**

Ejemplo 3. No sensatez de la siguiente regla UNTIL

$$\frac{\{p \wedge \neg B\}\ S\ \{p\}}{\{p\}\ \text{repeat } S \text{ until } B\ \{p \wedge B\}}$$

La semántica de la instrucción repeat es:

repeat S until B = S ; while ¬ B do S od

Veamos que la regla **no es sensata**, mediante un contraejemplo.

Supongamos **|= {p ∧ ¬ B} S {p}** (hipótesis inductiva).

A partir de ella, **encontraremos un caso en que no valga la fórmula |= {p} repeat S until B {p ∧ B}.**

La idea es tener en cuenta que la premisa de la regla asegura que S preserva p **sólo a partir de ¬B**. Efectivamente, podría suceder que **a partir de p ∧ B, S no preserve p**.

Veamos el desarrollo de la prueba en el slide siguiente:

Sean:

{p ∧ ¬ B} S {p}
{p} repeat S until B {p ∧ B}

p = (x = 0)

B = (y = 0)

S :: if ¬(y = 0) then skip else x := 1 fi (notar que S no preserva p cuando vale B)

Veamos que se cumple |= {p ∧ ¬ B} S {p} pero que no se cumple |= {p} repeat S until B {p ∧ B}:

1. Se cumple |= {p ∧ ¬ B} S {p}:

|= {x = 0 ∧ ¬(y = 0)} if ¬(y = 0) then skip else x := 1 fi {x = 0}

2. No se cumple |= {p} repeat S until B {p ∧ B}:

Es decir, no se cumple |= {x = 0} repeat if ¬(y = 0) then skip else x := 1 fi until y = 0 {x = 0 ∧ y = 0}:

Si el estado inicial σ cumple: |= (x = 0 ∧ y = 0), entonces el repeat se ejecuta sólo una vez, y el estado final σ' cumple: σ' |= (x = 1 ∧ y = 0).