Practica 5 – Clases 11 – 12

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 { }).

REP* ahora se basa como antes en un invariante p, y 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:

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

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.

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

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! ∧ a ≤ 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 ≤ x)**	Al comienzo del while, a ≤ 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 ≥ 0. A lo largo de sus iteraciones, t se decrementa en 1. Y al final se cumple t = 0.

Inicializaciones:
1. ⟨x > 0⟩ a := 1 ; y := 1 ⟨a ≤ x⟩ (ASI*, SEC*, CONS*)

Repetición:
Se tiene: p = (a ≤ 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: ⟨p ∧ B⟩ S ⟨p⟩
2. ⟨a ≤ x ∧ a < x⟩ a := a + 1 ; y := y . a ⟨a ≤ x⟩ (ASI*, SEC*, CONS*)

Premisa 2: ⟨p ∧ B ∧ t = Z⟩ S ⟨t < Z⟩
3. ⟨a ≤ x ∧ a < x ∧ x – a = Z⟩ a := a + 1 ; y := y . a ⟨x – a < Z⟩ (ASI*, SEC*, CONS*)

Premisa 3: p → t ≥ 0
4. a ≤ x → x – a ≥ 0 (MAT)

Conclusión: ⟨p⟩ while B do S od ⟨p ∧ ¬B⟩
5. ⟨a ≤ x⟩ while a < x do a := a + 1 ; y := y . a od ⟨a ≤ x ∧ ¬(a < x)⟩ (2, 3, 4, REP*)

Programa completo:
6. ⟨x > 0⟩ a :=1 ; y := 1 ; while a < x do a := a + 1 ; y := y . a od ⟨true⟩ (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 ⟨x > 0⟩ S_{fac} ⟨a = x⟩. 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*

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

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

Las fórmulas de correctitud probadas por H y H* son verdaderas. Se suele anteponer la expresión Tr (por true) para indicar que el método incluye todos los axiomas del dominio semántico.

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

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

Las fórmulas de correctitud verdaderas pueden ser probadas por H y H*.
Se recurre a la técnica de inducción. Antes de avanzar con la prueba, pasamos a repasar la técnica brevemente.

Sensatez del método H

Para todo S y todo par p y q, se cumple: |– H {p} S {q} → |= {p} S {q}. Se prueba por inducción matemática (fuerte), considerando la longitud de la prueba (1 o más pasos).

a. Base inductiva: 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 |– {p} skip {p}, hay que probar |= {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 |– {p} S1 ; S2 {q}, hay que probar |= {p} S1 ; S2 {q}.
- |– {p} S1 ; S2 {q} se obtiene de |– {p} S1 {r} y |– {r} S2 {q} (pruebas más cortas).
- Hipótesis inductiva: |= {p} S1 {r} y |= {r} S2 {q}. Veamos que |= {p} S1 ; S2 {q}.
- Sea σ1 |= p, y asumamos que S1 ; S2 termina desde σ1 (si no temina, la terna se cumple trivialmente).
- Por la semántica de PLW y la hipótesis inductiva: (S1 ; S2 , σ1) →* (S2 , σ2) →* (E, σ3), con σ3 |= q, que es lo que queríamos demostrar.

Completitud del método H

Para todo S y todo par p y q, se cumple: |= {p} S {q} → |– H {p} S {q}. Se prueba por inducción estructural, considerando las 5 formas de los programas PLW.

a. Base inductiva: 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: |= {p} skip {q} → |– {p} skip {q}.
- Se tiene |= {p} skip {q}.
- Como por la semántica del skip vale |= {p} skip {p}, entonces necesariamente p → q.
- Así se obtiene |– {p} skip {q}:
    1. {p} skip {p} (SKIP)
    2. p → q (MAT)
    3. {p} skip {q} (CONS, 1, 2)

Ejemplo de (b). Prueba de: |= {p} if B then S1 else S2 fi {q} → |– {p} if B then S1 else S2 fi {q}.
- Se tiene |= {p} if B then S1 else S2 fi {q}.
- Por la semántica del if then else vale |= {p ∧ B} S1 {q} y |= {p ∧ ¬B} S2 {q} (componentes más simples).
- Hipótesis inductiva: |– {p ∧ B} S1 {q} y |– {p ∧ ¬B} S2 {q}.
- Así se obtiene |– {p} if B then S1 else S2 fi {q}, aplicando la regla COND.

Otro ejemplo de (b). Prueba de: |= {p} S1 ; S2 {q} → |– {p} S1 ; S2 {q}.
- Se tiene |= {p} S1 ; S2 fi {q}.
- Por la semántica de la secuencia, debe existir r tal que |= {p} S1 {r} y |= {r} S2 {q}.
- Hipótesis inductiva: |– {p} S1 {r} y |– {r} S2 {q}.
- Así se obtiene |– {p} S1 ; S2 {q}, aplicando la regla SEC.

Hay que hacer esto con el resto de las instrucciones

¿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)).

La completitud de un método de prueba depende de los lenguajes utilizados y la interpretación de las variables.

Otra posible 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*: |– ⟨p⟩ while B do S od ⟨p ∧ ¬B⟩ → |= ⟨p⟩ while B do S od ⟨p ∧ ¬B⟩:

La prueba |– ⟨p⟩ while B do S od ⟨p ∧ ¬B⟩ se obtiene de:
|– ⟨p ∧ B⟩ S ⟨p⟩, |– ⟨p ∧ B ∧ t = Z⟩ S ⟨t < Z⟩, p → t ≥ 0.

Hipótesis inductiva:
|= ⟨p ∧ B⟩ S ⟨p⟩, |= ⟨p ∧ B ∧ t = Z⟩ S ⟨t < Z⟩, |= p → t ≥ 0.

Veamos que se cumple |= ⟨p⟩ while B do S od ⟨p ∧ ¬B⟩.
- Sea σ0 |= p. Probaremos que el while termina, desde σ0, en un estado σk tal que σk |= p ∧ ¬B.
- Supongamos por el absurdo que desde σ0 el while no termina. Llegaremos a una contradicción.
- Si el while no termina desde σ0, la computación infinita correspondiente tiene la forma siguiente:

(while B do S od, σ0) →* (while B do S od, σ1) →* ... →* (while B do S od, σi) →* ...

con σi |= ((p ∧ B) ∧ t ≥ 0), porque |= ⟨p ∧ B⟩ S ⟨p⟩ y |= p → t ≥ 0.
- Por lo tanto, también es infinita la cadena: σ0 (t), σ1 (t), ..., σi (t), ... }
- Como |= ⟨p ∧ B ∧ t = Z⟩ S ⟨t < Z⟩, se cumple que σi (t) > σi+1(t) para todo i ≥ 0. Y por lo tanto la cadena infinita de los σi (t) es descendente, absurdo porque t es una función entera y siempre tiene un valor positivo.
- Así, el while termina desde σ0, y el estado final σk cumple σk |= p ∧ ¬B.

Completitud del método H*

Nuevamente, sólo hay que considerar la regla REP*:

|= ⟨p⟩ while B do S odS ⟨q⟩ → |– H* ⟨p⟩ while B do S od ⟨q⟩

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:

Sx :: 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.

Propiedades safety y liveness

La correctitud parcial pertenece a la familia de las propiedades safety 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 pertenece a la familia de las propiedades liveness. 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.

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 ≥ 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 ∧ r ≥ 0 ∧ 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.

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

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

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

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**.

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).

Repaso de conceptos fundamentales

- Verificación sintáctica (axiomática) vs semántica.
- Correctitud Total = Correctitud Parcial + Terminación (o No Divergencia). Se prueban con técnicas distintas.
    - Correctitud parcial (método H). Terna de Hoare {p} S {q}. Prueba inductiva. Propiedad safety.
    - Terminación (método H*). Terna de Hoare ⟨p⟩ S ⟨true⟩. Prueba no inductiva, en base al orden bien fundado (N, >)). Propiedad liveness.
- Una instrucción de repetición (while) concretiza dos nociones:
    - Un invariante que se cumple antes, durante y después del while.
    - Una variante (función que varía en (N, >)) que se decrementa luego de cada iteración del while.

- Los métodos son composicionales. Permiten componer ternas {p} S {q} y ⟨p⟩ S ⟨q⟩ independientemente del contenido de los S (es decir que se tratan como cajas negras).
- Los métodos son sensatos (propiedad imprescindible) y completos (propiedad deseable).
- La metodología de pruebas maneja fórmulas con programas y especificaciones. No existe la noción de programa correcto aisladamente, sino con respecto a una especificación.
- Visión de método de prueba como guía para la obtención de programas correctos por construcción.

Ejemplo 2.

Antes probamos **|= {true} S {true}** empleando la definición de correctitud parcial, y entonces, por la completitud de H, probamos **|– {true} S {true}.**

Ahora probaremos **|– {true} S {true}** directamente, por inducción estructural, sin usar la hipótesis de completitud.

Prueba.

Base de la inducción:

S :: skip
Se cumple **|– {true} skip {true}** por el axioma SKIP.

S :: x := e
Se cumple **|– {true} x := e {true}** por el axioma ASI.

Paso inductivo:

S :: S_1 ; S_2
Por hipótesis inductiva: |– {true} S_1 {true} y |– {true} S_2 {true}.
Por SEC sobre lo anterior: **|– {true} S_1 ; S_2 {true}.**

Paso inductivo (continuación):

S :: if B then S_1 else S_2 fi

Por hipótesis inductiva: |– {true} S_1 {true} y |– {true} S_2 {true}.
Por MAT: true $\wedge$ B $\rightarrow$ true y true $\wedge$ $\neg$B $\rightarrow$ true.
Por CONS sobre lo anterior: |– {true $\wedge$ B} S_1 {true} y |– {true $\wedge$ $\neg$B} S_2 {true}.
Finalmente por COND sobre lo anterior: **|– {true} if B then S_1 else S_2 fi {true}.**

S :: while B do S_1 od

Por hipótesis inductiva: {true} S_1 {true}.
Por MAT: true $\wedge$ B $\rightarrow$ true.
Por CONS sobre lo anterior: {true $\wedge$ B} S_1 {true}.
Por REP sobre lo anterior: {true} while B do S_1 od {true $\wedge$ $\neg$B}.
Por MAT: true $\wedge$ $\neg$B $\rightarrow$ true.
Finalmente por CONS sobre lo anterior: **|– {true} while B do S_1 od {true}.**