Tema 5

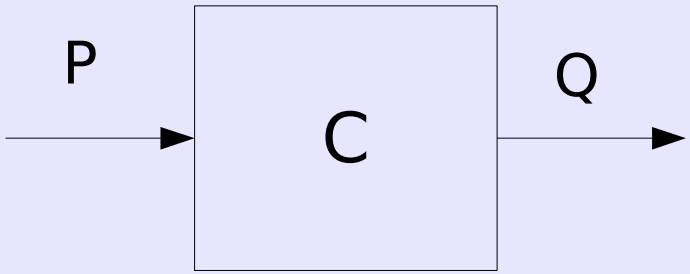
Verificación formal de algoritmos

Introducción

- Definición
  - La verificación formal de programas consiste en un conjunto de técnicas de comprobación formales que permiten demostrar si un programa funciona correctamente.
    - Programa funciona correctamente: cumple con unas especificaciones dadas
    - Técnicas de comprobación: hacen uso de procesos de inferencia. Cada tipo de sentencia ejecutable posee una regla de inferencia
    - Representación formal: Aserciones y Ternas de Hoare

Lógica de Hoare

- Permite probar la verdad o falsedad de ciertas propiedades de programas sin concurrencia o paralelismo
  - Corrección y terminación
- Basada en la idea de diagrama de flujo anotado



Lógica de Hoare

- Basada en la idea de diagrama de flujo anotado
  - C es una frase de lenguaje de alto nivel (una entrada, una salida)
    - Instrucción
    - Bloque de instrucciones consecutivas
    - Un programa o subprograma
  - Cada frase *C* denota una relación entre dos estados
    - Al ejecutar *C*, se modifica el valor de una o varias variables → tránsito de un estado a otro
  - P y Q son aserciones (anotaciones)
    - P → precondición (estado anterior a *C*)
    - Q → postcondición (estado posterior a *C*)

Lógica de Hoare

- Aserción
  - Fórmula lógica de predicados que toma un valor booleano en un contexto determinado.
  - Se encierra entre llaves
    - {x>2}
- Notación: Terna de Hoare
  - Expresión de la forma $\{P\}C\{Q\}$
    - P y Q → aserciones
    - C → sentencia o conjunto de sentencias de un programa
      - P → precondición de C
      - Q → postcondición de C

Introducción

- ***Ejemplo 1:*** Programa que calcula el inverso de un número
  - *{y≠0}x:=1/y {x=1/y}*
    - Precondición → valores de entrada diferentes a 0
    - Postcondición → inverso del valor de entrada
- Ejemplo 2
  - { } a:=b {a=b}
    - Siempre se obtienen estados que satisfacen la postcondición independientemente de la precondición
    - { } representa la aserción vacía
      - Verdadero en todos los estados

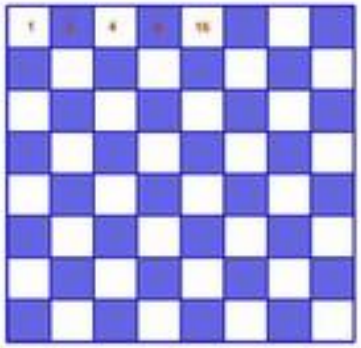
Introducción

- Conceptos preliminares
  - Código o parte de código
    - Sentencia o conjunto de sentencias
    - Programa entero
  - Sintaxis
    - No declaraciones de tipos
    - No sentencias de E/S
  - La concatenación de dos códigos C_1 y C_2 supone la ejecución secuencial de dichos códigos
    - $C_1; C_2$

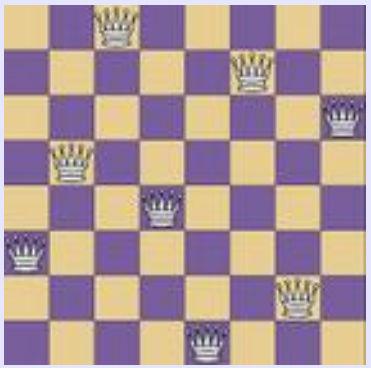
Introducción

- Conceptos preliminares
  - Un programa es una secuencia de sentencias que transforman el estado inicial en un estado final
  - El estado inicial es el estado anterior a la ejecución de un código
  - El estado final es el estado posterior a la ejecución de dicho código
  - El estado del sistema viene determinado por los valores de las variables en cada momento

Introducción



```
private static bool UbicaReinas(bool [,] tab, int reinas)
{
    if(reinas==0)
        return true;
    else
    {
        int n=tab.GetLength(0);
        int j= n-reinas;
        //Ver si se puede ubicar en la columna j
        for(int i=0; i<n; i++)
        {
            if(!Amenaza(tab, i, j))
            {
                tab[i,j]=true;
                //Intentar ubicar las otras reinas
                if(UbicaReinas(tab, reinas-1))
                    return true;
                //Fracasó el intento
                //Probar otra celda de esta columna
                else
                    tab[i,j]=false;
            }
        }
        return false;
        //No se pudo ubicar las 8 reinas
    }
}
```



Introducción

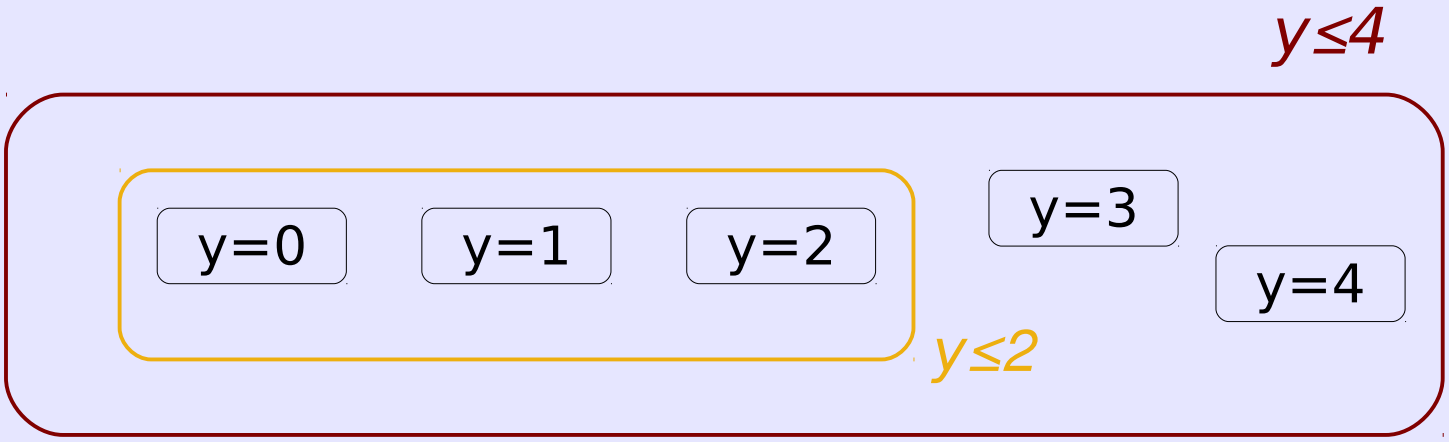
- Nomenclatura
  - El estado final depende del estado inicial
    - Precondición y postcondición no son independientes entre sí.
  - Forma de reflejarla
    - Mediante subíndices se indica si las variables representan valores iniciales o finales.
      - $\{a\omega=b\alpha\} \wedge \{b\omega=a\alpha\}$, donde "$\omega$" representa el estado final y "α" el estado inicial

Concepto de corrección

- Si $\{P\}C\{Q\}$ es un código con la precondición $\{P\}$ y la postcondición $\{Q\}$, entonces $\{P\}C\{Q\}$ es correcto si cada estado inicial posible que satisfaga $\{P\}$ da como resultado un estado final que satisface $\{Q\}$
- Distinción entre corrección parcial y total
  - **Parcial** → se dice que $\{P\}C\{Q\}$ es parcialmente correcto si el estado final de C, cuando termina el programa (aunque no se exige), satisface $\{Q\}$ siempre que el estado inicial satisface $\{P\}$
  - **Total** → cuando un código además de ser correcto termina
    - Sólo es necesario en los códigos con bucles y/o recursividad

Fortaleza y debilidad de aserciones

- Una aserción R es más fuerte que una aserción S, si
  - Todos los estados que satisfacen R también satisfacen S pero no viceversa
    - $R \Rightarrow S$
  - R es más **fuerte** que S y S es más **débil** que R
    - *y≤ 2 es más fuerte que y≤4*
    - *{y≤ 2} ⇒ {y≤4}*

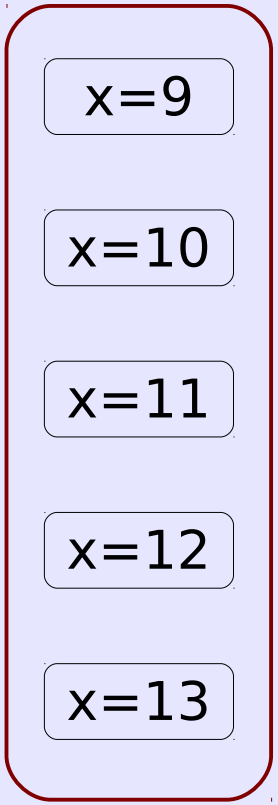


Fortaleza y debilidad de aserciones

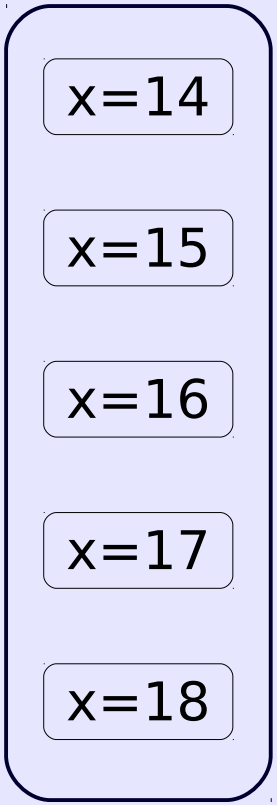
- Fortalecer una aserción
  - Disminuir el nº de estados que la pueden satisfacer
    - Más selectiva, más específica
- Debilitar una aserción
  - Aumentar el nº de estados que la pueden satisfacer
    - Menos selectiva, menos específica

Fortaleza y debilidad de aserciones

- Ejemplo
  - {x ≥ 9 } x:=x+5 {x ≥ 14}



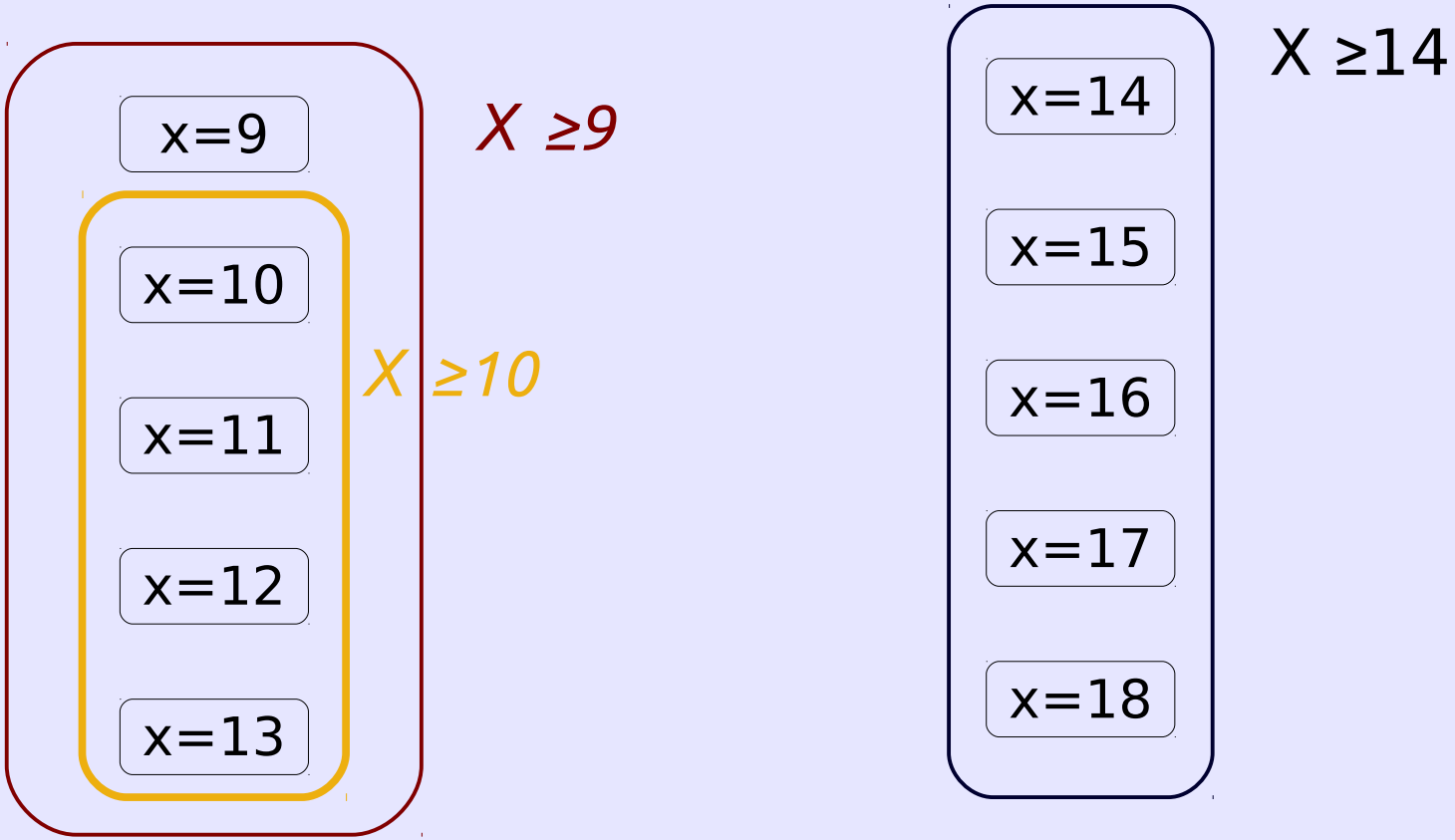
X ≥9



X ≥14

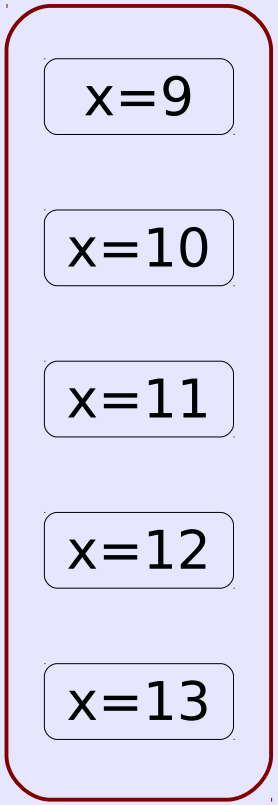
Fortaleza y debilidad de aserciones

- Ejemplo (Fortalecer la precondición)
  - {x ≥ 10 } x:=x+5 {x ≥ 14}

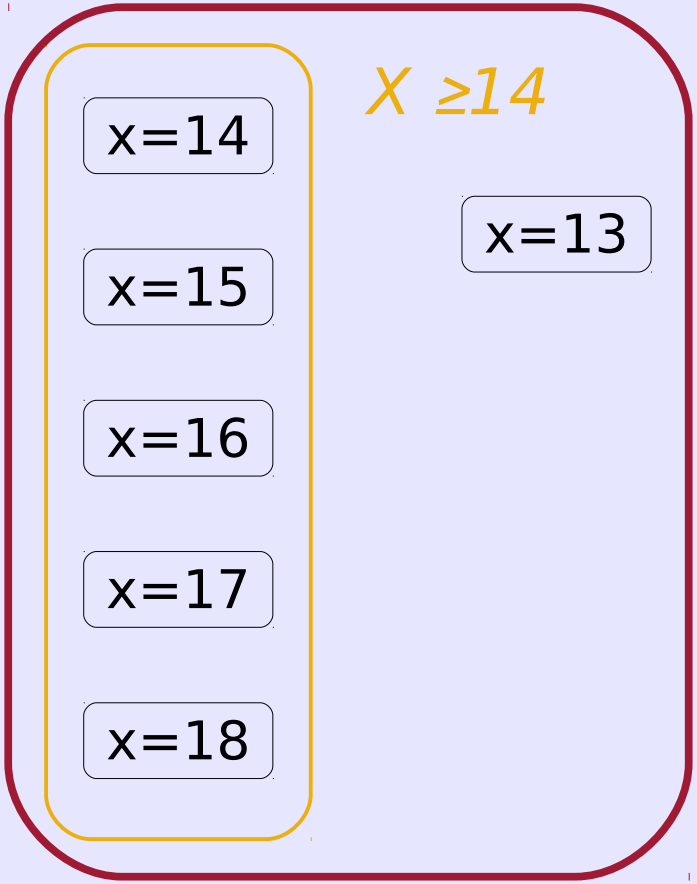


Fortaleza y debilidad de aserciones

- Ejemplo (Debilitar la postcondición)
  - {x ≥ 9 } x:=x+5 {x ≥ 13}



X ≥9



Fortaleza y debilidad de aserciones

- La aserción más débil es {}
  - Recoge todos los estados posibles
  - Constante lógica TRUE
- La aserción más fuerte será por tanto FALSE
  - Ningún estado satisface dicha condición

Verificación de códigos sin bucles

- Verificación
  - Para demostrar la corrección de las partes de un programa se parte de la postcondición final del código, es decir, de las condiciones que deben satisfacer los resultados. A partir de ahí se deduce la precondición inicial
    - Si coinciden → Verficado
  - El código se verifica en sentido contrario a como se ejecuta

Reglas de inferencia

- Verificación
  - Partiendo de una especificación ($\{P\}C\{Q\}$)
    - Aplicar una serie de reglas de inferencia
      - Fortalecimiento de la precondición
      - Debilitamiento de la postcondición
      - Conjunción
      - Disyunción
      - Asignación
      - Composición
      - Condicional
      - Bucle parcial
    - Demostrar el programa cumple esa especificación

Reglas de inferencia

- Fortalecimiento de las precondición
  - Si una parte de un código es correcta bajo la precondición $\{P\}$, seguirá siendo correcta si se refuerza $\{P\}$
  - Si $\{P\}C\{Q\}$ es correcto y $P_1 \Rightarrow P$
    - Podemos afirmar que $\{P_1\}C\{Q\}$ es correcto
    - Permite asumir una precondición más fuerte
  - Regla de inferencia

$$\frac{P_1 \Rightarrow P,\ \{P\}\ C\ \{Q\}}{\{P_1\}\ C\ \{Q\}}$$

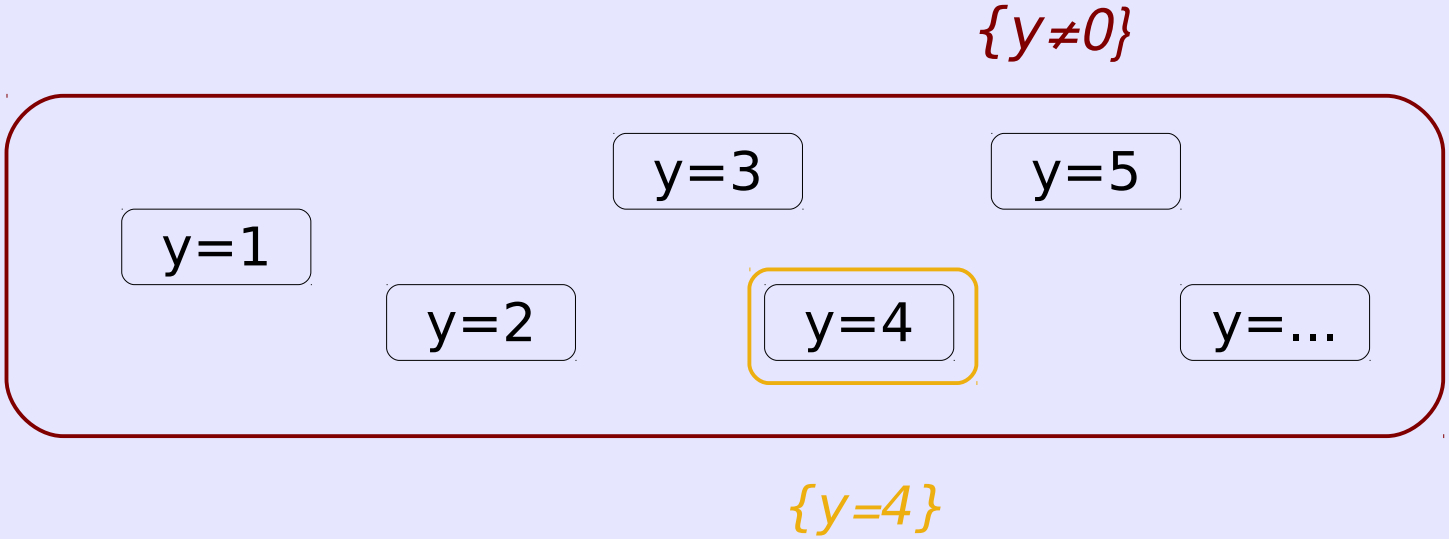
Reglas de inferencia

- Fortalecimiento de las precondición
  - Ejemplo
    - {y ≠ 0} x:=1/y {x=1/y} → Correcto
    - {y = 4} x:=1/y { x=1/y} → ¿Es correcto?

$$\frac{y=4 \Rightarrow y\neq 0 \qquad \{y \neq 0\}\ x:=1/y\ \{x=1/y\}}{\{y = 4\}\ x:=1/y\ \{\ x=1/y\}}$$

Reglas de inferencia

- ¿y=4 ⇒ y≠0?



Reglas de inferencia

- Fortalecimiento de la precondición vacía
  - La aserción vacía {} puede ser reforzada para producir cualquier precondición $\{P\}$
    - { } a:=b {a=b}
  - Como regla general
    - Será ventajoso intentar formular la precondición más débil posible que asegure que se cumple una cierta postcondición dada.
      - Cualquier precondición que implique la anterior (más fuerte) será satisfecha automáticamente
      - Los programas deben ser escritos de modo que resulten lo más versátiles posibles

Reglas de inferencia

- Debilitamiento de postcondición
  - Si $\{P\}C\{Q\}$ es correcto y $Q \Rightarrow Q_1$
    - Podemos afirmar que $\{P\}C\{Q_1\}$ es correcto
    - Permite asumir una postcondición más débil
  - Regla de inferencia

$$\frac{\{P\}\ C\ \{Q\},\ Q \Rightarrow Q_1}{\{P\}\ C\ \{Q_1\}}$$

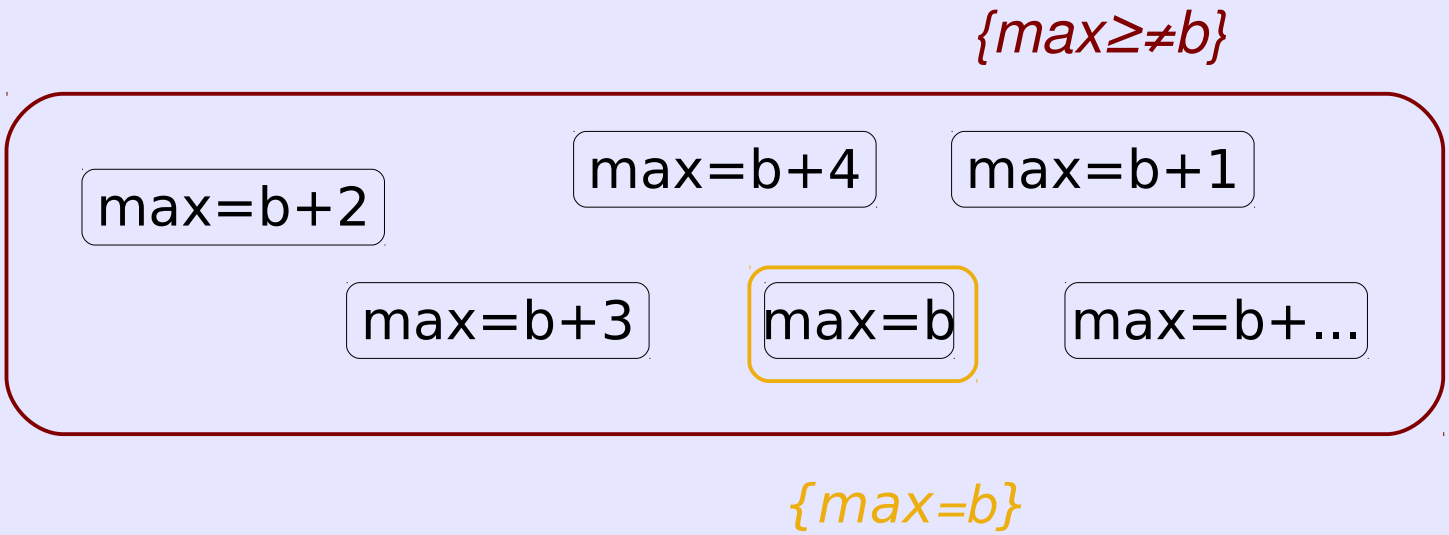
Reglas de inferencia

- Debilitamiento de las postcondición
  - Ejemplo
    - { } max:=b {max=b} → Correcto
    - { } max:=b {max≥b} → ¿Es correcto?

$$\frac{\{\}\ max:=b\ \{max=b\} \qquad max=b \Rightarrow max \geq b}{\{\}\ max:=b\ \{max \geq b\}}$$

Reglas de inferencia

- ¿max=b ⇒ max≥b?



Reglas de inferencia

- Regla de la conjunción
  - Permite reforzar la precondición y la postcondición de forma simultánea
  - Si C es una parte de código y se ha establecido que $\{P_1\}C\{Q_1\}$ y $\{P_2\}C\{Q_2\}$
    - Podemos afirmar que $\{P1 \wedge P2\}\ C\ \{Q_1 \wedge Q_2\}$

$$\frac{\{P_1\}\ C\ \{Q_1\},\ \{P_2\}\ C\ \{Q_2\}}{\{P_1 \wedge P_2\}\ C\{Q_1 \wedge Q_2\}}$$

Reglas de inferencia

- Regla de la conjunción
  - Ejemplo
    - $\{\ \}\ i:=i+1\ \{i_\omega = i_\alpha + 1\} \rightarrow$ Correcto
    - $\{i_\alpha > 0\}\ i:=i+1\ \{i_\omega > 0\} \rightarrow$ Correcto
    - $\{i_\alpha > 0\} i:=i+1\ \{i_\omega > 1\} \rightarrow$ **¿Es correcto?**
    - Aplicando la regla de la conjunción tenemos que
      - $\{i_\alpha > 0\} i:=i+1\ \ \{(i_\omega = i_\alpha + 1) \wedge (i_\omega > 1)\}$ es correto

$$\frac{\{\ \}\ i:=i+1\ \{i_\omega = i_\alpha + 1\} \qquad \{i_\alpha > 0\}\ i:=i+1\ \{i_\omega > 1\}}{\{i_\alpha > 0\} i:=i+1\ \ \{(i_\omega = i_\alpha + 1) \wedge (i_\omega > 1)\}}$$

Reglas de inferencia

- Regla de la conjunción
  - Ejemplo
    - ¿Cómo demostramos que $\{i_\alpha>0\}i:=i+1\ \{i_\omega>1\}$ es correcto?
      - Aplicando la regla del debilitamiento de la postcondición
      - Si $\{i_\omega>1\}$ es más débil que $\{(i_\omega=i_\alpha+1)\wedge(i_\alpha>1)\}$

$$\frac{\{i_\alpha>0\}i:=i+1\ \{(i_\omega=i_\alpha+1)\wedge(i_\omega>1)\} \quad \{(i_\omega=i_\alpha+1)\wedge(i_\omega>1)\}\Rightarrow\{i_\omega>1\}}{\{i_\alpha>0\}i:=i+1\ \{i_\omega>1\}}$$

    - ¿Es $\{i_\omega > 1\}$ más débil que $\{(i_\omega=i_\alpha+1)\wedge(i_\omega>1)\}$
      - Sí, porque hemos debilitado $\{(i_\omega=i_\alpha+1)\wedge(i_\alpha>1)\}$ quitando uno de los elementos de la conjunción

Reglas de inferencia

- Regla de la disyunción
  - Permite debilitar la precondición y la postcondición de forma simultánea
  - Si C es una parte de código y se ha establecido que $\{P_1\}C\{Q_1\}$ y $\{P_2\}C\{Q_2\}$
    - Podemos afirmar que $\{P1 \vee P2\}\ C\ \{Q_1 \vee Q_2\}$

$$\frac{\{P_1\}\ C\ \{Q_1\},\ \{P_2\}\ C\ \{Q_2\}}{\{P_1 \vee P_2\}\ C\{Q_1 \vee Q_2\}}$$

Reglas de inferencia

- Disyunción y disyunción inversa

$$\frac{\{P_1\}\ C\ \{Q\},\ \{P_2\}\ C\ \{Q\}}{\{P_1 \vee P_2\}\ C\ \{Q\}}$$

$$\frac{\{P_1 \vee P_2\}\ C\ \{Q\}}{\{P_i\}\ C\ \{Q\}}$$

- Conjunción y conjunción inversa

$$\frac{\{P\}\ C\ \{Q_1\},\ \{P\}\ C\ \{Q_2\}}{\{P\}\ C\ \{Q_1 \wedge Q_2\}}$$

$$\frac{\{P\}\ C\ \{Q_1 \wedge Q_2\}}{\{P\}\ C\ \{Q_i\}}$$

Reglas de inferencia

- Asignación
  - Sentencias de la forma $V{:=}E$
    - V → una variable
    - E → es una expresión
  - Regla de la asignación
    - Si C es una sentencia de la forma `x:=`E con la postcondición Q, entonces la precondición de C puede hallarse sustituyendo todos los casos de `x`, en Q, por `E` (Q_E^x)

```
{?} C {Q}
C →  x:=E
----------
{Q_E^x} C{Q}
```

Reglas de inferencia

- Concatenación
  - C1;C2
    - Las partes del código se ejecutan secuencialmente de tal forma que el estado final de la primera parte de un código se convierte en el estado inicial de la segunda parte del código.
  - Regla de la concatenación

```
{P} C1 {R}
{R} C2 {Q}
----------
{P}C1;C2{Q}
```

Reglas de inferencia

- Sentencia condicional simple
  - Es correcta si satisface las condiciones de verificación
    - $P \wedge \neg B \Rightarrow Q$
    - $\{P \wedge B\}\ S\ \{Q\}$

```
{P}
si B entonces
    S
fin_si
{Q}
```

$$\frac{\{P \wedge \neg B\} \Rightarrow \{Q\},\ \{P \wedge B\} S \{Q\}}{\{P\}\ \textbf{si}\ B\ \textbf{entonces}\ S\ \textbf{fin\_si}\ \{Q\}}$$

Reglas de inferencia

- Sentencia condicional doble
  - Es correcta si todas las condiciones de verificación generadas por:
    - $\{P \wedge B\}\ S1\ \{Q\}$
    - $\{P \wedge \neg B\}\ S2\ \{Q\}$

    son correctas

```
{P}
si B entonces
    S1
si_no
    S2
fin_si
{Q}
```

$$\frac{\{P \wedge B\}S1\{Q\},\{P \wedge \neg B\}S2\{Q\}}{\{P\}\ \textbf{si}\ B\ \textbf{entonces}\ S1\ \textbf{si\_no}\ S2\ \textbf{fin\_si}\ \{Q\}}$$

Reglas de inferencia

- Sentencia selección múltiple
  - Es correcta cuando
    - La alternativa está bien construida
      - $P \Rightarrow B_1 \vee B_2 \vee \ldots \vee B_n$
      - $\forall\, i,j: 1 \le i,j \le n \wedge i \neq j : P \Rightarrow \neg(B_i \wedge B_j)$
    - satisface su especificación
      - $\forall\, i,j: 1 \le i,j \le n : \{P \wedge B_i\} S_i \{Q\}$

```
{P}
según_sea B hacer
    B1:S1
    B2:S2
    …
    Bn:Sn
fin_según
{Q}
```

$$\frac{P \Rightarrow B_1 \vee B_2 \vee \ldots \vee B_n \;,\; \forall i,j: 1 \le i,j \le n \wedge i \neq j : P \Rightarrow \neg(B_i \wedge B_j) \;,\; \forall i,j: 1 \le i,j \le n : \{P \wedge B_i\} S_i \{Q\}}{\{P\}\ \textbf{según\_sea}\ B\ \textbf{hacer}\ B_1:S_1;B_2:S_2;\ldots;B_n:S_n\ \textbf{fin\_según}\ \{Q\}}$$

Corrección total

- Sentencia iterativa mientras
  - Es correcta si satisface las condiciones de verificación generadas por:
    - $P \Rightarrow I$
    - $I \wedge \neg B \Rightarrow Q$
    - $\{I \wedge B\}\ S\ \{I\}$
    - $I \wedge B \Rightarrow t > 0$
    - $\{I \wedge B \wedge t = T\}\ S\ \{t < T\}$

```
{P}
mientras B hacer {I}
    S
fin_mientras
{Q}
```

$$\frac{P \Rightarrow I,\ I \wedge \neg B \Rightarrow Q,\ \{I \wedge B\}\,S\,\{I\},\ I \wedge \neg B \Rightarrow t > 0,\ \{I \wedge B \wedge t = T\}\,S\,\{t < T\}}{\{P\}\ \textbf{mientras}\ B\ \textbf{hacer}\ S\ \textbf{fin\_mientras}\ \{Q\}}$$

Corrección total

- Sentencia iterativa mientras
  - Invariante (I)
    - Conjunto de condiciones que deben de cumplirse:
      - Antes de la primera iteración
      - Al comienzo de cada iteración del bucle
      - Al salir del bucle
  - Corrección parcial
    - Los tres primeros puntos

Corrección total

- Sentencia iterativa mientras
  - Corrección total
    - Los cinco puntos
    - Demostrar que el bucle termina tras un número finito de repeticiones
  - Función de cota o limitadora (t)
    - Cota superior del nº de iteraciones que quedan por hacer
    - Función monótona acotada que varía con cada iteración
    - Depende de las variables que intervienen en el bucle
      - Aquellas que aparecen en la condición B y toman valores enteros

Algoritmos iterativos

- Especificación de un subalgoritmo
  - Cabecera
    - Nombre de la función, parámetros (tipo y nombre), variables locales que contendrán los resultados
  - Precondición
    - Aserto con las condiciones que deben cumplir los parámetros que reciben los datos antes de ejecutarse la primera instrucción del cuerpo de la función
  - Postcondición
    - Aserto con las condiciones que deben cumplir los resultados

Algoritmos iterativos

- Especificación de un subalgoritmo
  - Calcula el factorial de un número

```
entero función factorial (E entero:n)
   {n>0}
     S
   {res=n!}
   Devolver res
fin_función
```

  - Verificación
    - Demostrar la especificación del cuerpo del subalgoritmo