Tema 4. Condiciones para el paralelismo: Dependencias

Organización de Computadores

LUIS ENRIQUE MORENO LORENTE
RAÚL PÉRULA MARTÍNEZ
ALBERTO BRUNETE GONZALEZ
DOMINGO MIGUEL GUINEA GARCIA ALEGRE
CESAR AUGUSTO ARISMENDI GUTIERREZ
JOSÉ CARLOS CASTILLO MONTOYA

Departamento de Ingeniería de Sistemas y Automática

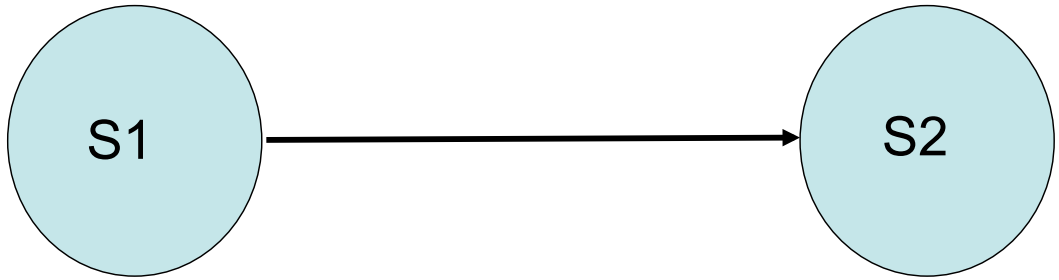
DEPENDENCIAS

- Para poder ejecutar diferentes segmentos de código en paralelo, tenemos que asegurarnos de que son independientes, es decir, el orden de ejecución no importa (no afecta al resultado).
- En general nos referimos a segmentos de código (grupos de instrucciones).
- Por simplicidad, consideraremos instrucciones simples.
- Las dependencias entre (grupos de) instrucciones pueden describirse por medio del grafo de dependencias.
- El análisis de dependencias es estático, es decir, no se realiza en run-time.

GRAFO DE DEPENDENCIAS

- Consideremos dos secuencias de instrucciones S1 y S2, para las que existe un camino de ejecución entre ellas (S1 primero, y después S2).
- Nodos: (grupo de) instrucciones que se van a ejecutar en paralelo.
- Arcos: indican dependencia.

- Ejemplo:



S1	Ld r2, A
	Add r1, r2, r1
	Subi r2, r2, #1

S2	Sub r3, r2, r1 Sto B, r3

DEPENDENCIAS

Existen tres tipos de dependencias:

- Dependencias de datos
- Dependencias de control
- Dependencias de recursos

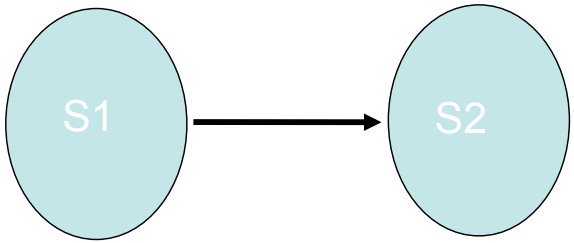
DEPENDENCIAS DE DATOS

- Puede plantearse un conflicto cuando dos instrucciones hacen uso del mismo dato: el orden de ejecución es importante.

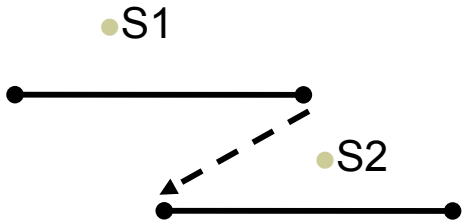
1. Dependencias de flujo
2. Antidependencias
3. Dependencias de salida
4. Dependencias de E/S (a fichero)
5. Desconocidas

DEPENDENCIAS DE DATOS

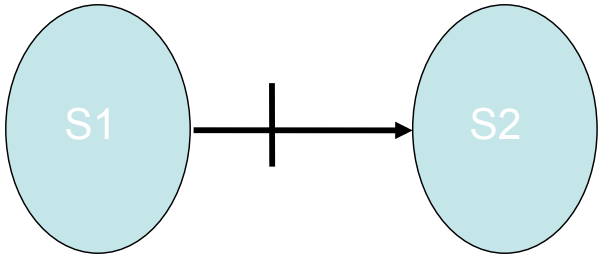
1.Dependencia de flujo: la salida de S1 alimenta la entrada de S2.



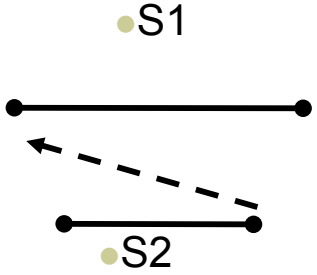

```
A = 1
...
B = A+C
```



2. Antidependencia: la salida de S2 se solapa con la entrada de S1.

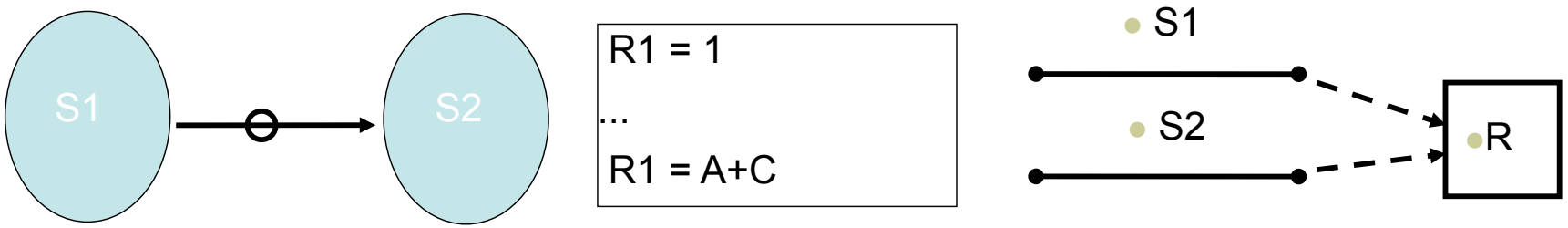



```
B = A
...
A = C+D
```

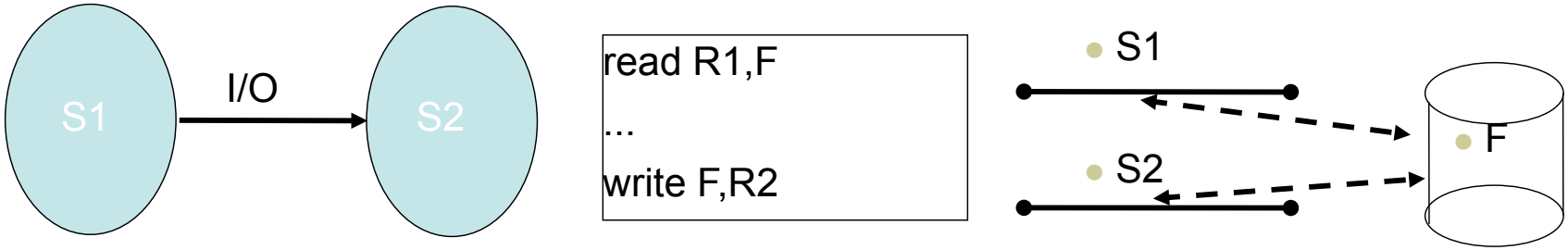


DEPENDENCIAS DE DATOS

3. Dependencia de salida: S1 y S2 escriben en la misma salida.



4. Dependencia de I/O: S1 y S2 acceden al mismo fichero.



DEPENDENCIAS DE DATOS

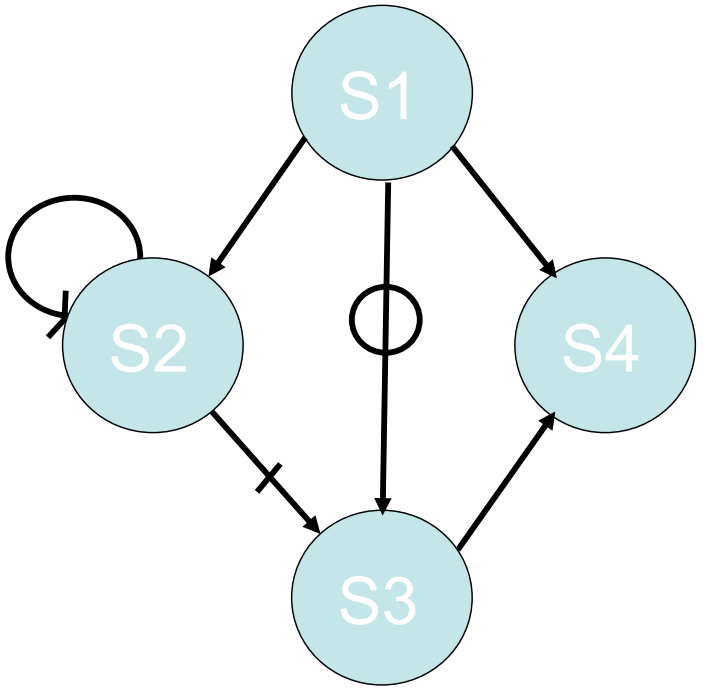
5. Dependencias desconocidas: ciertas dependencias no pueden ser determinadas.
   - Direccionamientos indirectos –> no es posible saber a que posición se esta accediendo.
   - Variables índices en bucles:
     - Las variable aparecen más de una vez en un bucle con diferentes índices.
     - índices no lineales.
     - índices que no contienen la variable del bucle.

DEPENDENCIAS DE DATOS

- Los casos (4) y (5) presuponen el peor caso posible: en el que no es posible determinar que esta ocurriendo, por lo que se asume que existe una dependencia para evitar posibles errores.

Ejemplo:

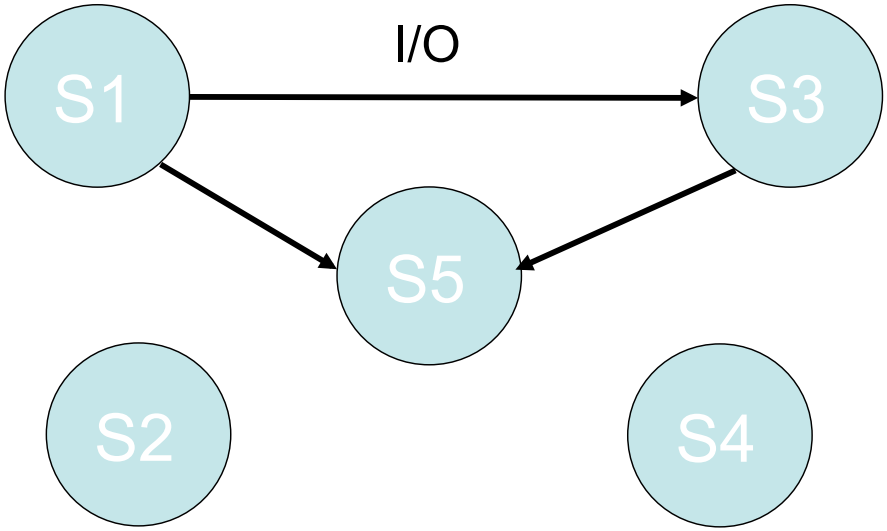
```
S1:  load R1,A  /* R1 <- mem(A) */
S2:  add R2,R1/* R2 <- R2+R1 */
S3:  move R1,R3       /* R1 <- R3 */
S4:  store B,R1/* mem(B) <- R1 */
```



DEPENDENCIAS DE DATOS

Ejemplo 2:

```
S1:  read 4,A    /* A <- tape 4    */
S2:  rewind 4    /* rewind tape 4 */
S3:  write 4, B   /*  tape 4 <- B    */
S4:  rewind 4    /* rewind tape 4 */
S5:  add R1,A,B /* R1 <- A + B   */
```



DEPENDENCIAS DE CONTROL

- Tenemos dependencias de control cuando el orden de ejecución no puede ser determinado estáticamente, sino sólo en tiempo de ejecución.
  - Ejemplo: sentencias tipo “IF” o instrucciones tipo branch/jump.
  - También aparecen en bucles (ver ejemplo).
  - Dado que el orden no es conocido anticipadamente, las dependencias reales no pueden ser determinadas.

DEPENDENCIAS DE CONTROL

Ejemplo:

```
for (i=0;i<n;i++) {
    a[i] = c[i];
    if (a[i] < 0)
        a[i] = 1;
}
```

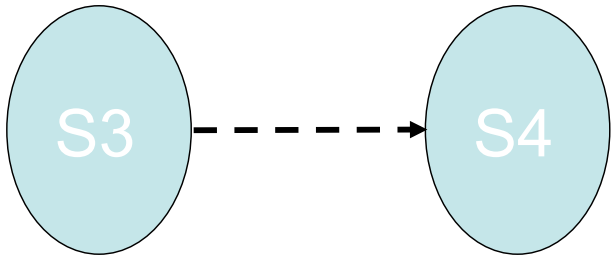
```
for (i=1;i<n;i++) {
    if (a[i -1] < 0)
        a[i] = 1;
}
```

- En ambos casos no sabemos cual será la próxima instrucción a ser ejecutada después del test.
- Pero, en el primer caso no hay dependencia de control, y en el segundo caso sí la hay. ¿Por qué?

DEPENDENCIAS DE RECURSOS

- Tenemos dependencias de recursos cuando recursos compartidos del procesador (ej. ALU, UF coma flotante, buses, etc.) son demandados por instrucciones diferentes al mismo tiempo.

- Ejemplo:
  - Dependencias por ALU
  - Dependencias de almacenamiento



Ejemplo:

S1: load R1, A
S2: load R2, B
S3: add R3,R1,R2
S4: sub R4,R1,R2

CONDICIONES PARA EL PARALELISMO

- La transformación de código secuencial en paralelo puede realizarse:
  - Manualmente por el programador (paralelismo explícito).
  - Automáticamente por el compilador (paralelismo implícito).
- En ambos casos, el "particionamiento" del código (decomposition) es un paso clave.
- El particionamiento o división del código determina sí (y cómo) un segmento de código puede ser ejecutado en paralelo o en un orden en particular.
- La detección de segmentos paralelos requiere la verificación de las relaciones de dependencia.

CONDICIONES DE BERNSTEIN

Conjunto de condiciones formales que determinan si dos procesos (segmentos de programa) pueden ser ejecutados en paralelo.

I_i : Conjunto de variables de entrada al proceso P_i (read set).

O_i : Conjunto de variables de salida del proceso P_i (write set).

"Variables": operandos a ser loaded/stored en registros o posiciones de memoria.

CONDICIONES DE BERNSTEIN

- Consideremos los procesos P1 y P2 y sus conjuntos de entrada (I1, I2) y salida (O1, O2)

- Condiciones de Bernstein (dependencias de datos)

1. $I_1 \cap O_2 = 0$ (antidependencia)
2. $I_2 \cap O_1 = 0$ (flujo)
3. $O_1 \cap O_2 = 0$ (salida)

- Si se verifican las tres condiciones, P1 y P2 pueden ejecutarse en paralelo, esto lo denotamos como :

P1 || P2

CONDICIONES DE BERNSTEIN

- En resumen, si P1 || P2 el orden de ejecución de dos procesos es irrelevante, en tanto que:
  - La salida de uno no es usada como entrada por el otro
  - No escriben sobre las mismas variables de salida

- Propiedades de la relación de paralelismo || :
  - Conmutativa: Pi || Pj <=> Pj || Pi
  - NO transitiva: Pi || Pj and Pj || Pk =/> Pi || Pk
  - Asociativa: (Pi || Pj) || Pk => Pi || (Pj || Pk)

- Por tanto, no es una relación de equivalencia.

CONDICIONES DE BERNSTEIN: Ejemplo

- Detectar que instrucciones del siguiente código pueden ejecutarse concurrentemente
  - Instrucción 1 **x:=y+1**
  - Instrucción 2 **y:=x-2**
  - Instrucción 3 **z=a+b-7**

CONDICIONES DE BERNSTEIN: Ejemplo

- Detectar que instrucciones del siguiente código pueden ejecutarse concurrentemente

- Lo primero es establecer qué variables se acceden en modo lectura. En las instrucciones vemos que son:
  - Para la instrucción 1 es la variable "y".
  - Para la instrucción 2 es la variable "x".
  - Para la instrucción 3 son las variables "a", "b".
- También necesitaremos saber qué variables se acceden en modo escritura. Son las siguientes:
  - Para la instrucción 1 es la variable "x".
  - Para la instrucción 2 es la variable "y".
  - Para la instrucción 3 es la variable "z".

I1 **x:=y+1**
I2 **y:=x-2**
I3 **z=a+b-7**

CONDICIONES DE BERNSTEIN: Ejemplo

- Ahora aplicamos las tres condiciones:
- **Condición 1: La intersección entre las variables de lectura de un conjunto de instrucciones C1 y las variables de escritura de un conjunto C2 debe ser vacía.**
- Instrucciones 1 y 2 → No cumple la condición

Lectura 1	Escritura 2
y	y

- Instrucciones 1 y 3 → Cumple la condición

Lectura 1	Escritura 3
y	z

- Instrucciones 2 y 3 → Cumple la condiciónte

Lectura 2	Escritura 3
x	z

I1 **x:=y+1**
I2 **y:=x-2**
I3 **z=a+b-7**

CONDICIONES DE BERNSTEIN: Ejemplo

- **Condición 2: La intersección entre las variables de escritura de un conjunto de instrucciones C1 y las variables de lectura de un conjunto C2 debe ser vacía.**
- Instrucciones 1 y 2 → No cumple la condición

Escritura 1	Lectura 2
x	x

- Instrucciones 1 y 3 → Cumple la condición

Escritura 1	Lectura 3
x	a,b

- Instrucciones 2 y 3 → Cumple la condición

Escritura 2	Lectura 3
y	a,b

I1 **x:=y+1**
I2 **y:=x-2**
I3 **z=a+b-7**

CONDICIONES DE BERNSTEIN: Ejemplo

- **Condición 3: La intersección entre las variables de escritura de un conjunto de instrucciones C1 y las variables de escritura de un conjunto C2 debe ser vacía.**
- Instrucciones 1 y 2 → Cumple la condición

Escritura 1	Escritura 2
x	y

- Instrucciones 1 y 3 → Cumple la condición

Escritura 1	Escritura 3
x	z

- Instrucciones 2 y 3 → Cumple la condición

Escritura 2	Escritura 3
y	z

I1 **x:=y+1**
I2 **y:=x-2**
I3 **z=a+b-7**

CONDICIONES DE BERNSTEIN: Ejemplo

- **Ahora podemos saber qué instrucciones pueden ejecutarse concurrentemente**

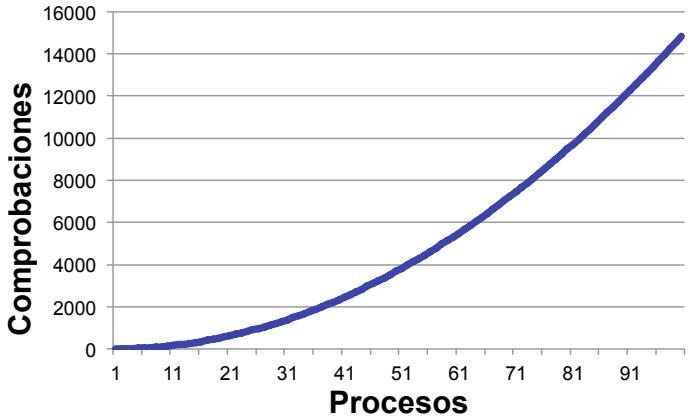
Par de instrucciones	Condición 1	Condición 2	Condición 3	¿Pueden ejecutarse concurrentemente?
1 y 2	No	No	Si	No
1 y 3	Si	Si	Si	Si
2 y 3	Si	Si	Si	Si

I1 **x:=y+1**
I2 **y:=x-2**
I3 **z=a+b-7**

CONDICIONES DE BERNSTEIN

- Observar que la condición $I_i \cap I_j$!= 0 no impide el paralelismo.
- Si tenemos n procesos, la violación de alguna de las condiciones impide total o parcialmente la ejecución en paralelo de P1…Pn

$$3*n*(n-1)/2$$



- Qué ocurre con las otras dependencias?
  - Cualquier instrucción que dependa de condiciones de tiempo de ejecución no puede ponerse en forma paralela (subscripts, tests, Branches cond. …).
  - Los ficheros pueden tratarse e introducirse en los conjuntos de entrada I y salida O.

CONDICIONES DE BERNSTEIN

- Ejemplo:
  - Detectar el paralelismo del siguiente código.
  - Suponiendo que cada instrucción requiere 1 paso (no pipeline).
  - Suponiendo que tenemos dos sumadores disponibles (pipeline).

P1: C = D*E

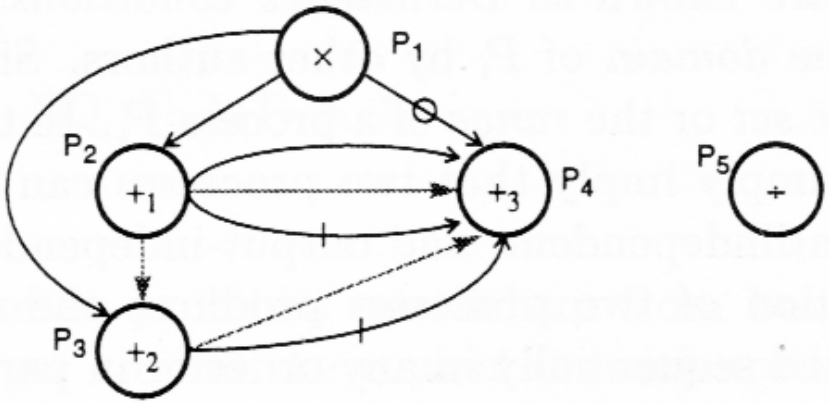
P2: M = G+C

P3: A = B+C

P4: C = L+M

P5: F = G/E

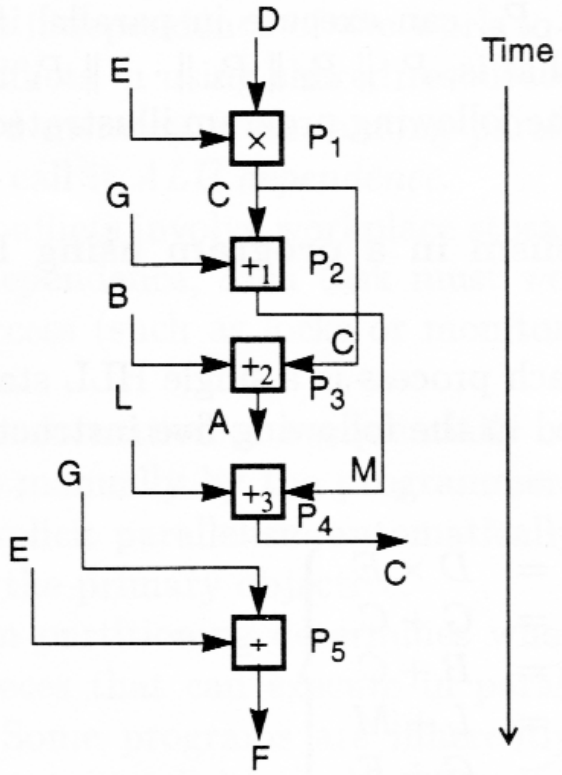
CONDICIONES DE BERNSTEIN



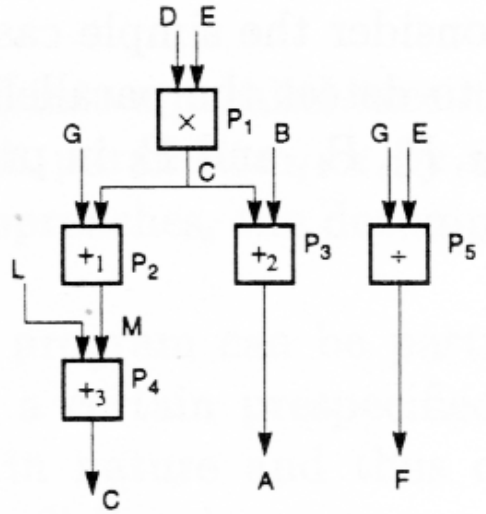
(a) A dependence graph showing both data dependence (solid arrows) and resource dependence (dashed arrows)

Imagen de Kwang, 1993

CONDICIONES DE BERNSTEIN



(b) Sequential execution in five steps, assuming one step per statement (no pipelining)



(c) Parallel execution in three steps, assuming two adders are available per step

Imagenes de Kwang, 1993

CONDICIONES DE BERNSTEIN

Generalización:

- Las condiciones de Bernstein a nivel de instrucción pueden extenderse a niveles superiores, como segmentos de código, subrutinas, procesos, programas.
- Las dependencias a niveles altos se infieren de las dependencias a niveles inferiores.

REFERENCIAS

[Hwan, 1993] K. HWANG. Advanced Computer Architecture. Mc Graw-Hill, 1993.