Generación de Código

CI4722 – Lenguajes de Programación III

Ernesto Hernández-Novich
<emhn@usb.ve>

Universidad "Simón Bolívar"

UNIVERSIDAD SIMÓN BOLÍVAR

Asignación y Reserva de Registros

- El Generador estudiado aprovecha registros un bloque por vez – el código no es tan inocente pero puede ser mediocre.
  - Las variables vivas siempre son almacenadas al terminar.
  - Es posible un escenario en que un registro se use poco o nada, mientras el resto es usado con más frecuencia.

UNIVERSIDAD SIMÓN BOLÍVAR

Asignación y Reserva de Registros

- El Generador estudiado aprovecha registros un bloque por vez – el código no es tan inocente pero puede ser mediocre.
  - Las variables vivas siempre son almacenadas al terminar.
  - Es posible un escenario en que un registro se use poco o nada, mientras el resto es usado con más frecuencia.
- Podemos ahorrar los almacenamientos y subsiguientes cargas:
  - Manteniendo en registros variables de uso frecuente.
  - Asegurando el uso consistente *global* de esos registros.
- Ciclos son los candidatos naturales.

UNIVERSIDAD SIMÓN BOLÍVAR

Asignación y Reserva de Registros

- El Generador estudiado aprovecha registros un bloque por vez – el código no es tan inocente pero puede ser mediocre.
  - Las variables vivas siempre son almacenadas al terminar.
  - Es posible un escenario en que un registro se use poco o nada, mientras el resto es usado con más frecuencia.
- Podemos ahorrar los almacenamientos y subsiguientes cargas:
  - Manteniendo en registros variables de uso frecuente.
  - Asegurando el uso consistente *global* de esos registros.
- Ciclos son los candidatos naturales.

¿Cómo *mantener* valores en los registros el máximo tiempo posible?

UNIVERSIDAD SIMÓN BOLÍVAR

Asignación Global de Registros

- Continuamos con las mismas suposiciones:
  - Conocemos la estructura de ciclos de un Grafo de Flujo.
  - Sabemos cuáles valores de un Bloque Básico son usados fuera de él.
- Podemos definir un conjunto de registros que contengan los valores "más activos" por ciclo – para algún valor de "más activo".
  - Es fácil de implantar – se usa desde la década de 1960.
  - Puede haber conjuntos diferentes para ciclos diferentes.
  - No es fácil "pegarla" con el tamaño del conjunto para *todos* los programas compilados – pero sigue siendo una buena aproximación.
  - Puede ofrecerse una construcción en el lenguaje para que el programador influya sobre el compilador – `register` en C.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo medir el ahorro?

Criterio para determinar "más activo"

- Supongamos que mantener `x` en un registro ahorra una unidad de costo por cada referencia a `x` en un ciclo.
- El generador vía *getReg* intenta mantener `x` en un registro hasta el final del bloque – el ahorro solamente aplica para usos en bloques donde **no** se asigna `x`.
- Evitar un `ST` nos ahorra dos unidades – ocurre si `x` está viva y en un registro al salir de un bloque.

UNIVERSIDAD SIMÓN BOLÍVAR

No todo es ahorro

Criterio para determinar "más activo"

- La primera vez que se ingresa al ciclo tenemos que cargar `x` en el registro – ocurre *una* vez por ciclo.
- En cada bloque de salida del ciclo, si `x` continúa viva es necesario almacenarla – ocurre *una* vez por ciclo.

El cuerpo de un ciclo se repite muchas veces – estos costos son despreciables a la larga.

UNIVERSIDAD SIMÓN BOLÍVAR

Beneficio de mantener x en un registro

Una aproximación razonable

$$Benefit(x) = \sum_{\text{bloques B en ciclo L}} use(x, B) + 2 \times live(x, B)$$

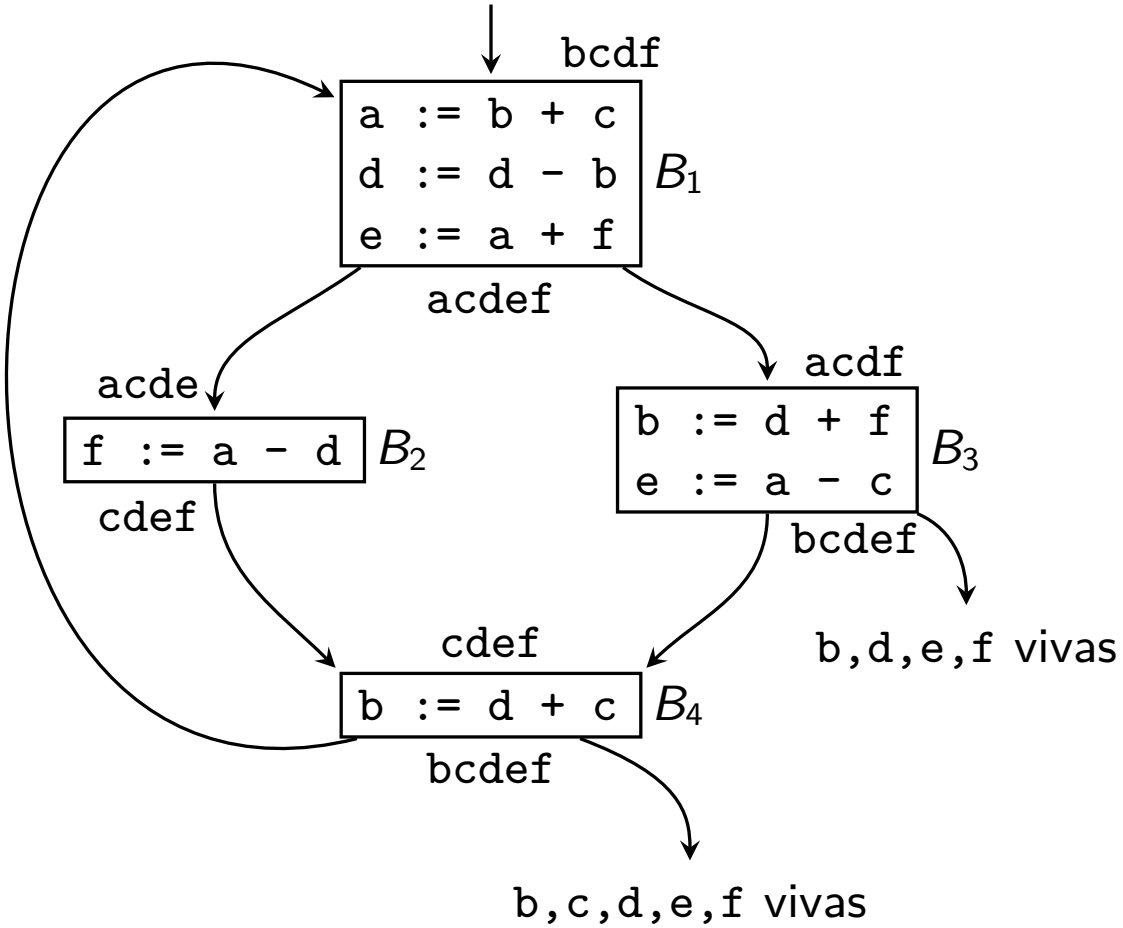
- $use(x, B)$ – veces que se usa x en B *antes* de alguna asignación a x.
- $live(x, B)$ – caracteriza que x ha sido asignada y viva al salir de B.
- Aproximación suficientemente buena porque
  - No todos los bloques de un ciclos se ejecutan siempre.
  - Los ciclos no se ejecutan la misma cantidad de veces – suponemos que se ejecutan "muchas" veces.

Ajustar la fórmula según el modelo de costo de la arquitectura particular mejora la aproximación.

UNIVERSIDAD SIMÓN BOLÍVAR

Cálculo de costo

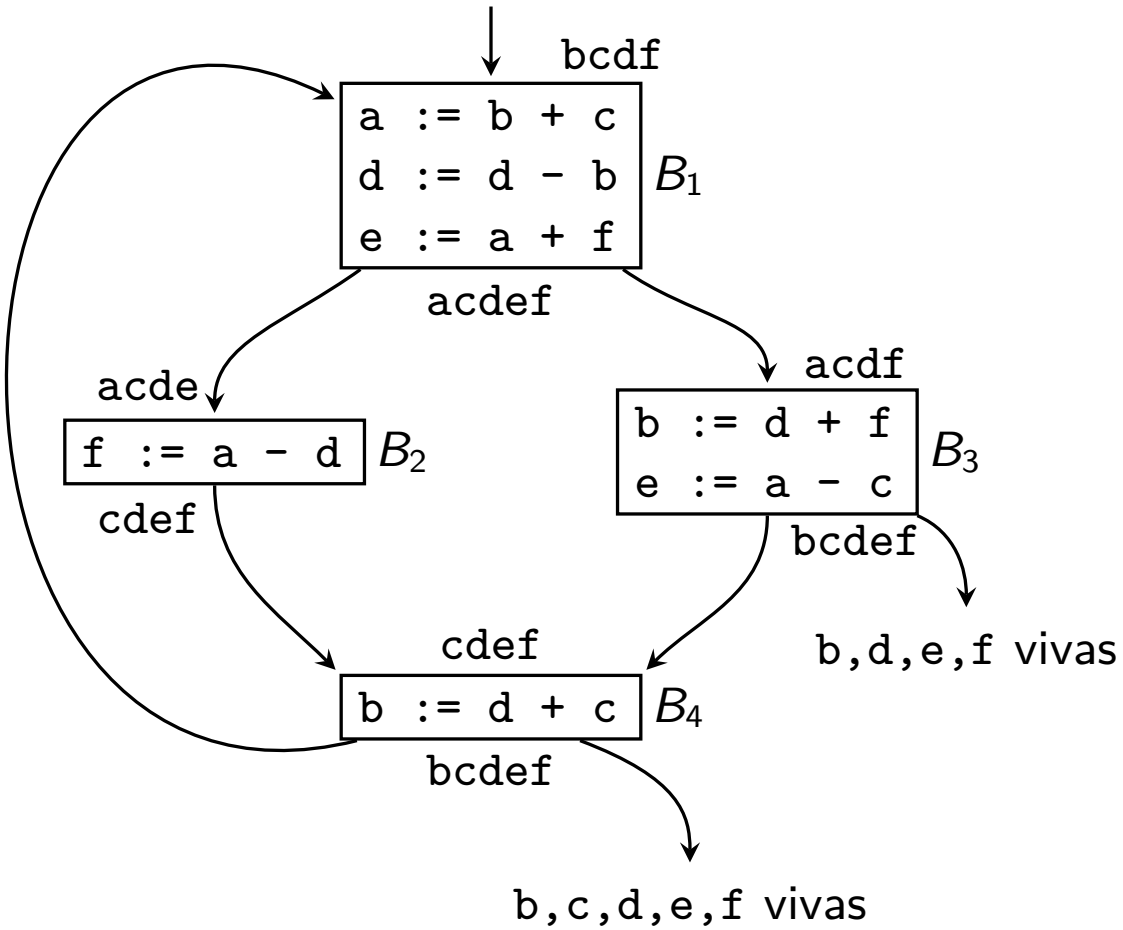
- R0, R1 y R2 para asignación global.



Cálculo de costo

- R0, R1 y R2 para asignación global.
- a viva solamente al salir de B_1 –

$$\sum 2 \times live(a, B) = 2$$



UNIVERSIDAD SIMÓN BOLÍVAR

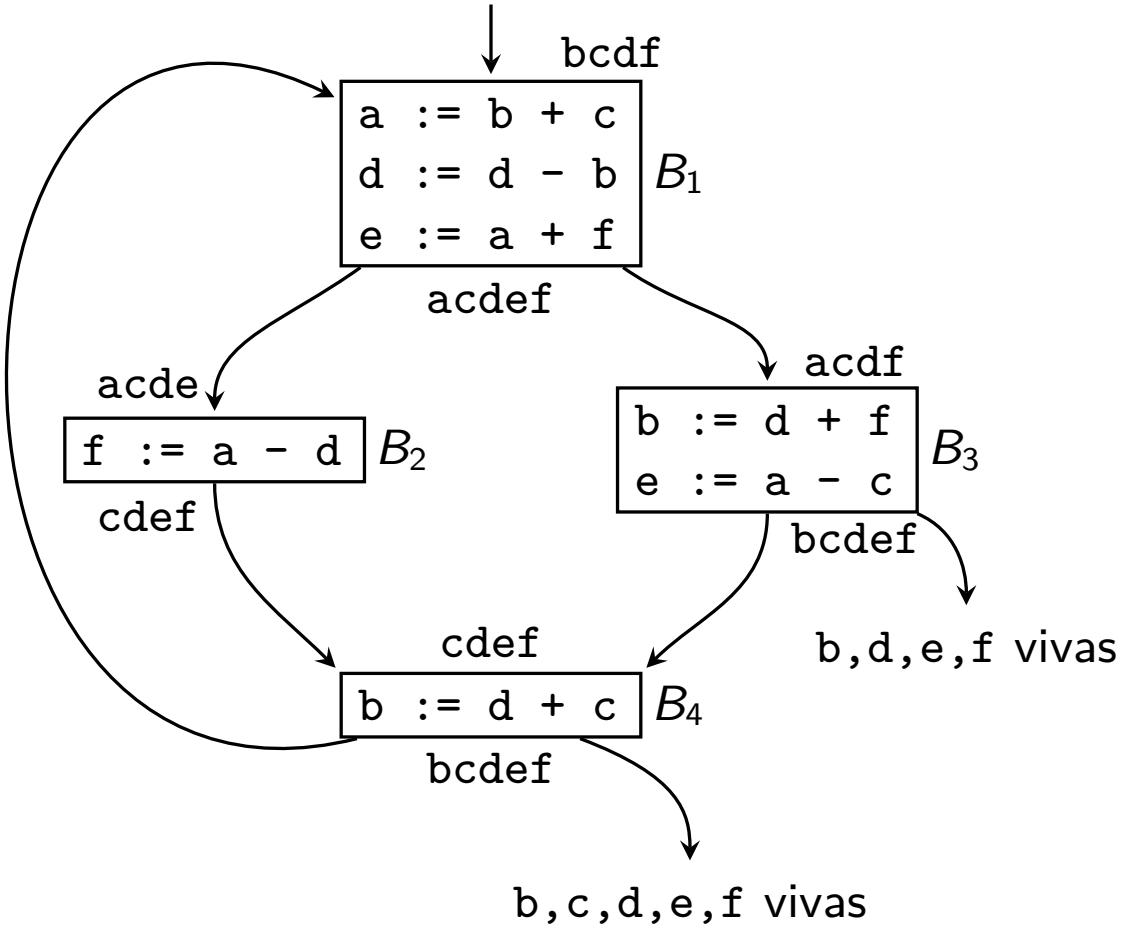
Cálculo de costo

- R0, R1 y R2 para asignación global.
- a viva solamente al salir de B_1 –

$$\sum 2 \times live(a, B) = 2$$

- B_2 y B_3 usan a –

$$\sum use(a, B) = 2$$



Cálculo de costo

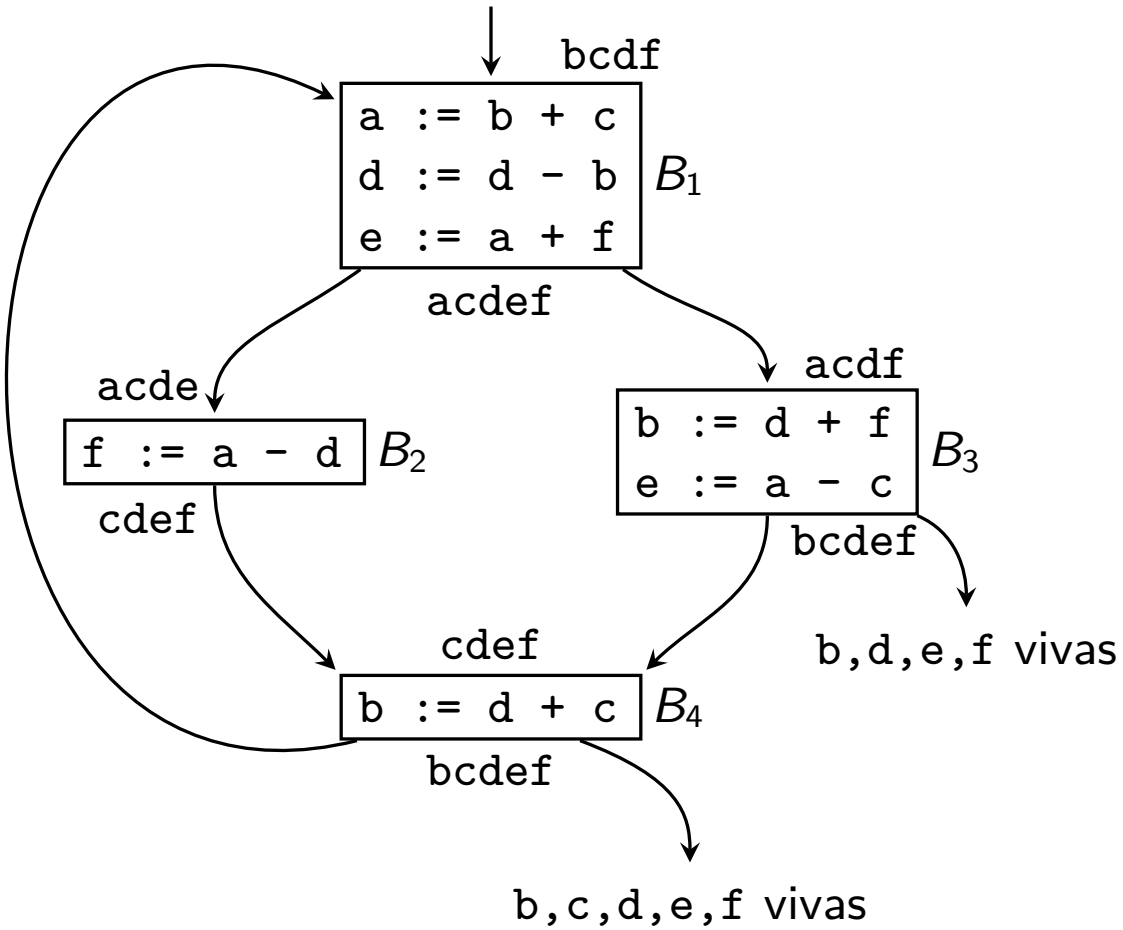
- R0, R1 y R2 para asignación global.
- a viva solamente al salir de B_1 –

$$\sum 2 \times live(a, B) = 2$$

- B_2 y B_3 usan a –

$$\sum use(a, B) = 2$$

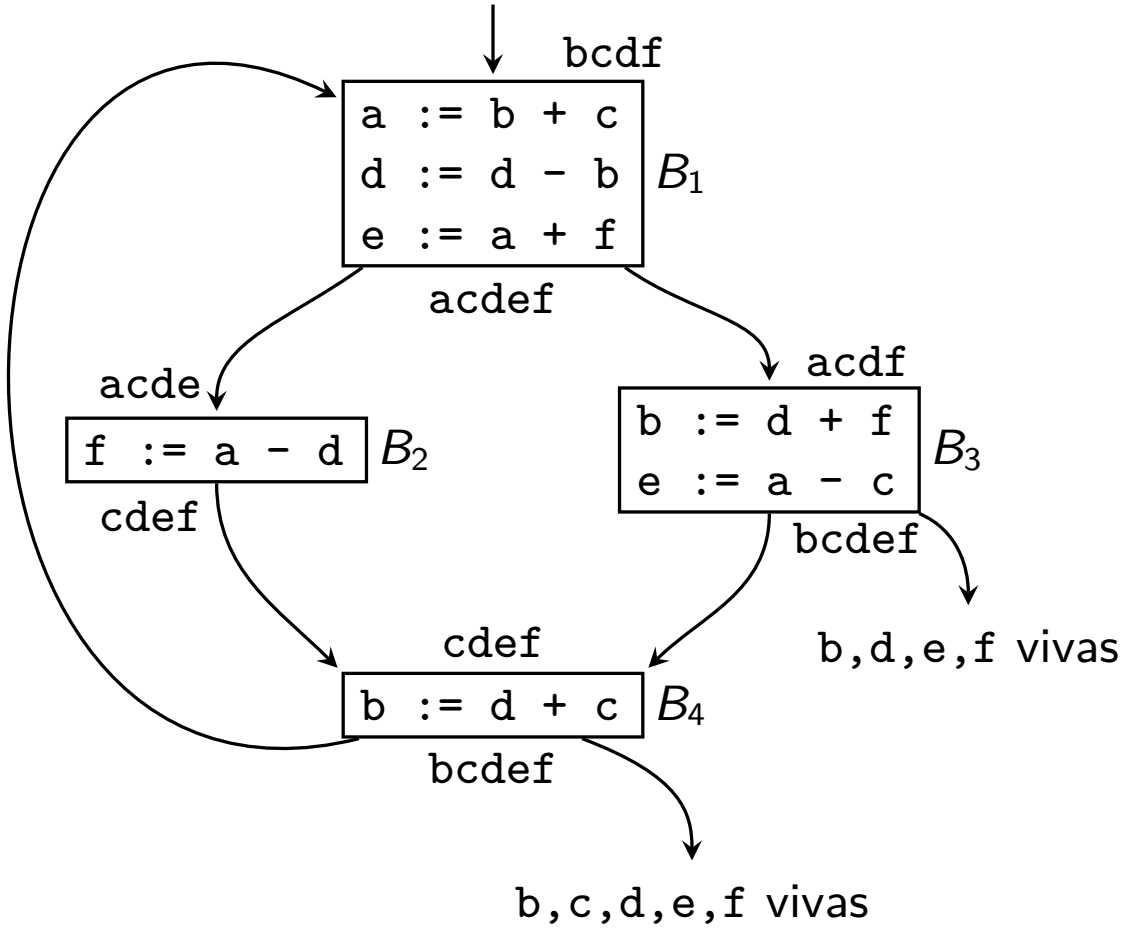
- $Benefit(a) = 4$



UNIVERSIDAD SIMÓN BOLÍVAR

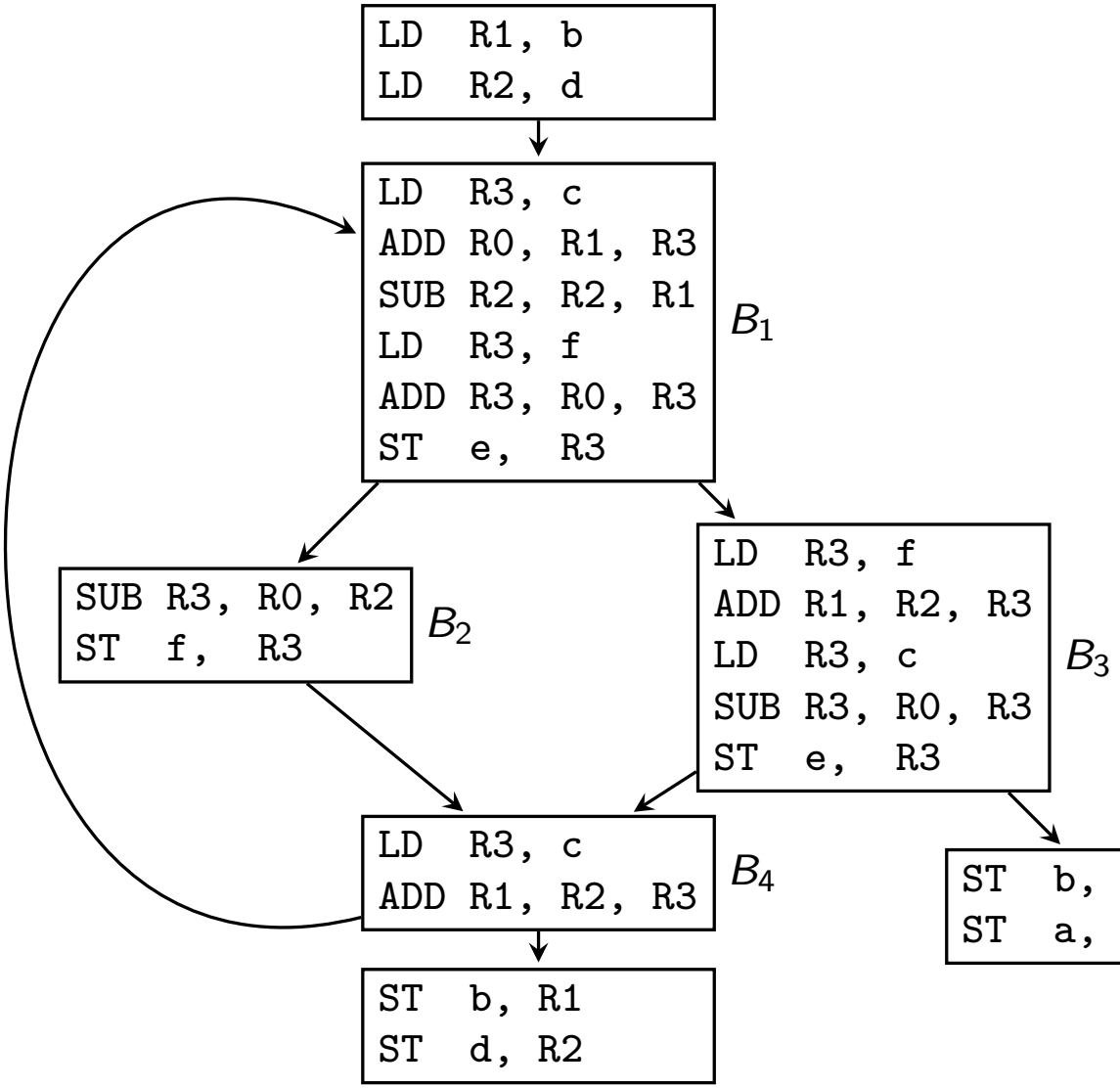
Cálculo de costo

- $Benefit(a) = 4$
- $Benefit(b) = 5$
- $Benefit(c) = 3$
- $Benefit(d) = 6$
- $Benefit(e) = 4$
- $Benefit(f) = 4$



UNIVERSIDAD SIMÓN BOLÍVAR

Código generado



Asignamos

- R0: a – (de a, e y f)
- R1: b
- R2: d

UNIVERSIDAD SIMÓN BOLÍVAR

¿Y si los ciclos están anidados?

Repetir hacia afuera con los registros restantes

- Al completar la asignación para un ciclo interno, se extiende la técnica hacia los ciclos exteriores.
- Si un ciclo L_1 contiene a un ciclo L_2
  - Nombres asignados a registros en L_2 no necesitan registros en $L_1 - L_2$.
  - Si x tiene un registro asignado en L_1 pero no lo tiene en L_2 se genera un almacenamiento antes de entrar y se vuelve a cargar al salir – no sabemos qué ocurre *dentro* de L_2.
  - Si x tiene un registro asignado en L_2 pero no lo tiene en L_1 se genera una carga antes de entrar y se almacena al salir – no sabemos qué ocurre *fuera* de L_2.

UNIVERSIDAD SIMÓN BOLÍVAR

Asignación por Coloración de Grafos

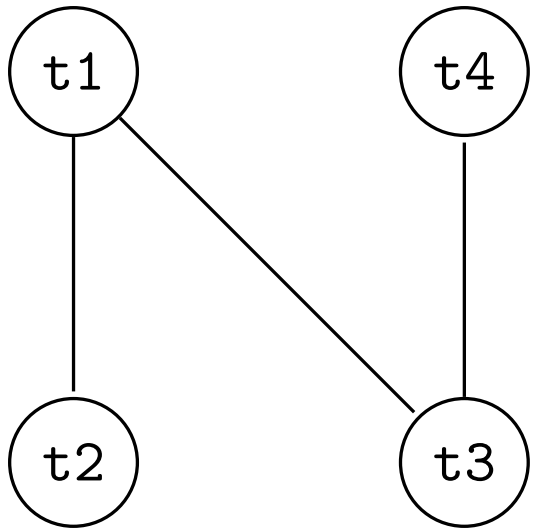
- Los métodos de asignación locales y globales deben completar escrituras (*spills*) cuando sea necesario reutilizar un registro.
- El problema de *minimizar* la cantidad de *spills* es isomorfo con la Coloración de Grafos (NP-completo) – Cocke, 1971.
- Primera implantación práctica hecha por IBM en 1981 – [Chaitin]
- Los compiladores modernos implantan la técnica en dos pasadas:
  1. Generar código suponiendo que hay *infinitos* registros asignables.
  2. Asignar registros minimizando los *spills*.

UNIVERSIDAD SIMÓN BOLÍVAR

Grafo de Interferencia

- El grafo se construye durante la segunda pasada
  - Se agrega un nodo por cada registro *simbólico* usado durante la generación – valores vivos a lo largo del código.
  - Se agrega una arista entre dos nodos si los tiempos de vida de esos registros simbólicos se *solapan* – "interferencia" entre los valores.

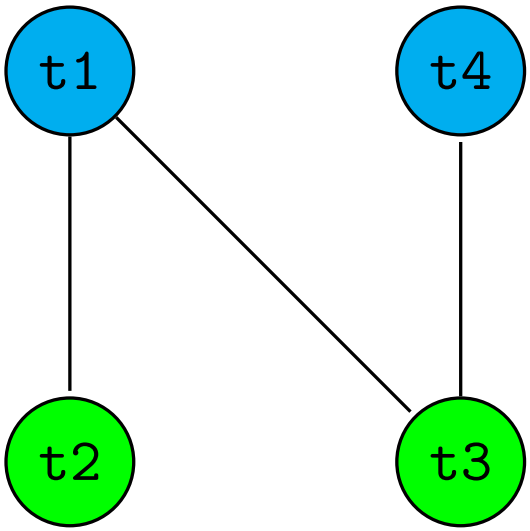
```
t1 := ...
t2 := ...
... t2 ...
t3 := ...
... t1 ...
t4 := ...
... t4 ...
... t3 ...
```



UNIVERSIDAD SIMÓN BOLÍVAR

Draw something

- Se intenta colorear el grafo con hasta k colores – tantos como registros reales asignables.
  - Los colores en ambos extremos de una arista han de ser diferentes.
  - La coloración final indica cuál registro asignar para cada valor.



Asignación

R1 – t1 y t4
R2 – t2 y t3

```
R1  :=  ...
R2  :=  ...
...  R2  ...
R2  :=  ...
...  R1  ...
R1  :=  ...
...  R1  ...
...  R2  ...
```

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo determinar si es $k-$coloreable?

Encontrar el número cromático

- Con mucho...

¿Cómo determinar si es $k-$coloreable?

Encontrar el número cromático

- Con mucho. . . cómputo – NP-Completo en general.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo determinar si es $k-$coloreable?

Encontrar el número cromático

- Con mucho. . . cómputo – NP-Completo en general.
- Heurística usualmente aplicada en la práctica
  - Sea un nodo $n \in G$ con menos de k vecinos.
  - Eliminar n y sus aristas de G para obtener G'.
  - Si G' es $k-$coloreable, G también es $k-$coloreable, siempre que n tenga como color uno que no tengan sus vecinos.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo determinar si es $k-$coloreable?

Encontrar el número cromático

- Con mucho. . . cómputo – NP-Completo en general.
- Heurística usualmente aplicada en la práctica
  - Sea un nodo $n \in G$ con menos de k vecinos.
  - Eliminar n y sus aristas de G para obtener G'.
  - Si G' es $k-$coloreable, G también es $k-$coloreable, siempre que n tenga como color uno que no tengan sus vecinos.
- Se repite la eliminación hasta. . .
  - Obtener el grafo vacío – coloración es posible y simple durante la reconstrucción.
  - Obtener un grafo cuyos nodos tienen k o más vecinos – coloración no es posible.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo determinar si es $k-$coloreable?

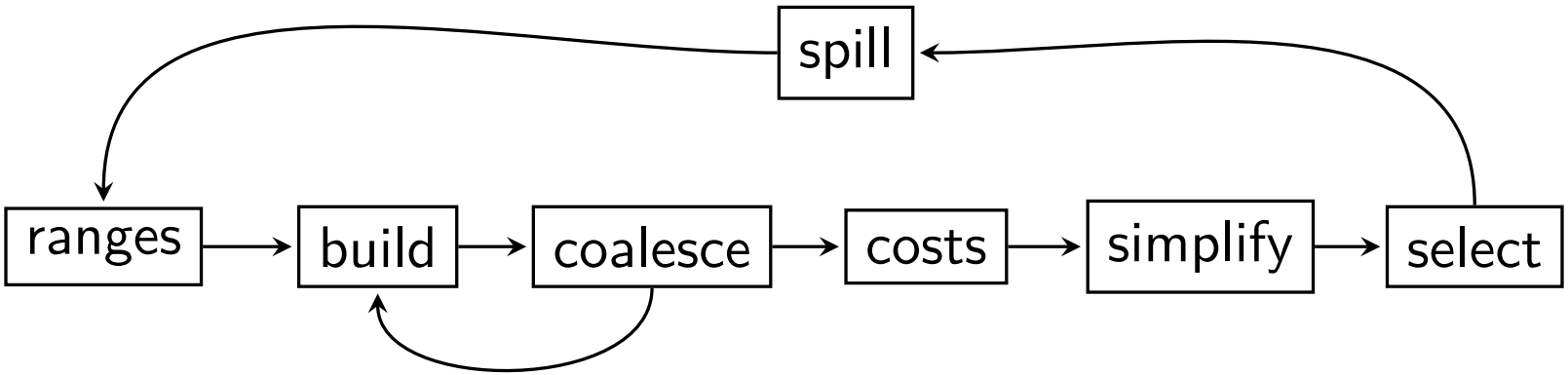
Encontrar el número cromático

- Con mucho. . . cómputo – NP-Completo en general.
- Heurística usualmente aplicada en la práctica
  - Sea un nodo $n \in G$ con menos de k vecinos.
  - Eliminar n y sus aristas de G para obtener G'.
  - Si G' es $k-$coloreable, G también es $k-$coloreable, siempre que n tenga como color uno que no tengan sus vecinos.
- Se repite la eliminación hasta. . .
  - Obtener el grafo vacío – coloración es posible y simple durante la reconstrucción.
  - Obtener un grafo cuyos nodos tienen k o más vecinos – coloración no es posible.
- Cuando la coloración no es posible, generar *spill* de nodos – heurísticas de selección según [Chaitin]

UNIVERSIDAD SIMÓN BOLÍVAR

Coloración Optimista

Método Chaitin/Briggs

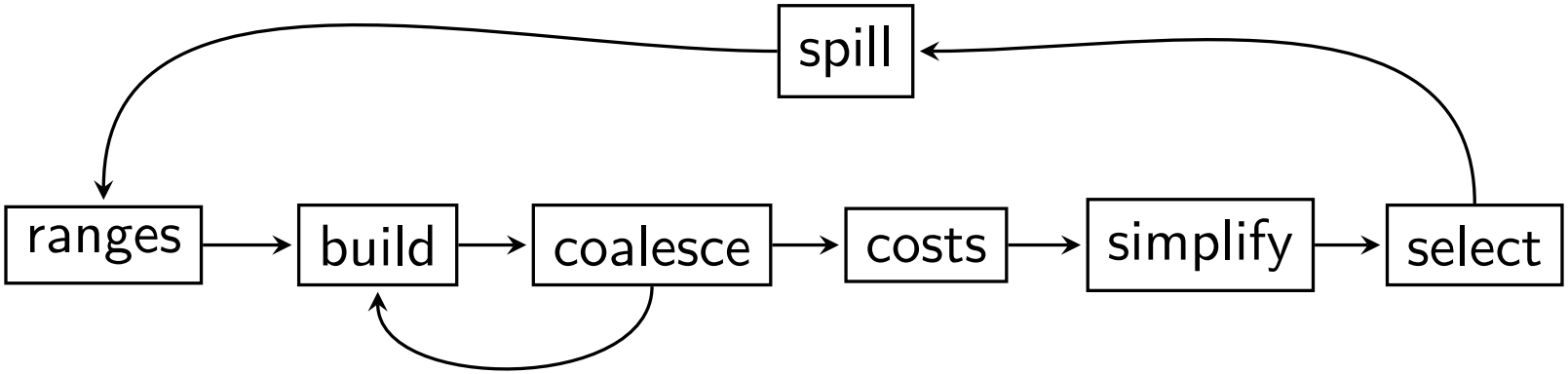


Variante más efectiva y comúnmente empleada.

UNIVERSIDAD SIMÓN BOLÍVAR

Coloración Optimista

Método Chaitin/Briggs – Rangos de Vida



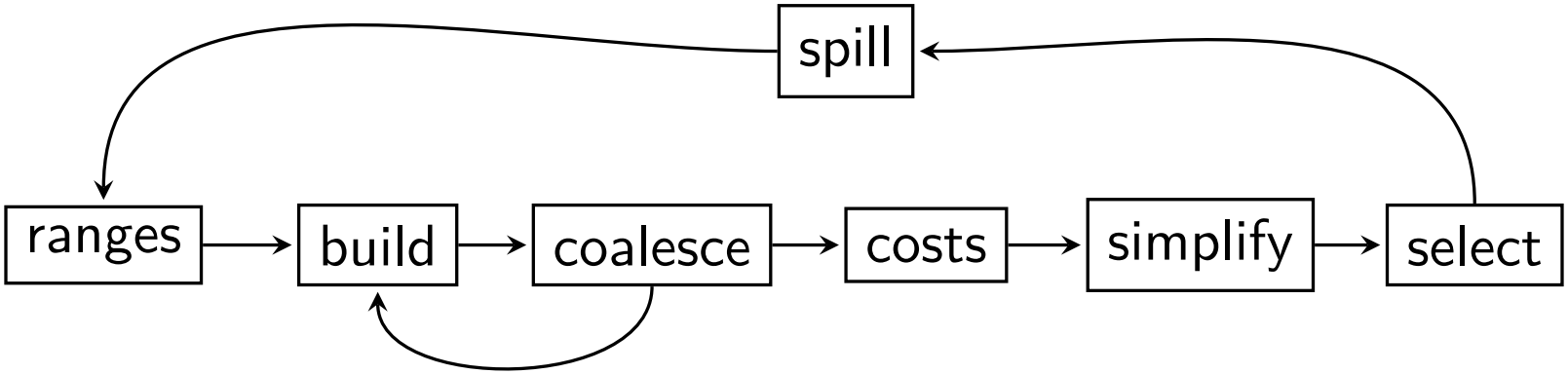
Ranges

- Determinar rangos de vida de registros virtuales – difícil porque los rangos no siempre son **contiguos**.
- Requiere técnicas de análisis de flujo – las estudiaremos pronto.

UNIVERSIDAD SIMÓN BOLÍVAR

Coloración Optimista

Método Chaitin/Briggs – Grafo de Interferencia



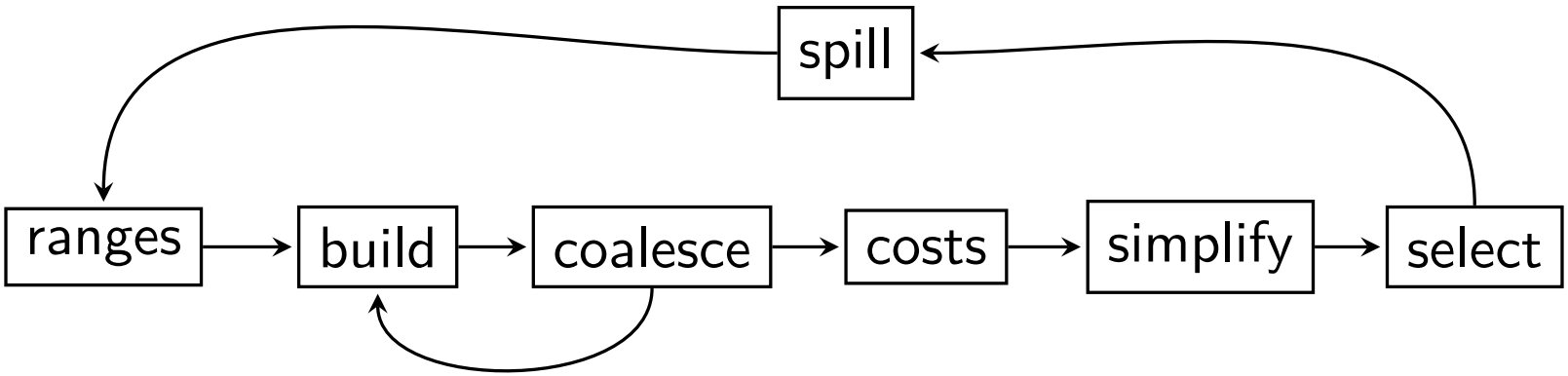
Build

- Construir grafo de interferencia – matriz **y** lista de adyacencia.
- Se agrega información para otras optimizaciones simultáneas – interferencia entre parámetros de funciones es la más frecuente.

UNIVERSIDAD SIMÓN BOLÍVAR

Coloración Optimista

Método Chaitin/Briggs – Eliminar copias

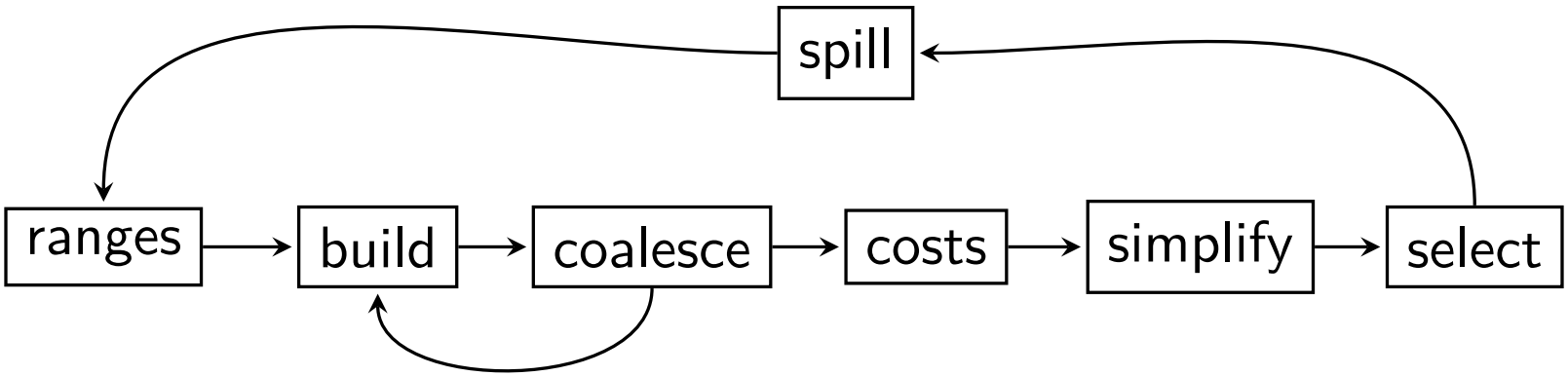


Coalesce

- Eliminar instrucciones de copia registro a registro.
- Podría cambiar la interferencia – reconstruir el grafo.

Coloración Optimista

Método Chaitin/Briggs – Costo



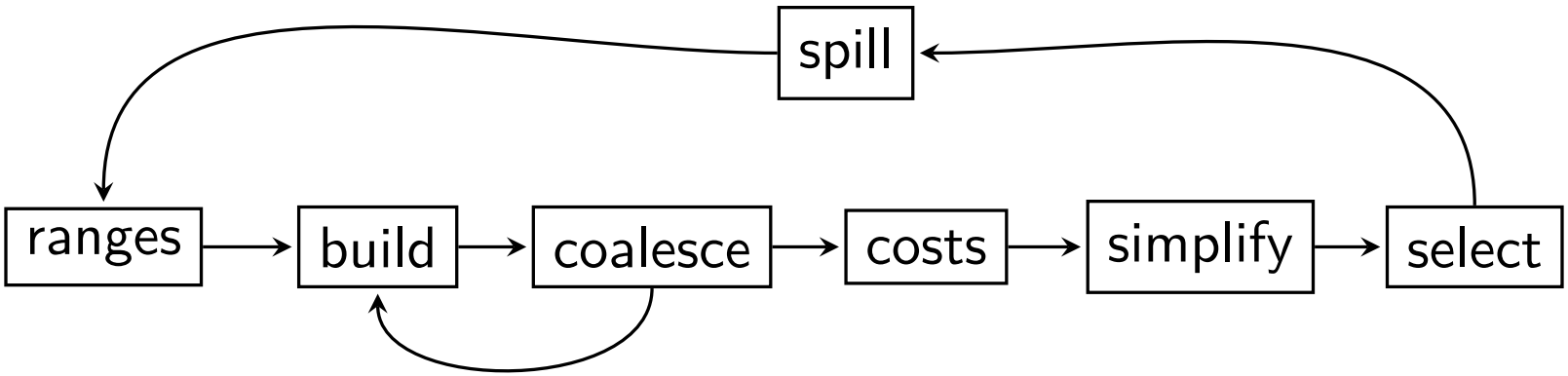
Costs

- Calcular el costo de *spills*.
- Heurísticas artísticas – contar referencias, participación en ciclos.

UNIVERSIDAD SIMÓN BOLÍVAR

Coloración Optimista

Método Chaitin/Briggs – Orden de asignación



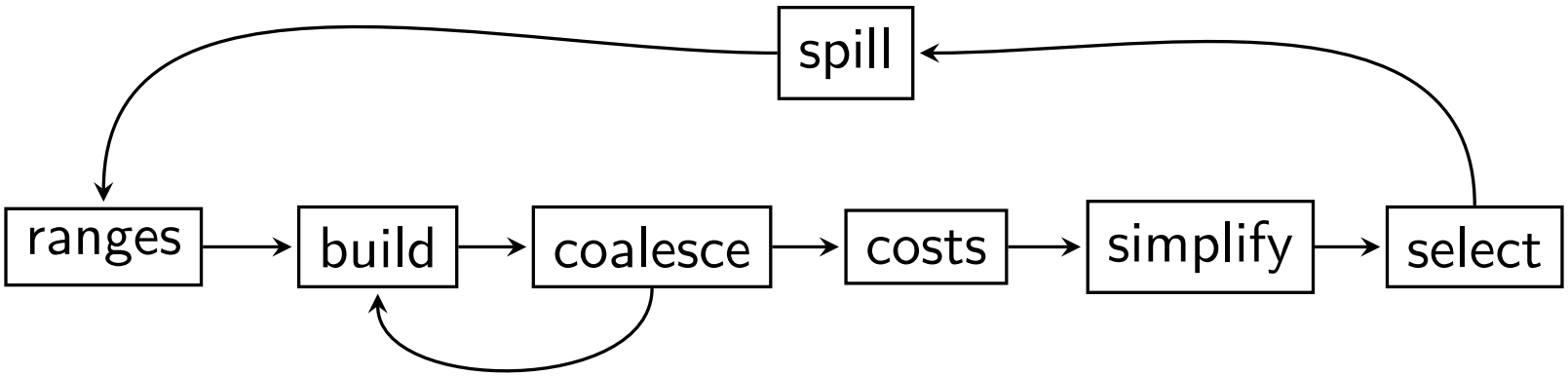
Simplify a.k.a. *Pruning*

- Heurística de Coloración – pila para recordar los pasos.
- Decisiones de *spill* al encontrar nodos con k vecinos.

UNIVERSIDAD SIMÓN BOLÍVAR

Coloración Optimista

Método Chaitin/Briggs – Asignación concreta



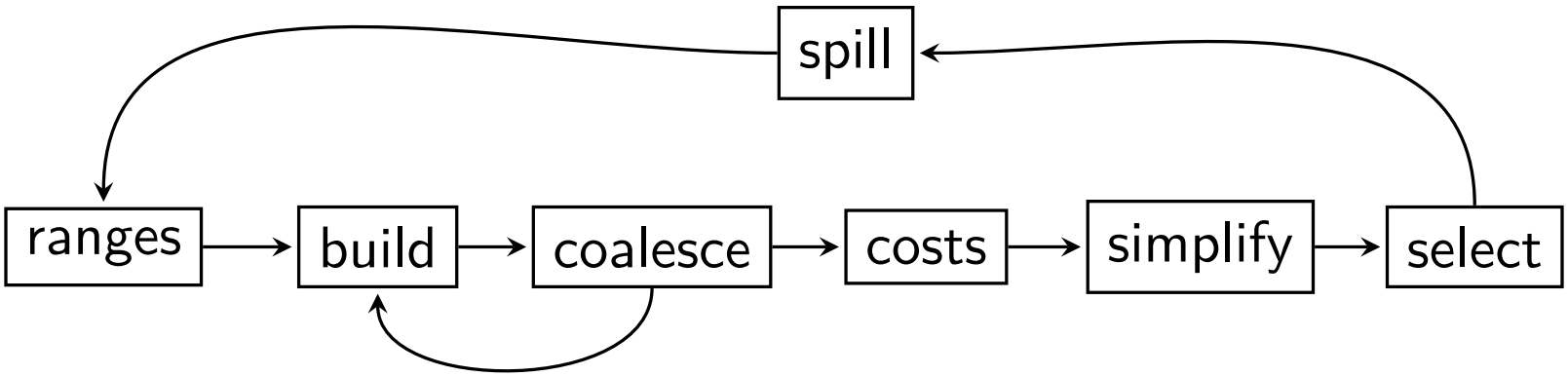
Select

- Selecciona un color para cada nodo a medida que los saca de la pila.
- Nodos con *spill* no se colorean.

UNIVERSIDAD SIMÓN BOLÍVAR

Coloración Optimista

Método Chaitin/Briggs – Rematerialización



Spill

- Si quedaron nodos sin colorear se genera el código para *spill*.
- Se repite la asignación de registros.

UNIVERSIDAD SIMÓN BOLÍVAR

Derramar no es trivial

Alternativas para el código de *spill*

- "Regaíto" – Cargar y almacenar alrededor de todos los usos.
  - Fácil de implantar.
  - Reduce el número de iteraciones de coloreado.
  - No es equivalente a "quitarlo" del grafo –
    lo que hace es crear *varios* sub-tiempos de vida.

UNIVERSIDAD SIMÓN BOLÍVAR

Derramar no es trivial

Alternativas para el código de *spill*

- "Regaíto" – Cargar y almacenar alrededor de todos los usos.
  - Fácil de implantar.
  - Reduce el número de iteraciones de coloreado.
  - No es equivalente a "quitarlo" del grafo – lo que hace es crear *varios* sub-tiempos de vida.
- Fragmentado – generar varios intervalos de vida
  - Analizar las nuevas interacciones – buscar aquellas en las que realmente es necesario el *spill*.
  - Requiere más iteraciones de coloreado.
  - Heurísticas basadas en dominación de bloques – hablaremos del tema más adelante.

UNIVERSIDAD SIMÓN BOLÍVAR

Derramar no es trivial

Alternativas para el código de *spill*

- "Regaíto" – Cargar y almacenar alrededor de todos los usos.
  - Fácil de implantar.
  - Reduce el número de iteraciones de coloreado.
  - No es equivalente a "quitarlo" del grafo –
    lo que hace es crear *varios* sub-tiempos de vida.
- Fragmentado – generar varios intervalos de vida
  - Analizar las nuevas interacciones –
    buscar aquellas en las que realmente es necesario el *spill*.
  - Requiere más iteraciones de coloreado.
  - Heurísticas basadas en dominación de bloques –
    hablaremos del tema más adelante.
- Jerárquico – *tiling* del código
  - Dividir el código en *tiles* contenidos en *tiles* contenidos en *tiles*...
  - Se aplica coloreado para cada *tile*.
  - Si quedan conflictos, los resuelve el *tile* de orden superior.

UNIVERSIDAD SIMÓN BOLÍVAR

Bibliografía

- [*Aho*]
  - Sección 8.8
  - Ejercicios 8.8.1 y 8.8.2
- [*Chaitin*]
  Register Allocation and spilling via graph coloring
- [*Briggs*]
  Register Allocation via Graph Coloring

UNIVERSIDAD SIMÓN BOLÍVAR