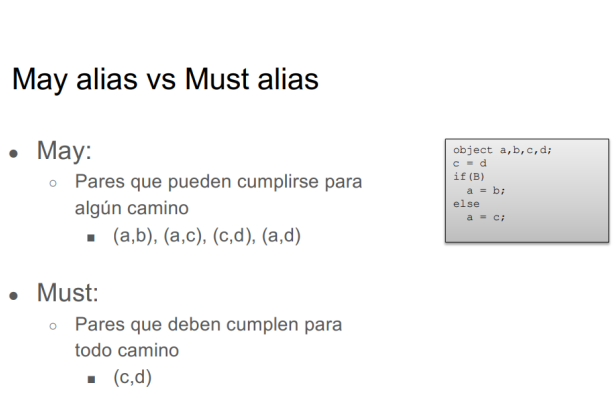
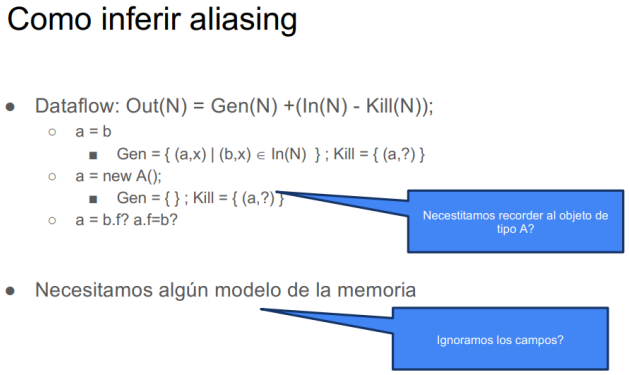
Alias analysis

Objetivo: Hacer un analisis del código para modelar aliasing

> aliasing: multiples variables apuntan a la misma posición en memoria de forma directa o indirecta. (Para la interpretación de modelado: multiples variables representan la misma locación mutable)

El analisis puede ser may alias (pueden cumplirse para algún camino) o must alias (se cumple para todo camino)





Con este modelado de generar tuplas no podemos modelar campos (a = b.f; a.f = b). Para ello necesitamos alguna forma de modelar la memoria (en particular el heap).

Points-to analysis

Objetivo: armar un analisis que nos diga a donde apunta cada puntero

Un anlisis de points-to es un may forward dataflow analysis que permite determinar a que objetos puede apuntar una variable durante la ejecución de un programa.

Se simplifica a 4 operaciones

```
x = new C(); // new
x = y // copy
x = y.f // load
x.f = y // store
```

Tipos de analisis

- Sensibles a flujo: computan un analisis points-to para cada punto del programa (son caros)
- Insensibles al flujo: hacen un analisis points-to para todo el programa en general y todas las asignaciones son no destructivas. Hay 2 tipos
  - Andersen: basado en inclusión
  - Steengard: basado en unificación (no entra en el parcial)
- Sensibles a contexto: Pueden distinguir entre distintos llamados a metodos.

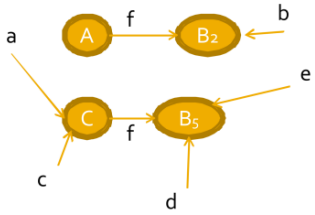
Points-to Graph (PTG)

Grafo donde tengo nodos por cada objeto (basicamente por cada new), nodos por cada puntero (o variable de tipo objeto, basicametne para trackear el aliasing), y ejes según las relaciones entre objetos, sus campos, y variables (punteros).

Para implementar el análisis utilizaremos un *points-to graph* (PTG) definido como $G = \langle N, E, L \rangle$ donde

- N es el conjunto de nodos del grafo, y representa los objetos creados por el programa. Cada nodo representa *todos* los objetos creados por una sentencia `new` del programa. Es decir, se creará un solo nodo incluso si el *new* se ejecuta dentro de un ciclo.
- E es el conjunto de aristas del grafo, y representa los objetos alcanzables desde cada objeto mediante campos. Una arista $e = (n_1, f, n_2)$ en el grafo indica que el nodo n_2 puede ser alcanzado desde el nodo n_1 mediante el campo f. Tener en cuenta que n_1 y n_2 representan ambos un conjunto de objetos.
- $L : Local \to P(N)$ es una función de etiquetado que asocia a cada variable local del programa un conjunto de nodos del grafo. Es decir, $L(x)$ es el conjunto de nodos a los que puede apuntar la variable x.

Gráficamente, un $PTG = \langle \{A, B_2, B_5, C\}, \{(A, f, B_2), (C, f, B_5)\}, [a \leftarrow \{C\}, b \leftarrow \{B_2\}, c \leftarrow \{C\}, d \leftarrow \{B_5\}, e \leftarrow \{B_5\}] \rangle$ luce de la siguiente forma:



El PTG debe ser utilizado como un reticulado por lo que hay que definir sus operaciones básicas:

- $\bot = \langle \{\}, \{\}, L \rangle$: con $L(x) = \{\}$ para toda variable local.
- $G_1 \subseteq G_2$ si $L_1 \subseteq L_2$ ó $(L_1 = L_2 \wedge N_1 \subseteq N_2)$ ó $(L_1 = L_2 \wedge N_1 = N_2 \wedge E_1 \subseteq E_2)$.
- $G_1 \cup G_2 = \langle N_1 \cup N_2, E_1 \cup E_2, L_1 \cup L_2 \rangle$.

Las reglas Dataflow para el análisis son las siguientes:

Expresión	Efecto
$p : x = new A()$	$G' = G$ con $L'(x) = \{p\}$, donde p es el número de línea en el programa.
$x = y$	$G' = G$ con $L'(x) = L(y)$
$x = y.f$	$G' = G$ con $L'(x) = \{n_2 \mid (n_1, f, n_2) \in E \wedge n_1 \in L(y)\}$
$x.f = y$	$G' = G$ con $E' = E \cup \{(n_1, f, n_2) \mid n_1 \in L(x) \wedge n_2 \in L(y)\}$

Pueden asumir que no hay variables globales.
Resúmen de las reglas en formato gráfico:

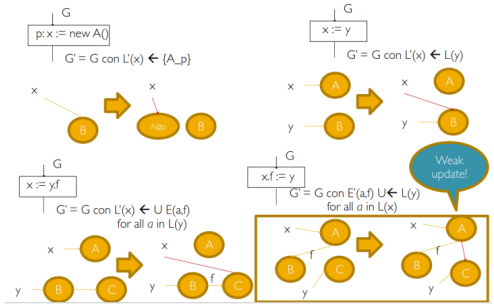
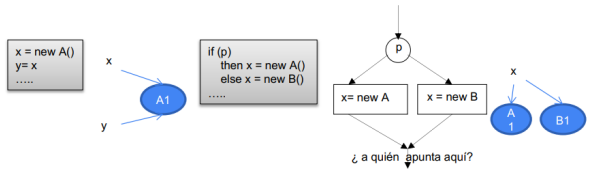


Figura 1: Weak update: No se sabe exactamente qué objeto va a ser actualizado; podría llegar por diferentes caminos.

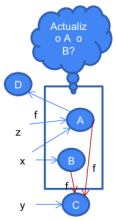
4ta operación dataflow

El weak update se justifica porque se tiene un analisis may. En el siguiente caso tenemos que x apunta a A y B. Luego si hacemos $x.f = y$ no sabemos si actualizar el campo f de A o B. Por eso ponemos ambas aristas.

- Si el points-to graph tiene ejes (*a*,*A*) y(*a*,*B*) una variable puede apuntar a *A* o B (MAY)
- Se puede mejorar con sensibilidad a caminos



- Strong update:
  - Cuando se sabe exactamente qué objeto está siendo escrito
  - Example: x := y
  - Se puede reemplazar la información de points-to de x
- Weak update:
  - No se sabe exactamente que objeto va a ser actualizado.
  - Ejemplo: x.f :=y
  - Durante el análisis puede no saberse a donde podría apuntar x
- Pequeña mejora: si el grado de salida es **1** se puede aplicar un strong update



El PTG como reticulado calcula un PTG por cada punto del porgrama empezando con el grafo vacio y siguiendo la regla de inclusión definida en las operaciones del reticulado (imagen).

Abstracciones del heap

Podemos modelar el heap como un solo nodo, un nodo por cada allocation site (cada new). Este ultimo soluciona el problema de tener un new dentro de un bucle, en tal caso tendriamos que modelar que cada iteración es un objeto nuevo.

```
...
while(i < k) {
    A a = new A();
    b[i] = a;
    ...
}
...
```

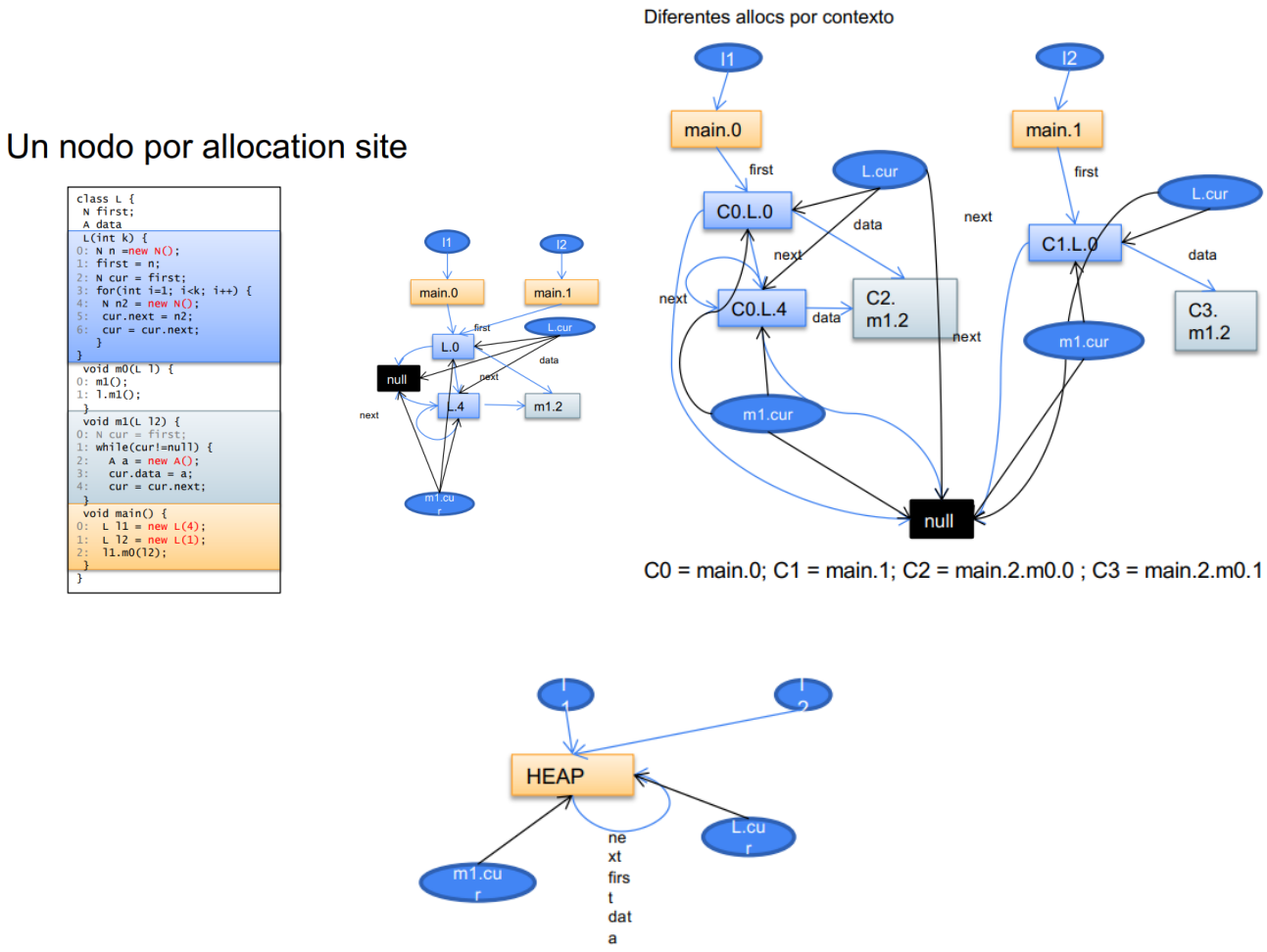


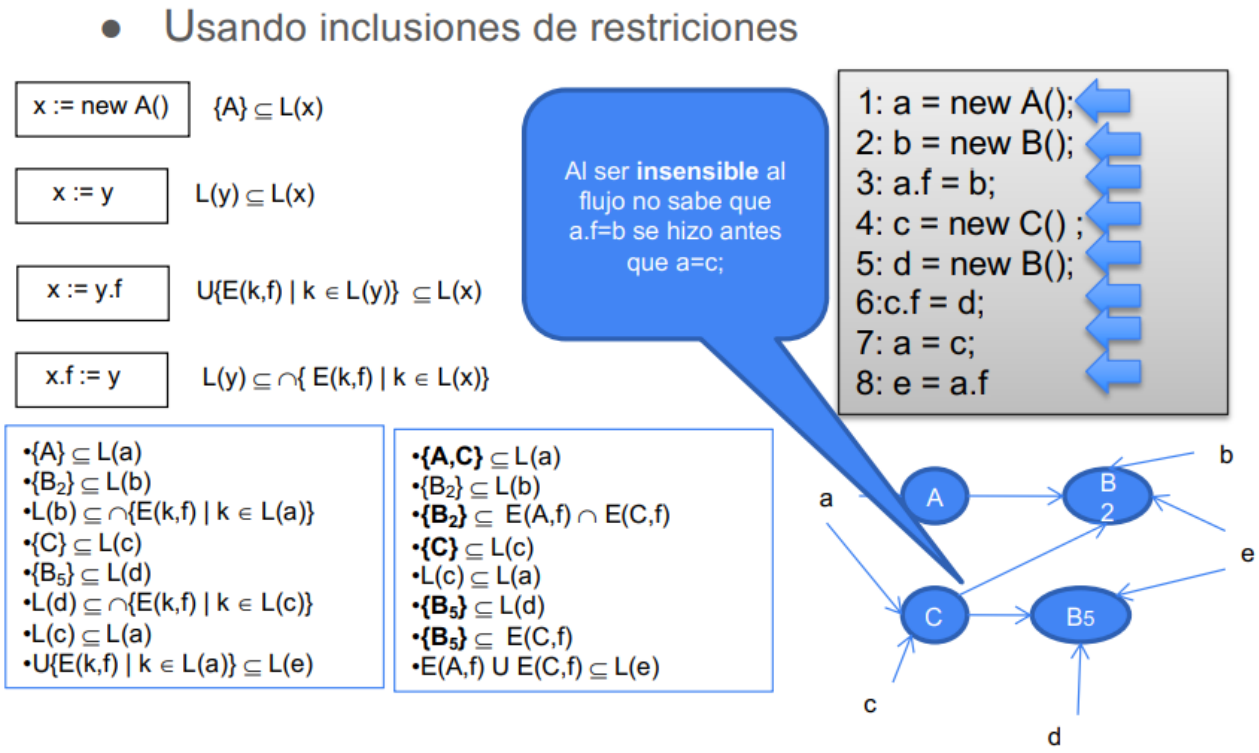
Figure 1: alt text

Andersen

Se basa en inclusiones: $p = q, pt(q) \subseteq pt(p)$. Es insensible a contexto y flujo (analisa todo el programa de una). Analisis may.

El algoritmo consiste en recorrer el codigo calculando las restricciones y luego resolverlas con la operación de inclusión hasta llegar a un punto fijo.

X = `alloc` P:	`alloc-`$i \in [\![X]\!]$
X_1 = &X_2:	$X_2 \in [\![X_1]\!]$
X_1 = X_2:	$[\![X_2]\!] \subseteq [\![X_1]\!]$
X_1 = *X_2:	$c \in [\![X_2]\!] \implies [\![c]\!] \subseteq [\![X_1]\!]$ for each $c \in Cell$
*X_1 = X_2:	$c \in [\![X_1]\!] \implies [\![X_2]\!] \subseteq [\![c]\!]$ for each $c \in Cell$



En palabras, cada instrucción define restricciones a cumplir sobre los objetos del programa:

$x = new(A)$: A pertenece a los objetos apuntados por x

$x = y$: el conjunto de objetos apuntados por x deben tener a todos los objetos apuntados por y. (No se borran aristas, en la imagen justo se tapa pero la arista $a- > A$ no se borró).

$x = y.f$: los objetos apuntados por x deben tener a todos los objetos alcanzables por y a travez de f

$x.f = y$: todos los objetos apuntas por y tienen que pertenecer a la intersección del conjunto de objetos formado por los objetos alcanzables por x a travez de f