TL;DR

Recorro el callgraph en orden topológico (de las hojas hacia el root) y aumento/reemplazo el binding trivial entre argumentos y parámetros (que pueden ser objetos) según las variables relevantes del callee (si son fields de un parámetro).

Esta enterización del binding se debe ir propagando hacia arriba. Si un argumento del binding caller → callee provenía a su vez de un parámetro del caller, se lo debe tener en cuenta para la enterización de los bindings de todos los llamados a *ese* caller. Si no provenía de un parámetro del caller, se va a eliminar existencialmente así que lo ignoro.

Nomenclatura:

- Parámetros relevantes: los parámetros **enterizados** de un método que son usados dentro del mismo y que podrían aparecer en los invariantes
- Variables relevantes: variables enteras (ints, sizes de listas, fields de objetos) que fueron considerados relevantes por el análisis (Live Variable Analysis)
- Variables inductivas: un subconjunto de las variables relevantes que son las que se van incrementando adentro de un loop

Problema: ¿Cómo hacer el binding entre argumentos de caller y parámetros relevantes de callee?

1) Por cada llamada caller → callee, armar el binding entre argumentos de caller y parámetros de callee, versión 'objetos'.

Por ejemplo si tengo

```
m1(obj b)
{
        m_2(b);
}


m2(obj c)
{
        //c.f es una variable relevante
}
```

Mantener la relación $b \to c$ en alguna estructura de datos.

2) En realidad tengo que generar un binding entre variables enteras. ¿Entre qué variables enteras? Necesariamente variables como $c.f$ del ejemplo tienen que estar bindeadas. Por cada parámetro relevante del callee agrego los bindings correspondientes.

Para eso me fijo las variables relevantes que aparecen a lo largo del callee (en cada Instrumentation Site). Usando los nombres del ejemplo, si una variable es

un field f de un parámetro c y teníamos el binding entre objetos $b \to c$ agrego un binding entre fields $b.f \to c.f$

3) Sin embargo esto no termina acá. Miremos una ampliación del ejemplo anterior:

```
m0(obj a)
{
        m_2(a);
}

m1(obj b)
{
        //b.f no es una variable relevante segun el analisis de relevantes,
        //ya que este es local
        m_2(b);
}

m2(obj c)
{
        //c.f es una variable relevante
}
```

Estos bindings se deben ir "propagando". En el ejemplo de arriba necesitamos saber que $a.f$ apunta a $b.f$ por cómo hacemos el análisis (composicional) de memoria. Al hacer el binding entre $m0$ y $m1$ necesitamos saber que $b.f$ es un parámetro relevante de $m1$. Para ello propongo el siguiente tuneo al paso **2)**:

Recorro el CallGraph pero ordenado topológicamente, de las hojas al root (o sea, siempre recorro primero el callee antes que el caller).

Por cada binding entre variables enteras que voy a armar entre caller y callee voy a agregar a una lista aparte L_{caller} las variables del lado del caller.

Esta lista **también** la voy a usar para armar los bindings, de la siguiente manera:

Además de las variables relevantes que aparecen a lo largo del callee (las de los Instrumentation Sites) miro también las variables de L_{callee}, que armé antes porque estoy recorriendo los callers con el orden topológico. Me voy a fijar si todas estas variables son fields de un parámetro del callee para armar el binding enterizado.

Observación: si una variable de L_{callee} no provenía de un parámetro, la voy a ignorar para después eliminarla existencialmente en el invariante.

En el ejemplo de arriba tendríamos lo siguiente:

i) Miro el call $m1 \to m2$. El binding entre objetos es $b \to c$. Quiero armar el binding entre variables enteras así que me fijo todos los Instrumentation Sites del callee, que es $m2$. En la práctica como a daikon ahora le podemos pasar objetos los Instrumentation Sites solo necesitan tener los objetos cuyos fields son relevantes, pero podemos mantener en memoria también los fields relevantes (y pasarle a daikon solo los objetos)

Por simplicidad asumamos que sólo aparece $c.f$ como relevante. Fijémonos que $L_{m2} = \emptyset$ porque acabamos de empezar. Entonces agrego al binding $b.f \to c.f$, y dejo como variable relevante extra a $b.f$, es decir, $L_{m1} = \{b.f\}$

Observación: el binding $b \to c$ se podría perfectamente tirar porque es un binding entre objetos, pero prefiero dejarlo para una optimización posterior porque no rompe nada y lo considero un feature medio estético extra.

ii) Ahora le toca al call $m0 \to m1$. El binding entre objetos es $a \to b$. Tenemos que $L_{m1} = \{b.f\}$.

Supongamos por simplicidad que no aparecen fields relevantes en el body de $m1$. Pero por lo anterior vamos a mirar L_{m1} también y entonce, como $b.f$ es un field de un parámetro de $m1$, agregamos $a.f \to b.f$, y agregamos como variable relevante extra a $a.f$, es decir, $L_{m0} = \{a.f\}$

Luego obtenemos que:

- En el call $m1 \to m2$ el binding es $b \to c, b.f \to c.f$
- En el call $m0 \to m1$ el binding es $a \to b, a.f \to b.f$

Claramente hay situaciones más complejas que voy a analizar, pero me parece que el algoritmo anda.