Extracting Dataflow Objects and other Flow Objects

Radu Vanciu **Marwan Abi-Antoun**

September 2013

Department of Computer Science
Wayne State University
Detroit, MI 48202

Abstract

The contribution of this technical report is a static analysis that extracts a hierarchical object graph with dataflow edges that refer to abstract objects. The analysis combines the aliasing precision provided by Ownership Domains with a domain-sensitive information flow analysis. The extracted dataflow edges refer to objects that are nodes in the object graph. The analysis also extracts flow objects shown on dataflow edges only. This report includes the formalization of the analysis and the proof of the soundness theorems.

Keywords: hierarchical object graphs, dataflow communication, ownership domains, flow objects

Contents

List of Figures

1 Introduction

A runtime architecture often shows multiple objects of the same type, and the objects are shown on the edges as messages. In previous work [9], we proposed a static analysis that extracts an hierarchical ownership object graph with dataflow edges as an approximation of the runtime architecture from code with Ownership Domain annotations. However, the dataflow edges show types rather than objects. In addition, previous analysis does not handle some extensions to Ownership Domains that increase the expressiveness of type system and are used in practice.

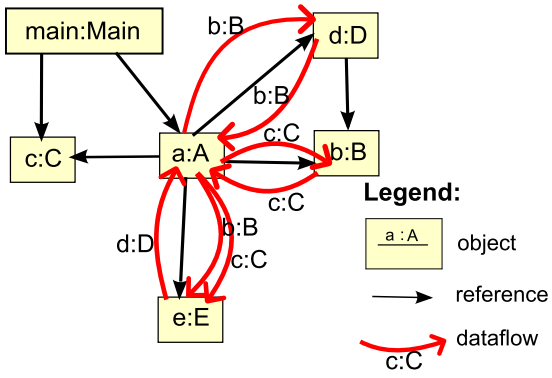
For additional expressiveness, Ownership Domains support `lent` for objects borrowed in method invocations, and `unique` for objects passed linearly [2]. Our goal is to extract an approximation of a runtime architecture and use it for security analysis. In this report, we focus on the details of the static analysis such as how the analysis extracts dataflow edges from expressions that involve references that are declared to be `lent` or `unique`.

In this technical report, we formally describe the static analysis that extracts a hierarchical object graph with dataflow communication edges that refer to abstract objects. There are two possibilities: the abstract object is in an actual or the abstract object is declared `unique`, and the analysis is unable to find an actual domain. In the later case, the analysis creates a flow object in a fresh domain. In particular, a flow object represents an abstract object that is passed linearly between other abstract objects such that none of these abstract objects have a reference to the flow object.

The abstract graph is sound such that, for any execution of the system, there is a mapping from the runtime object graph to the abstract graph. Each runtime object has exactly one representative abstract object or flow object in the abstract graph. Each runtime edge has a corresponding abstract edge between the representatives of the source and of the destination. We use a constraint-based specification instead of transfer functions to describe the analysis. This formalizes the static analysis as a set of inference rules, and makes it easier to prove soundness, i.e., we prove the mapping exists and it is an over-approximation of a runtime object graph for any execution of the system.

The technical report is organized as follows. Section 2 defines dataflow communication. Section 3.1 reviews ownership domains using Featherweight Domain Java (FDJ). Section 3.2 formalizes the Object Graph (OGraph). Section 3.6 formally presents the proof of the soundness theorems. Section 4 describes the extension of the analysis.

```
1  class Main{
2    A a; C c;
3    void main(){
4      main = new Main();
5      main.run();
6    }
7    void run(){
8      a = new A(c);
9      //no dataflow communication
10     a.m1(); //method call
11   }
12 }
```




```
1  class A{
2    B b; C c; D d; E e;
3    A(C c){
4     //no dataflow communication
5     //field initialization
6      this.c = c;
7    }
8    void m1(){
9    //method call (a -B-> d)
10     d.setB(b);
11   }
12   B m2(){
13   //method call (d -B-> a)
14     return d.getB();
15   }
16   C m3(){
17   //field read (b -C-> a)
18     return b.c;
19   }
20   void m4(){
21   //field write (a -C-> b)
22     b.c = c;
23   }
24   D m5(){
25   //method call(a -B-> e, a -C-> e, e -D-> a)
26     return e.me(b,c);
27   }
28 }
```

Figure 1: Example of dataflow communication.

2 Dataflow Communication

Object Graph. An object graph such as an OOG that is statically extracted has nodes representing abstract objects. An abstract object is a representative for a possible unbounded number of objects that may exist at runtime. Edges between abstract objects represent possible edges that may exist during an execution. We are interested in extracting dataflow edges because a dataflow communication may lead to security vulnerabilities such as information disclosure when the object that the dataflow edge refers to contains confidential information, and the destination is untrusted [8].

Dataflow communication means that an object `a:A` has a reference to an object `o:O` and passes it to an object `b:B`, or an object `a:A` has a reference to an object `b:B` and receives a reference to an object `o:O` [7]. The objects `a:A` and `b:B` represent the source or destination objects, and `o:O` is a dataflow object that the dataflow communication refers to. To capture the directionality of the flow an object graph has *import* and *export* edges. An import dataflow edge exists due to the return value of a method invocation or a field read. An export dataflow edge exists due to an argument of a method invocations or a field write.

Soundness. An object graph is sound if and only if there is a mapping between any runtime object graph and

$$\frac{\Gamma,\Sigma,\theta \vdash e : T_0 \qquad fields(T_0) = \overline{T}\ \overline{f} \qquad \Gamma,\Sigma,\theta \vdash e' : T \qquad T <: T_i}{\Gamma,\Sigma,\theta \vdash e.f_i = e' : T}[\text{T-WRITE}] \qquad \frac{S[\ell] = C\text{<}\overline{p}\text{>}(\overline{v}) \qquad fields(C\text{<}\overline{p}\text{>}) = \overline{T}\ \overline{f} \qquad S' = S[\ell \mapsto C\text{<}\overline{p}\text{>}([v/v_i]\overline{v})]}{\ell.f_i = v; S \rightsquigarrow v; S'}[\text{R-WRITE}]$$

$$\frac{\theta \vdash e_0; S \to e_0'; S'}{\theta \vdash e_0.f_i = e_1; S \to e_0'.f_i = e_1; S'}[\text{RC-WRITE-RCV}] \qquad \frac{\theta \vdash e_1; S \to e_1'; S'}{\theta \vdash v.f_i = e_1; S \to v.f_i = e_1'; S'}[\text{RC-WRITE-ARG}]$$

Figure 2: Field write semantics.

the OOG, and the mapping has the following properties. Every runtime object has as a unique representative abstract object in the OOG. Every runtime edge between two runtime objects has a corresponding abstract edge between the representatives of the two objects. For a runtime dataflow edge that refers to a runtime object, the corresponding abstract dataflow edge refers to the representative of the runtime object, which can also be a flow object.

3 Formalization

3.1 Abstract Syntax

We formally describe our static analysis using Featherweight Domain Java (FDJ), which models a core of the Java language with ownership domain annotations [2]. To keep the language simple and easier to reason about, FDJ uses Featherweight Java, which ignores Java language constructs such as interfaces and static code.

We adopt the FDJ abstract syntax (Fig. 12) but with the following changes. We exclude cast expressions and domain links, which are part of FDJ, but not crucial to our discussion. We also include a field write expression $e.f = e'$, which can lead to dataflow communication. (Fig. 2)

In FDJ, C ranges over class names; T ranges over types; f ranges over field names; v ranges over values; d ranges over domain names; e ranges over expressions; x ranges over variable names; n ranges over values and variable names; S ranges over stores; ℓ and θ ranges over locations in a store; θ represents the value of `this`; a store S maps locations ℓ to their contents; the set of variables includes the distinguished variable `this` of type T_{this} used to refer to the receiver of a method; the result of the computation is a location ℓ, which is sometimes referred to as a value v; $S[\ell]$ denotes the store entry of ℓ; $S[\ell, i]$ denotes the value of i^{th} field of $S[\ell]$; $S[\ell \mapsto C\text{<}\overline{\ell'.d}\text{>}(\overline{v})]$ denotes adding an entry for location ℓ to S; α and β range over formal domain parameters; m ranges over method names; p ranges over formal domain parameters, actual domains,

$$
\begin{array}{rcl}
CT & ::= & \overline{cdef} \\
cdef & ::= & \texttt{class}\ C\texttt{<}\overline{\alpha},\overline{\beta}\texttt{>}\ \texttt{extends}\ C'\texttt{<}\overline{\alpha}\texttt{>} \\
 & & \{\ \overline{dom};\ \ \overline{T}\ \overline{f};\ \ C(\overline{T'}\ \overline{f'},\overline{T}\ \overline{f}) \\
 & & \{\texttt{super}(f');\texttt{this}.\overline{f}=\overline{f};\}\ \ \overline{md}\ \} \\
dom & ::= & [\texttt{public}]\ \texttt{domain}\ d; \\
md & ::= & T_R\ m(\overline{T}\ \overline{x})\ \ T_{this}\ \{\texttt{return}\ e_R;\} \\
e & ::= & x\ \mid \texttt{new}\ C\texttt{<}\overline{p}\texttt{>}(\overline{e})\ \mid e.f\ \mid e.f=e' \\
 & & \mid e.m(\overline{e})\ \mid \ell\ \mid \ell \triangleright e \\
n & ::= & x\ \mid\ v \\
p & ::= & \alpha\ \mid\ n.d\ \mid\ \texttt{SHARED} \\
T & ::= & C\texttt{<}\overline{p}\texttt{>} \\
v,\ell,\theta & \in & \mathit{locations} \\
S & ::= & \ell\ \rightarrow\ C\texttt{<}\overline{\ell'.d}\texttt{>}(\overline{v}) \\
\Sigma & ::= & \ell\ \rightarrow\ T \\
\Gamma & ::= & x\ \rightarrow\ T
\end{array}
$$

Figure 3: Simplified FDJ abstract syntax [2].

or the special domain `SHARED`; the expression form $\ell \triangleright e$ represents a method body e executing with a receiver ℓ; an overbar denotes a sequence; the fixed class table CT maps classes to their definitions; a program is a tuple (CT, e) of a class table and an expression; Γ is the typing context; and Σ is the store typing.

3.2 Data Type Declarations

The analysis extracts a hierarchical object graph (OGraph) with nodes that represent abstract objects (OObjects) and group of objects (ODomains), and edges that represent dataflow communication between abstract objects (Fig. 13). The OGraph is a triplet $G = \langle DO, DD, DE \rangle$, where DO is a set of OObjects, DD maps a pair $(O, C{::}d)$ to an ODomain D, and DE is a set of dataflow edges. Each OEdge is a directed edge from a source O_{src} to a destination O_{dst}. The label of an OEdge is the OObject that the dataflow refers to. The flag distinguishes between OEdges that represent import or an export dataflow communication. The OGraph is a multi-graph, where multiple edges with different labels might exists between the same source and destination.

The analysis distinguishes between different instances of the same class C that are in different domains, even if created at the same `new` expression in the program. In addition, the analysis treats an instance of class C with actual parameters $\overline{p}$ differently from another instance that has actual parameters $\overline{p'}$. Hence, the data type of an OObject uses $C\texttt{<}\overline{D}\texttt{>}$. We follow the FDJ convention and consider an OObject's owning ODomain as the first element D_1 of $\overline{D}$. Our analysis relies on the precision about aliasing that Ownership Domains offer, and avoids merging object excessively. The Ownership Domains type system guarantees that two objects in different domains cannot alias. Our analysis only merges two objects of the same class if all their domains are the same. The context Υ records the combination of class and domain parameters $C\texttt{<}\overline{D}\texttt{>}$

$$
\begin{array}{ll}
G \in \mathsf{OGraph} & ::= \langle\ \mathbf{Objects} = DO,\ \mathbf{DomainMap} = DD, \mathbf{Edges} = DE\ \rangle \\
D \in \mathsf{ODomain} & ::= \langle\ \mathbf{Id} = D_{id},\ \mathbf{Domain} = C{::}d\ \rangle \\
O \in \mathsf{OObject} & ::= \langle\ \mathbf{Type} = C{<}\overline{D}{>}\ \rangle \\
E \in \mathsf{OEdge} & ::= \langle\ \mathbf{From} = O_{src},\ \mathbf{To} = O_{dst}, \mathbf{Label} = O_{label}, \mathbf{Flag} = \mathit{Imp}\ |\mathit{Exp}\ \rangle \\
DD & ::= \emptyset\ \mid\ DD \cup \{\ (O, C{::}d) \mapsto D\ \} \cup \{\ (O, C{::}\alpha) \mapsto D\ \} \\
DO & ::= \emptyset\ \mid\ DO \cup \{\ O\ \} \\
DE & ::= \emptyset\ \mid\ DE \cup \{\ E\ \} \\
\Upsilon & ::= \emptyset\ \mid\ \Upsilon \cup \{\ C{<}\overline{D}{>}\ \} \\
H & ::= \emptyset\ \mid\ H \cup \{\ \ell \mapsto O\ \} \\
K & ::= \emptyset\ \mid\ K \cup \{\ \ell.d \mapsto D\ \} \\
L_I & ::= \emptyset\ \mid\ L_I \cup \{\ (\ell_{src}, \ell_{dst}) \mapsto \{E\}\} \\
L_E & ::= \emptyset\ \mid\ L_E \cup \{\ (\ell_{src}, \ell_{dst}) \mapsto \{E\}\}
\end{array}
$$

Figure 4: Data type declarations for the OGraph.

analyzed, to avoid non-termination of the analysis.

In addition to the OEdges that have source and destination OObjects, the OGraph has ownership edges. The OGraph representation is well-formed with respect to the ownership relations declared in the code using the annotations. The data type declaration of the OGraph captures this hierarchy using the DD map without defining directly a set of ownership edges. An ownership edge states that an OObject $O = \langle C{<}\overline{D}{>}\rangle$ is a child of D_1, or that O owns a domain D. Given a mapping $\{(O, C'{::}d) \mapsto D\}$ in DD where $C'{::}d$ is a domain declaration, D is a child of O. Since domains are inherited across classes [2], the class C of O can be a subclass of C' where d is declared. The analysis also uses DD to map formal domain parameters $C{::}\alpha$ to actual domains.

Although a domain d is declared by a class C, each runtime instance of type C gets its own runtime domain $\ell.d$. For example, if there are two distinct object locations ℓ and ℓ' of class C, then $\ell.d$ and $\ell'.d$ are distinct. Since an ODomain represents a runtime domain $\ell_i.d_i$, one domain declaration d in the code can create multiple ODomains D_i in the OGraph and the fresh identifier D_{id} ensures that multiple ODomains can be created for the same domain declaration $C{::}d$. Since no class declares the `SHARED` domain, we qualify it as `::SHARED`.

During initialization, the analysis creates a global ODomain $D_{\texttt{shared}}$, the root of the OGraph. A developer picks a root class, C_{root}, and the analysis creates O_{root} in $D_{\texttt{shared}}$. The analysis also requires an initial context. We use a dummy OObject O_{world}, which does not correspond to an actual runtime object. Next, the analysis changes the context from O_{world} to O_{root}, and continues recursively with all the expressions in

the methods of C_{root}.

Instrumentation. The maps H, K, L_I, and L_E are part of the instrumented dynamic semantics (Fig. 13). H maps a location ℓ to the corresponding OObject, and K maps a runtime domain $\ell.d$ to an ODomain. The multi-valued maps L_I and L_E map a pair of locations (ℓ_{src}, ℓ_{dst}) to a set of OEdges $\{E\}$. We use two maps for edges because a pair $(H[\ell_1], H[\ell_2])$ can be associated with an import edge from $H[\ell_1]$ to $H[\ell_2]$, or with an export edge from $H[\ell_1]$ to $H[\ell_2]$.

Notation. For a map M, a key k, and a value v, we use $M[k]$ to denote the lookup of k, and $M' = M[k \mapsto v]$ for adding an entry for k to M. For a multi-valued map M, we use the notation $M' = M[k \mapsto_{\cup} \{v\}]$ for adding an entry for k to M. If the map already has an entry for k, the resulting value is the union of the existing value set and $\{v\}$.

Static Semantics. We formalize our static analysis using a constraint-based specification, as a set of inference rules, then prove that the OGraph is sound, i.e., it has all the required OObjects, ODomains, and OEdges.

In this context, soundness means that we can build a map between a ROG and an OGraph. Soundness consists of object soundness and edge soundness. With object soundness, every runtime object maps to a unique representative OObject in the OGraph. With object soundness, every runtime edge maps to a unique representative OEdge in the OGraph. To build the maps, we instrument the FDJ dynamic semantics. We map every newly created runtime object to an OObject. Also, for every field read, field write, or a method invocation, we map the corresponding runtime edge to an OEdge.

In FDJ, a program is a tuple (CT, e) that consists of a class table CT, which maps classes to their definitions, and an expression e. Our analysis starts with a root expression e_{root}, that explicitly instantiates the root class C_{root}. The analysis result is the least solution $G = \langle DO, DD, DE \rangle$ of the following constraint system:

$$\emptyset, \emptyset, G \vdash (CT, e_{root})$$

The analysis starts by creating the OObject O_{world} and its owning ODomain $D_{\texttt{SHARED}}$, which constitutes the root of the OGraph,

$$D_{\texttt{SHARED}} = \langle D_0, ::\texttt{SHARED} \rangle \qquad\qquad O_{world} = \langle C_{dummy} < . > \rangle$$

then abstractly interprets e_{root} in the context of O_{world}:

$$\emptyset, \emptyset, G \vdash_{O_{world}} e_{root}$$

The judgement form for expressions is as follows:

$$\Gamma, \Upsilon, G \vdash_{O,H} e$$

The O subscript on the turnstile captures the context-sensitivity, and represents the context object that the analysis uses to abstractly interpret e. The H subscript is a map used by the dynamic semantics and the store typing rule in the static semantics (not shown). For readability, we omit H when not in use. $CT(C)$ and $CT(\texttt{Object})$ represent a lookup of a class C and the class `Object` in the class table, and is an implicit clause in all the static rules. (We list these clauses once at the top of Fig. 5 to avoid repetition.)

In Df-New, the analysis interprets an object allocation in the context of O. The analysis first ensures that DO contains an OObject O_C for the newly allocated object. Then, using *dparams*, *Df-New* ensures that each of the actual domain parameters p_i maps to an actual domain D_i in the context of O, where the corresponding formal domain parameter α_i maps to the same D_i but in the context of O_C. *Df-New* also ensures that the object hierarchy is created such that new ODomains are created for each domain declarations in C according to the auxiliary judgment *ddomains*. Both *dparams* and *ddomains* are recursive auxiliary judgments that consider inheritance, i.e., the domain may be declared by a class C' that C extends. The base case for the recursion is the `java.lang.Object` class (Fig. 5).

Then, Df-New uses the auxiliary judgement Aux-Dom to ensure that DD has an ODomain corresponding to each domain that C locally declares ($(O_C, C{::}d_j) \mapsto D_j$). Aux-Dom recursively includes inherited domains from base classes as well. Aux-Obj1, the base case of the recursion, deals with the class `Object`, for which Aux-Obj1 does nothing, because `Object` has no fields, domains, or methods in FDJ.

Df-New then obtains each expression e_R in each method m of C, and recursively processes e_R in the context of the new OObject O_C. To avoid infinite recursion, before Df-New analyzes e_R, it checks if the combination of the class C and actual domains $\overline{D}$ have been previously analyzed by looking for this combination in Υ. If this combination does not exist, Df-New extends Υ with the current combination. As a side note, Υ tracks previously analyzed OObjects only at the call stack level. It does not do so globally across the program because similar combinations of the same class and domain parameters can occur in different contexts, and must be analyzed separately. Finally, Df-New analyzes each argument of the constructor.

$$CT(C) = \texttt{class}\ C\langle\overline{\alpha},\overline{\beta}\rangle\ \texttt{extends}\ C'\langle\overline{\alpha}\rangle\ \{\ \overline{T}\ \overline{f};\ \overline{dom};\ \ldots;\ \overline{md};\ \} \qquad CT(\texttt{Object}) = \texttt{class Object}\langle\alpha_o\rangle\ \{\ \}$$

$$\frac{\begin{array}{c} G = \langle DO, DD, DE\rangle \quad O = C_{\texttt{this}}\langle\overline{D_O}\rangle \quad \forall i \in 1..|\overline{p}| \quad G \vdash_O D_i \in findD(C_{\texttt{this}}::p_i) \\ O_C = \langle\ C\langle\overline{D}\rangle\ \rangle \quad \{O_C\} \subseteq DO \\ G \vdash_O dparams(C, O_C) \quad \{(O_C, qual(p_i)) \mapsto D_i\} \subseteq DD \\ G \vdash_O ddomains(C, O_C) \\ \forall m \in \overline{md}\ mbody(m, C\langle\overline{p}\rangle) = (\overline{x} : \overline{T},\ e_R) \\ C\langle\overline{D}\rangle \notin \Upsilon \Longrightarrow \{\overline{x} : \overline{T},\ \texttt{this} : C\langle\overline{p}\rangle\}, \Upsilon \cup \{C\langle\overline{D}\rangle\}, G \vdash_{O_C} e_R \\ \Gamma, \Upsilon, G \vdash_O \overline{e} \end{array}}{\Gamma, \Upsilon, G \vdash_O \texttt{new}\ C\langle\overline{p}\rangle(\overline{e})}\ [\textsc{Df-New}]$$

$$\frac{\begin{array}{c} e_0 : C\langle\overline{p}\rangle \quad (T_k\ f_k) \in fieldDecls(C) \\ G \vdash_O import(C\langle\overline{p}\rangle, T_k) \\ \Gamma, \Upsilon, G \vdash_O e_0 \end{array}}{\Gamma, \Upsilon, G \vdash_O e_0.f_k}\ [\textsc{Df-Read}] \qquad \frac{\begin{array}{c} e_0 : C\langle\overline{p}\rangle \quad (T_k\ f_k) \in fields(C\langle\overline{p}\rangle) \\ e_1 : C_1\langle\overline{p''}\rangle \quad C_1\langle\overline{p''}\rangle <: T_k \\ G \vdash_O export(C\langle\overline{p}\rangle, C_1\langle\overline{p''}\rangle) \\ \Gamma, \Upsilon, G \vdash_O e_0 \quad \Gamma, \Upsilon, G \vdash_O e_1 \end{array}}{\Gamma, \Upsilon, G \vdash_O e_0.f_k = e_1}\ [\textsc{Df-Write}]$$

$$\frac{\begin{array}{c} G = \langle DO, DD, DE\rangle \\ G \vdash_O O_i \in lookup\ (T_{src}) \\ G \vdash_{O_i} O_j \in lookup\ (T_{label}) \\ \{\langle O_i, O, O_j, Imp\rangle\} \subseteq DE \end{array}}{G \vdash_O import\ (T_{src}, T_{label})}\ [\textsc{Aux-Import}] \qquad \frac{\begin{array}{c} G = \langle DO, DD, DE\rangle \\ G \vdash_O O_i \in lookup\ (T_{dst}) \\ G \vdash_O O_j \in lookup\ (T_{label}) \\ \{\langle O, O_i, O_j, Exp\rangle\} \subseteq DE \end{array}}{G \vdash_O export\ (T_{dst}, T_{label})}\ [\textsc{Aux-Export}]$$

$$\frac{\begin{array}{c} e_0 : C\langle\overline{p}\rangle \quad mtype(m, C\langle\overline{p}\rangle) = \overline{T'} \to T'_R \quad mtypeDecl(m, C) = \overline{T_f} \to T_R \\ G \vdash_O import(C\langle\overline{p}\rangle, T_R) \\ \forall k \in 1..|\overline{e}|\ e_k : T_a \quad T_a <: T'_k \quad G \vdash_O export(C\langle\overline{p}\rangle, T_a) \\ \Gamma, \Upsilon, G \vdash_O e_0 \quad \Gamma, \Upsilon, G \vdash_O \overline{e} \end{array}}{\Gamma, \Upsilon, G \vdash_O e_0.m(\overline{e})}\ [\textsc{Df-Invk}]$$

Figure 5: Static semantics.

Since our analysis distinguishes between a field initialization in a constructor and a field write, Df-New does not require dataflow edges in DE.

Df-Lookup defines the auxiliary judgement *lookup* that returns the set of the OObjects O_k in DO such that the class of O_k is C' or one of its subclasses. It also ensures that each domain D_i of O_k corresponds to D'_i, a domain associated with O in DD. The second condition increases the precision of our analysis, because *lookup* returns only a subset of all the objects of class C' or its subclasses in DO. From this subset, our analysis picks the source or destination OObjects, and finds the flow object of an OEdge. Due to subtyping, the number of actual domain parameters $\overline{p}$ is smaller than or equal to the number of actual ODomains $\overline{D}$.

The auxiliary judgements Aux-Import and Aux-Export ensure import and export edges between the context OObject O and the OObjects O_i, where O_i is the result of *lookup* (T_{src}), and *lookup* (T_{dst}), respectively. The direction of the edge is from O_i to the context O for Aux-Import, and from the context O to O_i for Aux-Export. To identify an edge's label, Aux-Export calls *lookup* in the context of O, while Aux-Import calls the second *lookup* in the context of O_i. As a result, there could be multiple edges with

$$\frac{G = \langle DO, DD, DE\rangle \qquad O_C = C\overline{<D>} \qquad \forall \alpha_j \in params(C)\ \{(O_C, C::\alpha_j) \mapsto D_j\} \subseteq DD \qquad G \vdash_O dparams(C', O_C)}{G \vdash_O dparams(C, O_C)}[\text{Aux-Alpha}]$$

$$\frac{CT(\texttt{Object}) = \texttt{class Object}\overline{<\alpha_o>}\ \ \{\ \ \}}{G \vdash_O dparams(\texttt{Object}, O_C)}[\text{Aux-Alpha1}]$$

$$\frac{G = \langle DO, DD, DE\rangle \qquad O = C\overline{<D_O>} \qquad n : C_n\overline{<p>} \qquad G \vdash_O O_i \in lookup\ (C_n\overline{<p>}) \qquad D_i = DD[(O_i, C_n::d)]}{G \vdash_O D_i \in findD\ (C::n.d)}[\text{Aux-Find-Public}]$$

$$\frac{G = \langle DO, DD, DE\rangle \qquad O = C\overline{<D_O>} \qquad D_i = DD[(O, C::d_i)]}{G \vdash_O D_i \in findD\ (C::\texttt{this}.d_i)}[\text{Aux-FindThis}]$$

$$\frac{G = \langle DO, DD, DE\rangle \qquad O = C\overline{<D_O>} \qquad D_i = DD[(O, C::\alpha_i)]}{G \vdash_O D_i \in findD\ (C::\alpha_i)}[\text{Aux-FindD}]$$

$$\frac{}{G \vdash_O D_{\texttt{SHARED}} \in findD\ (::\texttt{shared})}[\text{Aux-FindShared}]$$

$$\frac{G = \langle DO, DD, DE\rangle \qquad \forall(\texttt{domain}\ d_j) \in \overline{dom} \qquad D_j = \langle D_{id_j},\ C::d_j\rangle \qquad \{(O_C, C::d_j) \mapsto D_j\} \subseteq DD \qquad G \vdash_O ddomains(C', O_C)}{G \vdash_O ddomains(C, O_C)}[\text{Aux-Dom}]$$

$$\frac{}{G \vdash_O ddomains(\texttt{Object}, O_C)}[\text{Aux-Obj1}]$$

$$\frac{G = \langle DO, DD, DE\rangle \qquad O = C_{\texttt{this}}\overline{<D>} \qquad O_k \in DO \qquad O_k = \langle C\overline{<D>}\ \rangle \qquad C <: C' \qquad \forall i \in 1..|\overline{p'}| \qquad G \vdash_O D'_i \in findD(C_{\texttt{this}}::p'_i) \qquad D'_i = D_i}{G \vdash_O O_k \in lookup\ (C'\overline{<p'>})}[\text{Aux-Lookup}]$$

Figure 6: Auxiliary judgments for static semantics.

different labels between the same two OObjects, depending on what *lookup* returns.

The auxiliary judgement Aux-Export ensures that export edges exist between the context OObject O and each of the OObjects O_i that *lookup* (T_{src}) returns. The auxiliary judgment *lookup* invoked using the T_{label} argument returns the set of label OObjects O_j. As a result, there could be multiple edges with different labels between the same two OObjects, depending on what *lookup* returns. Aux-Import is similar to Aux-Export, but the edge has an opposite direction from O_i to the context O. Another difference is that Aux-Import invokes the second *lookup* in the context of O_i rather than O because the imported object exists in the context of the receiver O_i.

Df-Read and Df-Write abstractly interpret field read and field write expressions, and use Aux-Import

$$\frac{}{\Gamma, \Upsilon, G \vdash_O x}[\textsc{Df-Var}] \qquad \frac{}{\Gamma, \Upsilon, G \vdash_O \ell}[\textsc{Df-Loc}] \qquad \frac{O_C = H[\ell] \qquad \Gamma, \Upsilon, G \vdash_{O_C} e}{\Gamma, \Upsilon, G \vdash_{O,H} \ell \triangleright e}[\textsc{Df-Context}]$$

$$\frac{\begin{array}{c} G = \langle DO, DD, DE \rangle \qquad \forall \ell \in dom(S), \Sigma[\ell] = C\langle \overline{p} \rangle \qquad H[\ell] = O = \langle C\langle \overline{D} \rangle \rangle \in DO \\ \forall m.\ mbody(m, C\langle \overline{p} \rangle) = (\overline{x} : \overline{T},\ e_R) \qquad \{\overline{x} : \overline{T},\ \texttt{this} : C\langle \overline{p} \rangle\}, \emptyset, G \vdash_O e_R \end{array}}{G \vdash_{CT,H} \Sigma}[\textsc{Df-Sigma}]$$

Figure 7: Static semantics (continued).

$$\frac{\Gamma; \Sigma; \theta \vdash n : C\langle \overline{p'} \rangle \qquad d \in domains(C\langle \overline{p'} \rangle)}{\Gamma; \Sigma; \theta \vdash qual(n.d) = C{::}d}[\textsc{Qual-Var}]$$

$$\frac{\Gamma; \Sigma; \theta \vdash \texttt{this} : C_{this}\langle \overline{p'} \rangle \qquad \alpha \in params(C_{this})}{\Gamma; \Sigma; \theta \vdash qual(\alpha) = C_{this}{::}\alpha}[\textsc{Qual-Param}]$$

$$\frac{}{\Gamma; \Sigma; \theta \vdash qual(\texttt{shared}) = {::}\texttt{shared}}[\textsc{Qual-Shared}]$$

Figure 8: Qualify domains rules.

and Aux-Export, respectively. Both auxiliary judgements take the type e_0 as the first argument, and pass it to *lookup* to set the source and destination OObjects. For the label, Df-Read uses the type of the field f_k, while Df-Write uses the type of the right-hand side expression e_1.

Df-Invk abstractly interprets method invocation expressions. First, it ensures the existence of import edges from the receiver of the method to the context OObject O. The labels of these import edges are the OObject returned by the method. Next, for each argument e_k, Df-Invk ensures the existence of export edges from O to the receiver of the method. The label of each export edge are the objects that arguments if the method refer to. The rule ensures export edges only for a method invocation with at least one argument.

Df-Var, and Df-Loc, and the rest of the rules complete our formalization and make the induction go through (Fig. 7). Df-Context analyzes expressions of the form $\ell \triangleright e$. The context for analyzing e changes from O to O_C, where O_C is the result of looking up the receiver ℓ in H. Finally, the induction requires an augmented store typing rule, Df-Sigma, to ensures that the method bodies have been analyzed for all the locations ℓ in the store, and that every ℓ has a corresponding OObject in DO. To denote all the objects in the store, we use the CT subscript instead of O.

Figure 8 shows the definitions we use to qualify a domain p by the class C that declares it. In the context of Γ, Σ, and θ, Qual-Var qualifies $n.d$ as $C{::}d$. This judgement also applies to the case when n is `this` and p = `this`.d. Qual-Param qualifies a formal domain parameter α as $C{::}\alpha$, where C is the class of

`this`. Since no class declares the `shared` domain, QUAL-SHARED qualifies it as `::shared`. We use these rules implicitly in the static and dynamic semantics to ensure that $(O, C{::}d) \mapsto D$ is in DD.

Dynamic Semantics. To complete the formalization, we instrumented the dynamic semantics (Fig. 9). The instrumentation extends the dynamic semantics of FDJ [2] (the common parts are highlighted), but is safe since discarding it produces exactly the FDJ dynamic semantics. The instrumented evaluation rule is of the following form:

$$\theta \vdash e; S; H; K; L_I; L_E \rightsquigarrow_G e'; S'; H'; K'; L'_I; L'_E$$

where $G = \langle DO, DD, DE \rangle$ is the statically computed object graph, and $\rightsquigarrow_G$ means that the expression e evaluates to e' in the context of θ, the value of `this`. The dynamic semantics keep G unchanged, but change the store S and the maps H, K, L_I, and L_E.

IR-NEW adds a new location ℓ to the store S, where ℓ maps to an object of type C with the specified ownership domain parameters, and the fields set to the values $\overline{v}$ passed to the constructor. The rule extends H by mapping ℓ and the OObject O_C from DO. The rule requires that each actual domains p_i passed during instantiation corresponds to an actual domain D_i of O_C. Next, the rule extends K such that for all the domains $C{::}d_j$, the pair $(O_C, C{::}d_j)$ has a corresponding D_j in DD.

IR-READ and IR-WRITE ensure that an OEdge E exists between the context OObject O and the receiver O_ℓ. They use θ and ℓ to lookup these OObjects in H. They also ensure that the edge label O_v is of a subclass of the field class C_i as return by the auxiliary judgment *irLookup*. Finally, the rules extend the maps L_I and L_E, respectively, by adding E to the set of edges associated with (ℓ, θ) in L_I, and (θ, ℓ) in L_E.

IR-INVK ensures that an import OEdge E' exists from the receiver O_ℓ to the context O, having as the edge's label an OObject of a subclass of the return class C_R. IR-INVK also ensures that an export OEdge E_k exist from O to O_ℓ for every parameter, having as edge label an OObject of a subclass of the method's parameter class C_k. The rule uses θ and ℓ to lookup O and O_ℓ in H . It extends both L_I and L_E by adding E' to the set of import edges between the locations ℓ and θ in L_I, and by adding each E_k to the set of export edges between the locations θ and ℓ in L_E.

The IR-LOOKUP auxiliary judgment returns the set of OObject found by looking up each actual domain $\ell'_i.d_i$ in K. When the method expression reduces to a value v, IR-CONTEXT propagates v outside of its method context. This rule does not affect the execution of the program.

Finally, the dynamic semantics include standard congruence rules. The congruence rules are similar to those in FDJ [2] (Fig. 10). In addition, there are two congruence rules for field-write: IRC-WRITE-

$$\frac{\boxed{\ell \notin dom(S) \qquad S' = S[\ell \mapsto C\!<\!\overline{p}\!>\!(\overline{v})]} \quad G = \langle DO, DD, DE\rangle \quad \overline{p} = \overline{\ell'.d} \quad \forall i \in 1..|\overline{\ell'.d}|\ D_i = K[\ell'_i.d_i] \quad \ell_i \in dom(H) \text{ s.t. } H[\ell_i] = O_i \quad D_i = DD[O_i, qual(\ell'_i.d_i)] \quad O_C = \langle C\!<\!\overline{D}\!>\rangle \quad O_C \in DO \quad H' = H[\ell \mapsto O_C] \quad \forall(\texttt{domain}\ d_j) \in domains(C\!<\!\overline{p}\!>) \quad D_j = DD[(O_C, C{::}d_j)] \quad K' = K[\ell.d_j \mapsto D_j]}{\theta \vdash \boxed{\texttt{new}\ C\!<\!\overline{p}\!>\!(\overline{v}); S}; H; K; L_I; L_E \rightsquigarrow_G \boxed{\ell; S'}; H'; K'; L_I; L_E}[\text{IR-NEW}]$$

$$\frac{\boxed{S[\ell] = C\!<\!\overline{p}\!>\!(\overline{v}) \qquad fields(C\!<\!\overline{p}\!>) = \overline{T}\ \overline{f}} \quad O = H[\theta] \quad O_\ell = H[\ell] \quad O_v = H[v_i] \quad T_i \in \overline{T} \quad E = \langle O_\ell, O, O_v, Imp\rangle \in DE \quad H; K; L_I; L_E \vdash O_v \in irLookup(T_i) \quad L'_I = L_I[(\ell, \theta) \mapsto_\cup \{E\}]}{\theta \vdash \boxed{\ell.f_i; S}; H; K; L_I; L_E \rightsquigarrow_G \boxed{v_i; S}; H; K; L'_I; L_E}[\text{IR-READ}]$$

$$\frac{\boxed{S[\ell] = C\!<\!\overline{p}\!>\!(\overline{v}) \qquad fields(C\!<\!\overline{p}\!>) = \overline{T}\ \overline{f}} \quad \boxed{S' = S[\ell \mapsto C\!<\!\overline{p}\!>\!([v/v_i]\overline{v})]} \quad O = H[\theta] \quad O_\ell = H[\ell] \quad O_v = H[v] \quad H; K; L_I; L_E \vdash O_v \in irLookup(T_i) \quad T_i \in \overline{T} \quad E = \langle O, O_\ell, O_v, Exp\rangle \in DE \quad L'_E = L_E[(\theta, \ell) \mapsto_\cup \{E\}]}{\theta \vdash \boxed{\ell.f_i = v; S}; H; K; L_I; L_E \rightsquigarrow_G \boxed{v; S'}; H; K; L_I; L'_E}[\text{IR-WRITE}]$$

$$\frac{\boxed{S[\ell] = C\!<\!\overline{p}\!>\!(\overline{v}) \qquad mbody(m, C\!<\!\overline{p}\!>) = (\overline{x}, e_R)} \quad O = H[\theta] \quad O_\ell = H[\ell] \quad mtype(m, C\!<\!\overline{p}\!>) = \overline{T} \to T_R \quad H; K; L_I; L_E \vdash O_r \in irLookup(T_R) \quad E' = \langle O_\ell, O, O_r, Imp\rangle \in DE \quad L'_I = L_I[(\ell, \theta) \mapsto_\cup \{E'\}] \quad \forall k \in 1..|\overline{x}|\ O_k = H[v_k] \quad H; K; L_I; L_E \vdash O_k \in irLookup(T_k) \quad T_k \in \overline{T} \quad E_k = \langle O, O_\ell, O_k, Exp\rangle \in DE \quad L'_E = L_E[(\theta, \ell) \mapsto_\cup \{E_k\}]}{\theta \vdash \boxed{\ell.m(\overline{v}); S}; H; K; L_I; L_E \rightsquigarrow_G \boxed{\ell \triangleright [\overline{v}/\overline{x}, \ell/\texttt{this}]e_R; S}; H; K; L'_I; L'_E}[\text{IR-INVK}]$$

$$\frac{}{\theta \vdash \boxed{\ell \triangleright v; S}; H; K; L_I; L_E \rightsquigarrow_G \boxed{v;\ S}; H; K; L_I; L_E}[\text{IR-CONTEXT}]$$

$$\frac{O_k \in rng(H) \quad O_k = \langle C'\!<\!\overline{D'}\!>\rangle \quad C' <: C \quad \forall i \in 1..|\overline{\ell'.d}|\ D_i = K[\ell'_i.d_i] \quad D'_i = D_i}{H; K; L_I; L_E \vdash O_k \in irLookup(C\!<\!\overline{\ell'.d}\!>)}[\text{IR-LOOKUP}]$$

Figure 9: Instrumented dynamic semantics (core rules).

RCV and IRC-WRITE-ARG. IRC-WRITE-RCV states that the receiver expression e_0 reduces to e'_0, while IRC-WRITE-ARG states that the right-hand side expression e_1 reduces to e'_1.

$$\frac{\theta \vdash e_i; S; H; K; L_I; L_E \rightsquigarrow_G e_i'; S'; H'; K'; L_I'; L_E'}{\begin{array}{c}\theta \vdash \texttt{new}\ C{<}\overline{p}{>}(v_{1..i-1}, e_i, e_{i+1..n}); S; H; K; L_I; L_E \rightsquigarrow_G \\ \texttt{new}\ C{<}\overline{p}{>}(v_{1..i-1}, e_i', e_{i+1..n}); S'; H'; K'; L_I'; L_E'\end{array}}[\text{IRC-New}]$$

$$\frac{\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'}{\begin{array}{c}\theta \vdash e_0.f_i; S; H; K; L_I; L_E \rightsquigarrow_G \\ e_0'.f_i; S'; H'; K'; L_I'; L_E'\end{array}}[\text{IRC-Read}]$$

$$\frac{\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'}{\begin{array}{c}\theta \vdash e_0.f_i = e_1; S; H; K; L_I; L_E \rightsquigarrow_G \\ e_0'.f_i = e_1; S'; H'; K'; L_I'; L_E'\end{array}}[\text{IRC-Write-Rcv}]$$

$$\frac{\theta \vdash e_1; S; H; K; L_I; L_E \rightsquigarrow_G e_1'; S'; H'; K'; L_I'; L_E'}{\begin{array}{c}\theta \vdash v.f_i = e_1; S; H; K; L_I; L_E \rightsquigarrow_G \\ v.f_i = e_1'; S'; H'; K'; L_I'; L_E'\end{array}}[\text{IRC-Write-Arg}]$$

$$\frac{\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'}{\begin{array}{c}\theta \vdash e_0.m(\overline{e}); S; H; K; L_I; L_E \rightsquigarrow_G \\ e_0'.m(\overline{e}); S'; H'; K'; L_I'; L_E'\end{array}}[\text{IRC-RecvInvk}]$$

$$\frac{\theta \vdash e_i; S; H; K; L_I; L_E \rightsquigarrow_G e_i'; S'; H'; K'; L_I'; L_E'}{\begin{array}{c}\theta \vdash v.m(v_{1..i-1}, e_i, e_{i+1..n}); S; H; K; L_I; L_E \rightsquigarrow_G \\ v.m(v_{1..i-1}, e_i', e_{i+1..n}); S'; H'; K'; L_I'; L_E'\end{array}}[\text{IRC-ArgInvk}]$$

$$\frac{\ell \vdash e; S; H; K; L_I; L_E \rightsquigarrow_G e'; S'; H'; K'; L_I'; L_E'}{\theta \vdash \ell \triangleright e; S; H; K; L_I; L_E \rightsquigarrow_G \ell \triangleright e'; S'; H'; K'; L_I'; L_E'}[\text{IRC-Context}]$$

Figure 10: Instrumented dynamic semantics (congruence rules).

3.3 Soundness

An OGraph is a *sound* approximation of a ROG, represented by a well-typed store S, if the OGraph relates to the ROG as follows:

Object soundness. There is a map H that maps each object ℓ in S to exactly one representative OObject in the OGraph. Similarly, there is a map K such that each runtime domain $\ell.d$ has exactly one representative ODomain in the OGraph.

Edge soundness. If there is a dataflow communication from an object ℓ_1 to ℓ_2 in a ROG, with their representatives OObjects O_1 and O_2 in the OGraph, then there are two maps L_I and L_E that map the pair (ℓ_1, ℓ_2) to a set of OEdges in the OGraph that represent the dataflow communication between O_1 and O_2.

To relate the dynamic and the static semantics of the analysis, we define an approximation relation (Df-Approx) between a runtime state (S,H,K,L_I,L_E) and an analysis result (DO, DD, DE). It ensures that the runtime objects, runtime domains and runtime edges are consistent with their representatives in the statically extracted OGraph.

Approximation Relation (Df-Approx).

$$
\begin{aligned}
&\forall\ \Sigma \vdash S,\quad (S, H, K, L_I, L_E) \sim (DO, DD, DE)\\
&\iff\\
&\forall \ell \in dom(S), \Sigma[\ell] = C\overline{<\ell'.d>}\\
&\implies\\
&H[\ell] = O_C = \langle C\overline{<D>}\rangle \in DO\\
&and\ \forall \ell'_j.d_j \in \overline{\ell'.d}\ K[\ell'_j.d_j] = D_j = \langle D_{id_j}, qual(\ell'_j.d_j)\rangle \in rng(DD)\\
&and\ \forall d_i \in domains(C\overline{<\ell'.d>})\\
&\qquad K[\ell.d_i] = D_i = \langle D_{id_i}, C{::}d_i\rangle\ \{(O_C, C{::}d_i) \mapsto D_i\} \in DD\\
&and\ \forall \ell_{src} \in dom(H),\ fields(\Sigma[\ell_{src}]) = \overline{T_{src}}\ \overline{f}\\
&\qquad \forall m.\ mtype(m, \Sigma[\ell_{src}]) = \overline{T} \to T_R\\
&\qquad\quad \forall T_k \in \{\overline{T_{src}}\} \cup \{T_R\}\\
&\qquad\qquad H; K; L_I; L_E \vdash O_k \in irLookup(T_k)\\
&\qquad\qquad\quad E'_k \in L_I[(\ell_{src}, \ell)]\ E'_k = \langle H[\ell_{src}], H[\ell], O_k, Imp\rangle \in DE\\
&and\ \forall \ell_{dst} \in dom(H),\ fields(\Sigma[\ell_{dst}]) = \overline{T_{dst}}\ \overline{f}\\
&\qquad \forall m.\ mtype(m, \Sigma[\ell_{dst}]) = \overline{T} \to T_R\\
&\qquad\quad \forall T_k \in \{\overline{T_{dst}}\} \cup \{\overline{T}\}\\
&\qquad\qquad H; K; L_I; L_E \vdash O_k \in irLookup(T_k)\\
&\qquad\qquad\quad E_k \in L_E[(\ell, \ell_{dst})]\ E_k = \langle H[\ell], H[\ell_{dst}], O_k, Exp\rangle \in DE
\end{aligned}
$$

Df-Approx states that given a well-typed store S of a program and an OGraph $G = \langle DO, DD, DE\rangle$ of the same program, there are maps H, K, L_I, and L_E, such that H maps each runtime object ℓ in the store to a unique OObject O_C from DO, K maps each runtime domain $\ell.d_i$ in the store to a unique ODomain D_i, and L_I and L_E map each pair of runtime objects (ℓ_{src}, ℓ) and (ℓ, ℓ_{dst}) to OEdges from DE. Df-Approx ensures the consistency of these mappings with the ownership relation, and with the dataflow communication.

The last two conditions relate runtime dataflow communication back to field reads, field writes, and method invocations that produce the corresponding import and export edges in DE. L_I maps a runtime

dataflow communication from a runtime object ℓ_{src} to another runtime object ℓ back to an import OEdge E'_k from DE. By our definition of import dataflow communication, E'_k exists in DE due to a field read or a method invocation expression that has ℓ_{src} as its receiver. The condition also ensures that the edge's label is an OObject of a subtype of T_k. T_k is the type of a field of ℓ_{src}, or the return type of a method of ℓ_{src}.

Similarly, L_E maps a runtime dataflow communication from a runtime object ℓ to another runtime object ℓ_{dst} back to an export OEdge E_k from DE. By our definition of export dataflow communication, E_k exists in DE due to a field write or a method invocation expression that has ℓ_{dst} as its receiver. The condition also ensures that the edge's label is an OObject of a subtype of T_k. T_k is the type of a field of ℓ_{dst}, or the type of a parameter of ℓ_{dst}'s methods.

Theorem: Dataflow Object Graph Soundness.

$$
\begin{aligned}
&\textit{If } G = \langle DO, DD, DE \rangle \\
&\quad G \vdash (CT, e_{root}) \\
&\quad \forall e,\ \theta_0 \vdash e; \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rightsquigarrow^*_G e; S; H; K; L_I; L_E \\
&\quad \Sigma \vdash S \\
&\textit{then } G \vdash_{CT,H} \Sigma \\
&\quad (S, H, K, L_I, L_E) \sim (DO, DD, DE)
\end{aligned}
$$

where $\rightsquigarrow^*_G$ relation is the reflexive and transitive closure of $\rightsquigarrow_G$ relation, and θ_0 is the location of the first object instantiated by e_{root}. To prove the Object Graph Soundness theorem, we prove the Dataflow Preservation and Dataflow Progress theorems, which extend the standard FDJ Preservation and Progress. The common parts are highlighted.

Theorem: Dataflow Preservation (Subject reduction).

$$
\begin{aligned}
&\textit{If } \boxed{\emptyset, \Sigma, \theta \vdash e : T} \\
&\quad \boxed{\Sigma \vdash S} \\
&\quad G = \langle DO, DD, DE \rangle \\
&\quad G \vdash_{CT,H} \Sigma \\
&\quad \emptyset, \emptyset, G \vdash_O e \\
&\quad (S, H, K, L_I, L_E) \sim (DO, DD, DE) \\
&\quad \theta \vdash \boxed{\text{e; S}}; H; K; L_I; L_E \rightsquigarrow_G \boxed{\text{e'; S'}}; H'; K'; L'_I; L'_E \\
&\textit{then } \boxed{\textit{there exists } \Sigma' \supseteq \Sigma \textit{ and } T' <: T \textit{ such that}} \\
&\quad \boxed{\emptyset, \Sigma', \theta \vdash e' : T' \textit{ and } \Sigma' \vdash S'} \\
&\quad (S', H', K', L'_I, L'_E) \sim (DO, DD, DE) \\
&\quad \emptyset, \emptyset, G \vdash_O e' \\
&\quad \textit{and } G \vdash_{CT,H} \Sigma'
\end{aligned}
$$

Theorem: Dataflow Progress.

$$
\begin{aligned}
&\textit{If } \boxed{\emptyset, \Sigma, \theta \vdash e : T} \\
&\quad \boxed{\Sigma \vdash S} \\
&\quad G = \langle DO, DD, DE \rangle \\
&\quad G \vdash_{CT,H} \Sigma \\
&\quad \emptyset, \emptyset, G \vdash_O e \\
&\quad (S, H, K, L_I, L_E) \sim (DO, DD, DE) \\
&\textit{then either } \boxed{e \textit{ is a value}} \\
&\quad \textit{or else } \theta \vdash \boxed{e; S}; H; K; L_I; L_E \rightsquigarrow_G \boxed{e'; S'}; H'; K'; L'_I; L'_E
\end{aligned}
$$

3.4 Theorem: Dataflow Preservation (Subject reduction)

If

$\boxed{\emptyset, \Sigma, \theta \vdash e : T}$

$\boxed{\Sigma \vdash S}$

$G = \langle DO, DD, DE \rangle$ $\qquad G \vdash_{CT,H} \Sigma$

$\emptyset, \emptyset, G \vdash_O e$

$(S, H, K, L_I, L_E) \sim (DO, DD, DE)$

$\theta \vdash \boxed{e; S}; H; K; L_I; L_E \rightsquigarrow_G \boxed{e'; S'}; H'; K'; L'_I; L'_E$

then

$\boxed{\textit{there exists } \Sigma' \supseteq \Sigma \textit{ and } T' <: T \textit{ such that } \emptyset, \Sigma', \theta \vdash e' : T' \textit{ and } \Sigma' \vdash S'}$

$(S', H', K', L'_I, L'_E) \sim (DO, DD, DE)$

$\emptyset, \emptyset, G \vdash_O e'$

and $G \vdash_{CT,H} \Sigma'$

The Dataflow Preservation theorem extends the FDJ Type Preservation theorem (the common parts are highlighted). Those parts are proved by induction over the derivation of the FDJ evaluation relation : $e; S \rightsquigarrow e'; S'$.

Proof: We prove preservation by induction on the instrumented evaluation relation

$$\theta \vdash e; S; H; K; L_I; L_E \rightsquigarrow_G e'; S'; H'; K'; L'_I; L'_E$$

The most interesting cases are Ir-New, Ir-Read (page 23), Ir-Write (page 25), and Ir-Invk (page 27).

Case Ir-New: e = `new` $C\langle\overline{\ell'.d}\rangle(\overline{v})$, and $e' = \ell$. We have:

$$\frac{\begin{array}{c} G = \langle DO, DD, DE \rangle \qquad O = C_{\texttt{this}}\langle\overline{DO}\rangle \qquad \forall i \in 1..|\overline{\ell'.d}| \qquad G \vdash_O D_i \in findD(C_{\texttt{this}}::\ell'.d_i) \\ O_C = \langle\, C\langle\overline{D}\rangle\, \rangle \qquad \{O_C\} \subseteq DO \\ G \vdash_O dparams(C, O_C) \qquad \{(O_C, qual(\ell'.d_i)) \mapsto D_i\} \subseteq DD \\ G \vdash_O ddomains(C, O_C) \\ \forall m \in \overline{md}\ mbody(m, C\langle\overline{\ell'.d}\rangle) = (\overline{x} : \overline{T},\ e_R) \\ C\langle\overline{D}\rangle \notin \Upsilon \Longrightarrow \{\overline{x} : \overline{T},\ \texttt{this} : C\langle\overline{\ell'.d}\rangle\}, \Upsilon \cup \{C\langle\overline{D}\rangle\}, G \vdash_{O_C} e_R \\ \Gamma, \Upsilon, G \vdash_O \overline{e} \end{array}}{\Gamma, \Upsilon, G \vdash_O \texttt{new}\ C\langle\overline{\ell'.d}\rangle(\overline{v})}\ [\textsc{Df-New}]$$

$$\frac{\begin{array}{c} \boxed{\ell \notin dom(S) \qquad S' = S[\ell \mapsto C\langle\overline{p}\rangle(\overline{v})]} \\ G = \langle DO, DD, DE \rangle \\ \overline{p} = \overline{\ell'.d} \qquad \forall i \in 1..|\overline{\ell'.d}|\ D_i = K[\ell'_i.d_i] \\ \ell_i \in dom(H) \text{ s.t. } H[\ell_i] = O_i \qquad D_i = DD[O_i, qual(\ell'_i.d_i)] \\ O_C = \langle C\langle\overline{D}\rangle\rangle \qquad O_C \in DO \qquad H' = H[\ell \mapsto O_C] \\ \forall(\texttt{domain}\ d_j) \in domains(C\langle\overline{p}\rangle) \qquad D_j = DD[(O_C, C::d_j)] \qquad K' = K[\ell.d_j \mapsto D_j] \end{array}}{\theta \vdash \boxed{\texttt{new}\ C\langle\overline{p}\rangle(\overline{v}); S}; H; K; L_I; L_E \rightsquigarrow_G \boxed{\ell; S'}; H'; K'; L_I; L_E}\ [\text{IR-New}]$$

$$\frac{\begin{array}{c} G = \langle DO, DD, DE \rangle \qquad \forall \ell \in dom(S), \Sigma[\ell] = C\langle\overline{p}\rangle \qquad H[\ell] = O = \langle C\langle\overline{D}\rangle\rangle \in DO \\ \forall m.\ mbody(m, C\langle\overline{p}\rangle) = (\overline{x} : \overline{T},\ e_R) \qquad \{\overline{x} : \overline{T},\ \texttt{this} : C\langle\overline{p}\rangle\}, \emptyset, G \vdash_O e_R \end{array}}{G \vdash_{CT,H} \Sigma}\ [\textsc{Df-Sigma}]$$

To Show:

(1) $(S', H', K', L'_I, L'_E) \sim (DO, DD, DE)$

(2) $\emptyset, \emptyset, G \vdash_O e'$

(3) $G \vdash_{CT,H'} \Sigma'$

$\theta \vdash e; S; H; K; L_I; L_E \rightsquigarrow_G e'; S'; H'; K'; L'_I; L'_E$ — By assumption

$(S, H, K, L_I, L_E) \sim (DO, DD, DE)$ — By assumption

$\forall \ell \in dom(S),\ \Sigma[\ell] = C\langle\overline{\ell'.d}\rangle$ — Since $\Sigma \vdash S$

$H[\theta] = O_C = \langle C\langle\overline{D}\rangle\rangle \in DO$

and $\forall \theta'_j.d_j \in \overline{\theta'.d}\ K[\theta'_j.d_j] = D_j = \langle D_{id_j}, qual(\theta'_j.d_j)\rangle \in rng(DD)$

and $\forall d_i \in domains(C\langle\overline{\theta'.d}\rangle)$

$K[\theta.d_i] = D_i = \langle D_{id_i}, C::d_i\rangle\ \{(O_C, C::d_i) \mapsto D_i\} \in DD$ — By Df-Approx

$\forall \ell_{src} \in dom(H),\ fields(\Sigma[\ell_{src}]) = \overline{T_{src}}\ \overline{f}$

$\forall m.\ mtype(m, \Sigma[\ell_{src}]) = \overline{T} \to T_R$

$\forall T_k \in \{\overline{T_{src}}\} \cup \{T_R\}$

$H; K; L_I; L_E \vdash O_k \in irLookup(T_k)$

$E'_k \in L_I[(\ell_{src}, \theta)]\ E'_k = \langle H[\ell_{src}], H[\theta], O_k, Imp\rangle \in DE$ — By Df-Approx

$\forall \ell_{dst} \in dom(H),\ fields(\Sigma[\ell_{dst}]) = \overline{T_{dst}}\ \overline{f}$

$\forall m.\ mtype(m, \Sigma[\ell_{dst}]) = \overline{T} \to T_R$

$\forall T_k \in \{\overline{T_{dst}}\} \cup \{\overline{T}\}$

$H; K; L_I; L_E \vdash O_k \in irLookup(T_k)$

$E_k \in L_E[(\theta, \ell_{dst})]\ E_k = \langle H[\theta], H[\ell_{dst}], O_k, Exp\rangle \in DE$ — By Df-Approx

$O_C = \langle C_\ell\langle\overline{D}\rangle\rangle \in DO$ — By sub-derivation of Ir-New

$S' = S[\ell \mapsto C_\ell\langle\overline{\ell'.d}\rangle(\overline{v})]$ — By sub-derivation of Ir-New

$H' = H[\ell \mapsto O_C]$ — By sub-derivation of Ir-New

$\forall i \in 1..|\overline{\ell'.d}|\ D_i = K[\ell'_i.d_i]$ — By sub-derivation of Ir-New

$\forall(\texttt{domain}\ d_j) \in domains(C\langle\overline{\ell'.d}\rangle) \quad D_j = DD[(O_C, C_\ell::d_j)]$

$K' = K[\ell.d_j \mapsto D_j]$ — By sub-derivation of Ir-New

$L'_I = L_I \quad L'_E = L_E$ — By sub-derivation of Ir-New

$\exists \Sigma' \supseteq \Sigma$ *and* $T' <: T$ *s.t.* $\emptyset, \Sigma', \theta \vdash e' : T'$ *and* $\Sigma' \vdash S'$ — By FDJ Type Preservation

$\Sigma'[\ell] = C_\ell\langle\overline{\ell'.d}\rangle$ — By $\Sigma' \vdash S'$

$(S', H', K', L'_I, L'_E) \sim (DO, DD, DE)$ — By Df-Approx

This proves (1).

$\emptyset, \emptyset, G \vdash_O e'$ — By Df-Loc, since $e' = \ell$

This proves (2).

$G \vdash_{CT,H} \Sigma$ — By assumption

$\forall \ell \in dom(S), \Sigma[\ell] = C_\ell\langle\overline{p}\rangle$ — By sub-derivation of Df-Sigma

$H[\ell] = O_\ell = \langle C_\ell\langle\overline{D_\ell}\rangle\rangle \in DO$ — By sub-derivation of Df-Sigma

$\forall m.\ mbody(m, C_\ell\langle\overline{p}\rangle) = (\overline{x} : \overline{T},\ e_R)$ — By sub-derivation of Df-Sigma

$\{\overline{x} : \overline{T},\ \texttt{this} : C_\ell\langle\overline{p}\rangle\}, \emptyset, G \vdash_{O_\ell} e_R$ — By sub-derivation of Df-Sigma

$O_C = \langle C\langle\overline{D}\rangle\rangle \in DO$ — By sub-derivation of Ir-New

$S' = S[\ell \mapsto C\langle\overline{p}\rangle(\overline{v})]$ — By sub-derivation of Ir-New

$H' = H[\ell \mapsto O_C]$ — By sub-derivation of Ir-New

$\emptyset, \emptyset, G \vdash_O e$ — By assumption with e, Υ below

$e = \texttt{new}\ C\langle\overline{\ell'.d}\rangle(\overline{v})$, and $\Upsilon = \emptyset$

$\forall m.\ mbody(m, C\langle\overline{p}\rangle) = (\overline{x} : \overline{T},\ e_R)$ — By sub-derivation of Df-New

$C\langle\overline{D}\rangle \notin \Upsilon \Longrightarrow$

$\{\overline{x} : \overline{T}, \texttt{this} : C\langle\overline{p}\rangle\}, \Upsilon \cup \{C\langle\overline{D}\rangle\}, G \vdash_{O_C} e_R$ — By sub-derivation of Df-New

$\{\overline{x} : \overline{T}, \texttt{this} : C\langle\overline{p}\rangle\}, \emptyset, G \vdash_{O_C} e_R$ — By Df-Strengthening Lemma

$\forall \ell \in dom(S'), \Sigma'[\ell] = C_\ell\langle\overline{p}\rangle$

$H'[\ell] = O_\ell = \langle C_\ell\langle\overline{D_\ell}\rangle\rangle \in DO$

$\forall m.\ mbody(m, C_\ell\langle\overline{p}\rangle) = (\overline{x} : \overline{T},\ e_R)$

$\{\overline{x} : \overline{T},\ \texttt{this} : C_\ell\langle\overline{p}\rangle\}, \emptyset, G \vdash_{O_\ell} e_R$ — By above

$G \vdash_{CT,H'} \Sigma'$ — By Df-Sigma with above H' and Σ'

This proves (3).

Case Ir-Read: e = $\ell.f_i$, and $e' = v_i$. We have:

$$\frac{e_0 : C\text{<}\overline{p}\text{>} \qquad (T_k\ f_k) \in fieldDecls(C) \qquad G \vdash_O import(C\text{<}\overline{p}\text{>}, T_k) \qquad \Gamma, \Upsilon, G \vdash_O e_0}{\Gamma, \Upsilon, G \vdash_O e_0.f_k}\text{[Df-Read]}$$

$$\frac{\boxed{S[\ell] = C\text{<}\overline{p}\text{>}(\overline{v}) \qquad fields(C\text{<}\overline{p}\text{>}) = \overline{T}\ \overline{f}} \qquad O = H[\theta] \qquad O_\ell = H[\ell] \qquad O_v = H[v_i] \qquad T_i \in \overline{T} \qquad E = \langle O_\ell, O, O_v, Imp\rangle \in DE \qquad H; K; L_I; L_E \vdash O_v \in irLookup(T_i) \qquad L'_I = L_I[(\ell,\theta) \mapsto_\cup \{E\}]}{\theta \vdash \boxed{\ell.f_i; S}; H; K; L_I; L_E \rightsquigarrow_G \boxed{v_i; S}; H; K; L'_I; L_E}\text{[IR-Read]}$$

$$\frac{G = \langle DO, DD, DE\rangle \qquad \forall \ell \in dom(S), \Sigma[\ell] = C\text{<}\overline{p}\text{>} \qquad H[\ell] = O = \langle C\text{<}\overline{D}\text{>}\rangle \in DO \qquad \forall m.\ mbody(m, C\text{<}\overline{p}\text{>}) = (\overline{x} : \overline{T},\ e_R) \qquad \{\overline{x} : \overline{T},\ \texttt{this} : C\text{<}\overline{p}\text{>}\}, \emptyset, G \vdash_O e_R}{G \vdash_{CT,H} \Sigma}\text{[Df-Sigma]}$$

To Show:

(1) $(S', H', K', L'_I, L'_E) \sim (DO, DD, DE)$
(2) $\emptyset, \emptyset, G \vdash_O e'$
(3) $G \vdash_{CT,H'} \Sigma'$

$\theta \vdash e; S; H; K; L_I; L_E \rightsquigarrow_G e'; S'; H'; K'; L'_I; L'_E$ — By assumption
$(S, H, K, L_I, L_E) \sim (DO, DD, DE)$ — By assumption
$\forall \ell \in dom(S),\ \Sigma[\ell] = C\text{<}\overline{\ell'.d}\text{>}$ — Since $\Sigma \vdash S$
$H[\theta] = O_C = \langle C\text{<}\overline{D}\text{>}\rangle \in DO$
and $\forall \theta'_j.d_j \in \overline{\theta'.d}\ K[\theta'_j.d_j] = D_j = \langle D_{id_j}, qual(\theta'_j.d_j)\rangle \in rng(DD)$
and $\forall d_i \in domains(C\text{<}\overline{\theta'.d}\text{>})$
$K[\theta.d_i] = D_i = \langle D_{id_i}, C::d_i\rangle\ \{(O_C, C::d_i) \mapsto D_i\} \in DD$ — By Df-Approx
$\forall \ell_{src} \in dom(H),\ fields(\Sigma[\ell_{src}]) = \overline{T_{src}}\ \overline{f}$
$\forall m.\ mtype(m, \Sigma[\ell_{src}]) = \overline{T} \to T_R$
$\forall T_k \in \{\overline{T_{src}}\} \cup \{T_R\}$
$H; K; L_I; L_E \vdash O_k \in irLookup(T_k)$
$E'_k \in L_I[(\ell_{src}, \theta)]\ E'_k = \langle H[\ell_{src}], H[\theta], O_k, Imp\rangle \in DE$ — By Df-Approx
$\forall \ell_{dst} \in dom(H),\ fields(\Sigma[\ell_{dst}]) = \overline{T_{dst}}\ \overline{f}$
$\forall m.\ mtype(m, \Sigma[\ell_{dst}]) = \overline{T} \to T_R$
$\forall T_k \in \{\overline{T_{dst}}\} \cup \{\overline{T}\}$
$H; K; L_I; L_E \vdash O_k \in irLookup(T_k)$
$E_k \in L_E[(\theta, \ell_{dst})]\ E_k = \langle H[\theta], H[\ell_{dst}], O_k, Exp\rangle \in DE$ — By Df-Approx
$S' = S, H' = H, K' = K, L'_E = L_E$ — By sub-derivation of Ir-Read
$S[\ell] = C_\ell\text{<}\overline{p}\text{>}(\overline{v}) \quad fields(C_\ell\text{<}\overline{p}\text{>}) = \overline{T}\ \overline{f}$ — By sub-derivation of Ir-Read
$O = H[\theta] \quad O_\ell = H[\ell] \quad O_v = H[v_i] \quad T_i \in \overline{T}$ — By sub-derivation of Ir-Read
$E' = \langle O_\ell, O, O_v, Imp\rangle \in DE \quad H; K; L_I; L_E \vdash O_v \in irLookup(T_i)$ — By sub-derivation of Ir-Read
$L'_I = L_I[(\ell,\theta) \mapsto_\cup \{E'\}]$ — By sub-derivation of Ir-Read

$\forall \ell_{src} \in dom(H'),\ fields(\Sigma'[\ell_{src}]) = \overline{T_{src}}\ \overline{f},$
$\forall m.\ mtype(m, \Sigma'[\ell_{src}]) = \overline{T} \to T_R$
$\forall T_k \in \{\overline{T_{src}}\} \cup \{T_R\}$
$\forall H'; K'; L'_I; L'_E \vdash O_k \in irLookup(T_k)$
$E'_k \in L'_I[(\ell_{src}, \theta)] = \langle H'[\ell_{src}], H'[\theta], O_k, Imp \rangle \in DE$ By above, since $\Sigma' = \Sigma$
$(S', H', K', L'_I, L'_E) \sim (DO, DD, DE)$ By Df-Approx
This proves (1).

$\emptyset, \emptyset, G \vdash_O e'$ By Df-Loc, since $e' = v_i$
This proves (2).

$G \vdash_{CT,H} \Sigma$ By assumption
$S' = S, H' = H$ By sub-derivation of Ir-Read
$G \vdash_{CT,H'} \Sigma'$ By Df-Sigma with the above H' and $\Sigma' = \Sigma$
This proves (3).

Case Ir-Write: e $= \ell.f_i = v$, and $e' = v$

We have:

$$\frac{\begin{array}{c} e_0 : C\!<\!\overline{p}\!> \qquad (T_k\ f_k) \in fields(C\!<\!\overline{p}\!>) \\ e_1 : C_1\!<\!\overline{p''}\!> \qquad C_1\!<\!\overline{p''}\!> <: T_k \\ G \vdash_O export(C\!<\!\overline{p}\!>, C_1\!<\!\overline{p''}\!>) \\ \Gamma, \Upsilon, G \vdash_O e_0 \qquad \Gamma, \Upsilon, G \vdash_O e_1 \end{array}}{\Gamma, \Upsilon, G \vdash_O e_0.f_k = e_1}\text{[Df-Write]}$$

$$\frac{\boxed{S[\ell] = C\!<\!\overline{p}\!>(\overline{v}) \qquad fields(C\!<\!\overline{p}\!>) = \overline{T}\ \overline{f}} \quad \boxed{S' = S[\ell \mapsto C\!<\!\overline{p}\!>([v/v_i]\overline{v})]}}{\begin{array}{c} O = H[\theta] \qquad O_\ell = H[\ell] \qquad O_v = H[v] \qquad H; K; L_I; L_E \vdash O_v \in irLookup(T_i) \qquad T_i \in \overline{T} \\ E = \langle O, O_\ell, O_v, Exp\rangle \in DE \qquad L'_E = L_E[(\theta, \ell) \mapsto_\cup \{E\}] \end{array}}{\theta \vdash \boxed{\ell.f_i = v; S}; H; K; L_I; L_E \rightsquigarrow_G \boxed{v; S'}; H; K; L_I; L'_E}\text{[IR-Write]}$$

$$\frac{\begin{array}{c} G = \langle DO, DD, DE\rangle \qquad \forall \ell \in dom(S), \Sigma[\ell] = C\!<\!\overline{p}\!> \qquad H[\ell] = O = \langle C\!<\!\overline{D}\!>\rangle \in DO \\ \forall m.\ mbody(m, C\!<\!\overline{p}\!>) = (\overline{x} : \overline{T},\ e_R) \qquad \{\overline{x} : \overline{T},\ \texttt{this} : C\!<\!\overline{p}\!>\}, \emptyset, G \vdash_O e_R \end{array}}{G \vdash_{CT,H} \Sigma}\text{[Df-Sigma]}$$

To Show:

(1) $(S', H', K', L'_I, L'_E) \sim (DO, DD, DE)$

(2) $\emptyset, \emptyset, G \vdash_O e'$

(3) $G \vdash_{CT,H'} \Sigma'$

$\theta \vdash e; S; H; K; L_I; L_E \rightsquigarrow_G e'; S'; H'; K'; L'_I; L'_E$ — By assumption

$(S, H, K, L_I, L_E) \sim (DO, DD, DE)$ — By assumption

$\forall \ell \in dom(S),\ \Sigma[\ell] = C\!<\!\overline{\ell'.d}\!>$ — Since $\Sigma \vdash S$

$H[\theta] = O_C = \langle C\!<\!\overline{D}\!>\rangle \in DO$

and $\forall \theta'_j.d_j \in \overline{\theta'.d}\ K[\theta'_j.d_j] = D_j = \langle D_{id_j}, qual(\theta'_j.d_j)\rangle \in rng(DD)$

and $\forall d_i \in domains(C\!<\!\overline{\theta'.d}\!>)$

$K[\theta.d_i] = D_i = \langle D_{id_i}, C{::}d_i\rangle\ \{(O_C, C{::}d_i) \mapsto D_i\} \in DD$ — By Df-Approx

$\forall \ell_{src} \in dom(H),\ fields(\Sigma[\ell_{src}]) = \overline{T_{src}}\ \overline{f}$

$\forall m.\ mtype(m, \Sigma[\ell_{src}]) = \overline{T} \to T_R$

$\forall T_k \in \{\overline{T_{src}}\} \cup \{T_R\}$

$H; K; L_I; L_E \vdash O_k \in irLookup(T_k)$

$E'_k \in L_I[(\ell_{src}, \theta)]\ E'_k = \langle H[\ell_{src}], H[\theta], O_k, Imp\rangle \in DE$ — By Df-Approx

$\forall \ell_{dst} \in dom(H),\ fields(\Sigma[\ell_{dst}]) = \overline{T_{dst}}\ \overline{f}$

$\forall m.\ mtype(m, \Sigma[\ell_{dst}]) = \overline{T} \to T_R$

$\forall T_k \in \{\overline{T_{dst}}\} \cup \{\overline{T}\}$

$H; K; L_I; L_E \vdash O_k \in irLookup(T_k)$

$E_k \in L_E[(\theta, \ell_{dst})]\ E_k = \langle H[\theta], H[\ell_{dst}], O_k, Exp\rangle \in DE$ — By Df-Approx

$H' = H, K' = K, L'_I = L_I$ — By sub-derivation of Ir-Write

$S[\ell] = C_\ell\langle\overline{p}\rangle(\overline{v}) \quad fields(C_\ell\langle\overline{p}\rangle) = \overline{T}\ \overline{f}$ — By sub-derivation of Ir-Write

$S' = S[\ell \mapsto C_\ell\langle\overline{p}\rangle([v/v_i]\overline{v})]$ — By sub-derivation of Ir-Write

$O = H[\theta] \quad O_\ell = H[\ell] \quad O_v = H[v] \quad T_i \in \overline{T}$ — By sub-derivation of Ir-Write

$E = \langle O, O_\ell, O_v, Exp\rangle \in DE \quad H; K; L_I; L_E \vdash O_v \in irLookup(T_i)$ — By sub-derivation of Ir-Write

$L'_E = L_E[(\theta, \ell) \mapsto_\cup \{E\}]$ — By sub-derivation of Ir-Write

$\exists \Sigma' \supseteq \Sigma \ \ and\ T' <: T\ s.t.\ \emptyset, \Sigma', \theta \vdash e' : T'\ and\ \Sigma' \vdash S'$ — By FDJ Type Preservation

$\Sigma'[\ell] = C\langle\overline{\ell'.d}\rangle$ — $\Sigma' \vdash S'$

$\forall \ell_{dst} \in dom(H'),\ fields(\Sigma'[\ell_{dst}]) = \overline{T_{dst}}\ \overline{f},$

$\quad \forall m.\ mtype(m, \Sigma'[\ell_{dst}]) = \overline{T} \to T_R$

$\quad\quad \forall T_k \in \{\overline{T_{dst}}\} \cup \{\overline{T}\}$

$\quad\quad\quad \forall O_k. H'; K'; L'_I; L'_E \vdash O_k \in irLookup(T_k)$

$\quad\quad\quad\quad E_k \in L'_E[(\theta, \ell_{dst})] = \langle H'[\theta], H'[\ell_{dst}], O_k, Exp\rangle \in DE$ — By above

$(S', H', K', L'_I, L'_E) \sim (DO, DD, DE)$ — By Df-Approx

This proves (1).

$\emptyset, \emptyset, G \vdash_O e'$ — By Df-Loc, since $e' = v_i$

This proves (2).

$G \vdash_{CT,H} \Sigma$ — By assumption

$\forall \ell \in dom(S), \Sigma[\ell] = C_\ell\langle\overline{p}\rangle$

$\quad H[\ell] = O_\ell = \langle C_\ell\langle\overline{D_\ell}\rangle\rangle \in DO$

$\quad \forall m.\ mbody(m, C_\ell\langle\overline{p}\rangle) = (\overline{x} : \overline{T},\ e_R)$

$\quad\quad \{\overline{x} : \overline{T},\ \texttt{this} : C_\ell\langle\overline{p}\rangle\}, \emptyset, G \vdash_{O_\ell} e_R$ — By sub-derivation of Df-Sigma

$H' = H$ — By sub-derivation of Ir-Write

$S[\ell] = C_\ell\langle\overline{p}\rangle(\overline{v}) \quad fields(C_\ell\langle\overline{p}\rangle) = \overline{T}\ \overline{f}$ — By sub-derivation of Ir-Write

$S' = S[\ell \mapsto C_\ell\langle\overline{(}\rangle[v/v_i]\overline{v})]$ — By sub-derivation of Ir-Write

$G \vdash_{CT,H'} \Sigma'$ — By Df-Sigma with the above H' and $\Sigma' = \Sigma$

This proves (3).

Case Ir-Invk: e $= \ell.m(\overline{v})$, and $e' = \ell \triangleright [\overline{v}/\overline{x}, \ell/\texttt{this}]e_R$

We have:

$$\frac{\begin{array}{c}\ell : C\langle\overline{\ell'.d}\rangle \qquad mtype(m, C\langle\overline{\ell'.d}\rangle) = \overline{T'} \to T'_R \qquad mtypeDecl(m, C) = \overline{T_f} \to T_R \\ G \vdash_O import(C\langle\overline{\ell'.d}\rangle, T_R) \\ \forall k \in 1..|\overline{v}|\ v_k : T_a \qquad T_a <: T'_k \qquad G \vdash_O export(C\langle\overline{\ell'.d}\rangle, T_a) \\ \Gamma, \Upsilon, G \vdash_O \ell \qquad \Gamma, \Upsilon, G \vdash_O \overline{v}\end{array}}{\Gamma, \Upsilon, G \vdash_O \ell.m(\overline{v})}\text{[Df-Invk]}$$

$$\frac{\begin{array}{c}\boxed{S[\ell] = C\langle\overline{p}\rangle(\overline{v}) \qquad mbody(m, C\langle\overline{p}\rangle) = (\overline{x}, e_R)} \\ O = H[\theta] \qquad O_\ell = H[\ell] \qquad mtype(m, C\langle\overline{p}\rangle) = \overline{T} \to T_R \\ H; K; L_I; L_E \vdash O_r \in irLookup(T_R) \qquad E' = \langle O_\ell, O, O_r, Imp\rangle \in DE \qquad L'_I = L_I[(\ell, \theta) \mapsto_\cup \{E'\}] \\ \forall k \in 1..|\overline{x}|\ O_k = H[v_k] \qquad H; K; L_I; L_E \vdash O_k \in irLookup(T_k) \qquad T_k \in \overline{T} \\ E_k = \langle O, O_\ell, O_k, Exp\rangle \in DE \qquad L'_E = L_E[(\theta, \ell) \mapsto_\cup \{E_k\}]\end{array}}{\theta \vdash \boxed{\ell.m(\overline{v}); S}; H; K; L_I; L_E \rightsquigarrow_G \boxed{\ell \triangleright [\overline{v}/\overline{x}, \ell/\texttt{this}]e_R; S}; H; K; L'_I; L'_E}\text{[IR-Invk]}$$

$$\frac{\begin{array}{c}G = \langle DO, DD, DE\rangle \qquad \forall \ell \in dom(S), \Sigma[\ell] = C\langle\overline{p}\rangle \qquad H[\ell] = O = \langle C\langle\overline{D}\rangle\rangle \in DO \\ \forall m.\ mbody(m, C\langle\overline{p}\rangle) = (\overline{x} : \overline{T},\ e_R) \qquad \{\overline{x} : \overline{T},\ \texttt{this} : C\langle\overline{p}\rangle\}, \emptyset, G \vdash_O e_R\end{array}}{G \vdash_{CT,H} \Sigma}\text{[Df-Sigma]}$$

To Show:

(1) $(S', H', K', L'_I, L'_E) \sim (DO, DD, DE)$

(2) $\emptyset, \emptyset, G \vdash_O e'$

(3) $G \vdash_{CT,H'} \Sigma'$

$\theta \vdash e; S; H; K; L_I; L_E \rightsquigarrow_G e'; S'; H'; K'; L'_I; L'_E$ — By assumption

$(S, H, K, L_I, L_E) \sim (DO, DD, DE)$ — By assumption

$\forall \ell \in dom(S),\ \Sigma[\ell] = C\langle\overline{\ell'.d}\rangle$ — Since $\Sigma \vdash S$

$H[\theta] = O_C = \langle C\langle\overline{D}\rangle\rangle \in DO$

and $\forall \theta'_j.d_j \in \overline{\theta'.d}\ K[\theta'_j.d_j] = D_j = \langle D_{id_j}, qual(\theta'_j.d_j)\rangle \in rng(DD)$

and $\forall d_i \in domains(C\langle\overline{\theta'.d}\rangle)$

$K[\theta.d_i] = D_i = \langle D_{id_i}, C::d_i\rangle\ \{(O_C, C::d_i) \mapsto D_i\} \in DD$ — By Df-Approx

$\forall \ell_{src} \in dom(H),\ fields(\Sigma[\ell_{src}]) = \overline{T_{src}}\ \overline{f}$

$\forall m.\ mtype(m, \Sigma[\ell_{src}]) = \overline{T} \to T_R$

$\forall T_k \in \{\overline{T_{src}}\} \cup \{T_R\}$

$H; K; L_I; L_E \vdash O_k \in irLookup(T_k)$

$E'_k \in L_I[(\ell_{src}, \theta)]\ E'_k = \langle H[\ell_{src}], H[\theta], O_k, Imp\rangle \in DE$ — By Df-Approx

$\forall \ell_{dst} \in dom(H),\ fields(\Sigma[\ell_{dst}]) = \overline{T_{dst}}\ \overline{f}$

$\forall m.\ mtype(m, \Sigma[\ell_{dst}]) = \overline{T} \to T_R$

$\forall T_k \in \{\overline{T_{dst}}\} \cup \{\overline{T}\}$

$H; K; L_I; L_E \vdash O_k \in irLookup(T_k)$

$E_k \in L_E[(\theta, \ell_{dst})]\ E_k = \langle H[\theta], H[\ell_{dst}], O_k, Exp\rangle \in DE$ — By Df-Approx

$S' = S \quad H' = H \quad K' = K$ By sub-derivation of Ir-Invk

$S[\ell] = C_\ell\text{<}\overline{p}\text{>}(\overline{v}) \quad mbody(m, C_\ell\text{<}\overline{p}\text{>}) = (\overline{x}, e_R)$ By sub-derivation of Ir-Invk

$H[\theta] = O \quad H[\ell] = O_\ell$ By sub-derivation of Ir-Invk

$mtype(m, C_\ell\text{<}\overline{p}\text{>}) = \overline{T} \to T_R \quad T_R = C_R\text{<}\overline{p'}\text{>}$ By sub-derivation of Ir-Invk

$\forall O_r.H, K, L_I, L_E \vdash O_r \in irLookup(T_R)\ E' = \langle O_\ell, O, O_r, Imp\rangle \in DE$ By sub-derivation of Ir-Invk

$L'_I = L_I[(\ell, \theta) \mapsto_\cup \{E'\}]$ By sub-derivation of Ir-Invk

$\forall k \in 1..|\overline{x}|\ O_k = H[v_k] \quad H; K; L_I; L_E \vdash O_k \in irLookup(T_k) \quad T_k \in \overline{T}$ By sub-derivation of Ir-Invk

$E_k = \langle O, O_\ell, O_k, Exp\rangle \in DE \quad L'_E = L_E[(\theta, \ell) \mapsto_\cup \{E_k\}]$ By sub-derivation of Ir-Invk

$\exists \Sigma' \supseteq \Sigma$ *and* $T' <: T$ *s.t.* $\emptyset, \Sigma', \theta \vdash e' : T'$ *and* $\Sigma' \vdash S'$ By FDJ Type Preservation

$\Sigma'[\ell] = C_\ell\text{<}\overline{\ell'.d}\text{>}$ $\Sigma' \vdash S'$

$\forall \ell_{src} \in dom(H'),\ fields(\Sigma'[\ell_{src}]) = \overline{T_{src}}\ \overline{f},$

$\forall m.\ mtype(m, \Sigma'[\ell_{src}]) = \overline{T} \to T_R$

$\forall T_k \in \{\overline{T_{src}}\} \cup \{T_R\}$

$\forall O_k.H'; K'; L'_I; L'_E \vdash O_k \in irLookup(T_k)$

$E'_k \in L'_I[(\ell_{src}, \theta)] = \langle H'[\ell_{src}], H'[\theta], O_k, Imp\rangle \in DE$ By above

$\forall \ell_{dst} \in dom(H'),\ fields(\Sigma'[\ell_{dst}]) = \overline{T_{dst}}\ \overline{f},$

$\forall m.\ mtype(m, \Sigma'[\ell_{dst}]) = \overline{T} \to T_R$

$\forall T_k \in \{\overline{T_{dst}}\} \cup \{\overline{T}\}$

$\forall O_k.H'; K'; L'_I; L'_E \vdash O_k \in irLookup(T_k)$

$E_k \in L'_E[(\theta, \ell_{dst})] = \langle H'[\theta], H'[\ell_{dst}], O_k, Exp\rangle \in DE$ By above

$(S', H', K', L'_I, L'_E) \sim (DO, DD, DE)$ By Df-Approx

This proves (1).

$\emptyset, \emptyset, G \vdash_O e$ By assumption

$e = \ell.m(\overline{v}) \quad e_0 = \ell \quad \overline{e} = \overline{v}$ By assumption

$e' = \ell \triangleright [\overline{v}/\overline{x}, \ell/\texttt{this}]e_R$ By assumption

$\emptyset, \Sigma, \theta \vdash e : T$ By assumption

$\exists \Sigma' \supseteq \Sigma$ *and* $T' <: T$ *s.t.* $\emptyset, \Sigma', \theta \vdash e' : T'$ *and* $\Sigma' \vdash S'$ By FDJ Type Preservation

$e_0 : T_0 \quad T_0 = C_\ell\text{<}\overline{p}\text{>}$ By sub-derivation of Df-Invk

$mtype(m, C_\ell\text{<}\overline{p}\text{>}) = \overline{T} \to T_R$ By sub-derivation of Df-Invk

$\emptyset, \emptyset, G \vdash_O e_0$ By sub-derivation of Df-Invk

$\emptyset, \emptyset, G \vdash_O \overline{e}$ By sub-derivation of Df-Invk

$\{\overline{x} : \overline{T}, \texttt{this} : C_\ell\text{<}\alpha, \overline{\beta}\text{>}\}, \Sigma, \theta \vdash e_R : T_R \quad T_R <: T$ By FDJ *Meth*OK:

$S[\ell] = C_\ell\text{<}d, \overline{d'}\text{>}(\overline{v})$ By sub-derivation of Ir-Invk

$mbody(m, C_\ell\text{<}d, \overline{d'}\text{>}) = (\overline{x}, e_R)$ By sub-derivation of Ir-Invk

$\Sigma[\ell] = C_\ell\text{<}d, \overline{d'}\text{>} = T_0$ Since $e_0 = \ell$, by T-Store

$e_0 : C_\ell\text{<}d, \overline{d'}\text{>}$ Since $e_0 = \ell$, by T-Store

$mtype(m, C_\ell\text{<}d, \overline{d'}\text{>}) = \overline{T} \to T_R$ Since $e_0 = \ell$, by T-Store

$\overline{v} : \overline{T_a}$ By inversion

$\overline{T_a} <: [\overline{v}/\overline{x}, \ell/\texttt{this}]\overline{T}$ For some $\overline{T_a}$ and $\overline{T}$

there are some $D\text{<}\overline{d}\text{>}$ and T'_R so that: By Method Lemma

$T'_R <: T_R$ and $C_\ell\text{<}d, \overline{d'}\text{>} <: D\text{<}\overline{d}\text{>}$ By Method Lemma

so that $\{\overline{x} : \overline{T}, \texttt{this} : D\text{<}\overline{d}\text{>}\}, \Sigma, \theta \vdash e_R : T'_R$ By Method Lemma

there exists $T_S, T_S <: T'_R$ s. t. $[\overline{v}/\overline{x}, \ell/\texttt{this}]e_R : T_S$ Since term substitution preserves typing

$T_S <: T'_R$ and $T'_R <: T_R$ By above

$T_S <: T_R$ By transitivity of $<:$

Take $T = T' = T_R$ in FDJ Preservation

$\{\overline{x} : \overline{T}, \texttt{this} : C_\ell\!<\!d, \overline{d'}\!>\}, \emptyset, G \vdash_{O_C} e_R$ — By Df-Sigma

$O_C = H[\ell]$ — By Df-Sigma

$\emptyset, \emptyset, G \vdash_O \ell$ — By Df-Loc

$\emptyset, \emptyset, G \vdash_{O_C} [\overline{v}/\overline{x}, \ell/\texttt{this}]e_R$ — By Df-Substitution Lemma

$\emptyset, \emptyset, G \vdash_O \ell \triangleright [\overline{v}/\overline{x}, \ell/\texttt{this}]e_R$ — By Df-Context

This proves (2).

$G \vdash_{CT,H} \Sigma$ — By assumption

$S' = S, H' = H$ — By sub-derivation of Ir-Invk

$G \vdash_{CT,H'} \Sigma'$ — By Df-Sigma with the above $H' = H$ and $\Sigma' = \Sigma$

This proves (3).

Case Ir-Context: e $= \ell \triangleright v$, and $e' = v$

We have:

$$\frac{O_C = H[\ell] \qquad \Gamma, \Upsilon, G \vdash_{O_C} e}{\Gamma, \Upsilon, G \vdash_{O,H} \ell \triangleright e}[\text{Df-Context}]$$

$$\frac{}{\theta \vdash \boxed{\ell \triangleright v; S}; H; K; L_I; L_E \rightsquigarrow_G \boxed{v;\ S}; H; K; L_I; L_E}[\text{IR-Context}]$$

$$\frac{\begin{array}{c} G = \langle DO, DD, DE\rangle \qquad \forall \ell \in dom(S), \Sigma[\ell] = C\!<\!\overline{p}\!> \qquad H[\ell] = O = \langle C\!<\!\overline{D}\!>\rangle \in DO \\ \forall m.\ mbody(m, C\!<\!\overline{p}\!>) = (\overline{x} : \overline{T},\ e_R) \qquad \{\overline{x} : \overline{T},\ \texttt{this} : C\!<\!\overline{p}\!>\}, \emptyset, G \vdash_O e_R \end{array}}{G \vdash_{CT,H} \Sigma}[\text{Df-Sigma}]$$

$$\frac{}{\Gamma, \Upsilon, G \vdash_O \ell}[\text{Df-Loc}]$$

To Show:

(1) $(S', H', K', L_I', L_E') \sim (DO, DD, DE)$

(2) $\emptyset, \emptyset, G \vdash_O e'$

(3) $G \vdash_{CT,H'} \Sigma'$

$(S, H, K, L_I, L_E) \sim (DO, DD, DE)$ — By assumption

$S' = S, H' = H, K' = K, L_I' = L_I, L_E' = L_E$ — By sub-derivation of Ir-Context

This proves (1).

$\emptyset, \emptyset, G \vdash_O e'$ — By Df-Loc, since $e' = v$

This proves (2).

$G \vdash_{CT,H} \Sigma$ — By assumption

$S' = S, H' = H$ — By sub-derivation of Ir-Context

$G \vdash_{CT,H'} \Sigma'$ — Take $\Sigma' = \Sigma$

This proves (3).

Case Irc-New: e $= \texttt{new}\ C\!<\!\overline{p}\!>(v_{1..i-1}, e_i, e_{i+1..n})$, and $e' = \texttt{new}\ C\!<\!\overline{p}\!>(v_{1..i-1}, e_i', e_{i+1..n})$.

We have:

$$\frac{\begin{array}{c} G = \langle DO, DD, DE\rangle \qquad O = C_{\texttt{this}}\!<\!\overline{D_O}\!> \qquad \forall i \in 1..|\overline{p}| \qquad G \vdash_O D_i \in findD(C_{\texttt{this}}::p_i) \\ O_C = \langle\ C\!<\!\overline{D}\!>\ \rangle \qquad \{O_C\} \subseteq DO \\ G \vdash_O dparams(C, O_C) \qquad \{(O_C, qual(p_i)) \mapsto D_i\} \subseteq DD \\ G \vdash_O ddomains(C, O_C) \\ \forall m \in \overline{md}\ mbody(m, C\!<\!\overline{p}\!>) = (\overline{x} : \overline{T},\ e_R) \\ C\!<\!\overline{D}\!> \notin \Upsilon \implies \{\overline{x} : \overline{T},\ \texttt{this} : C\!<\!\overline{p}\!>\}, \Upsilon \cup \{C\!<\!\overline{D}\!>\}, G \vdash_{O_C} e_R \\ \Gamma, \Upsilon, G \vdash_O \overline{e} \end{array}}{\Gamma, \Upsilon, G \vdash_O \texttt{new}\ C\!<\!\overline{p}\!>(\overline{e})}[\text{Df-New}]$$

$$\frac{\theta \vdash e_i; S; H; K; L_I; L_E \rightsquigarrow_G e_i'; S'; H'; K'; L_I'; L_E'}{\theta \vdash \texttt{new } C\!<\!\overline{p}\!>\!(v_{1..i-1}, e_i, e_{i+1..n}); S; H; K; L_I; L_E \rightsquigarrow_G \texttt{new } C\!<\!\overline{p}\!>\!(v_{1..i-1}, e_i', e_{i+1..n}); S'; H'; K'; L_I'; L_E'}[\textsc{Irc-New}]$$

$$\frac{G = \langle DO, DD, DE\rangle \qquad \forall \ell \in dom(S), \Sigma[\ell] = C\!<\!\overline{p}\!> \qquad H[\ell] = O = \langle C\!<\!\overline{D}\!>\rangle \in DO \qquad \forall m.\ mbody(m, C\!<\!\overline{p}\!>) = (\overline{x} : \overline{T},\ e_R) \qquad \{\overline{x} : \overline{T},\ \texttt{this} : C\!<\!\overline{p}\!>\}, \emptyset, G \vdash_O e_R}{G \vdash_{CT,H} \Sigma}[\textsc{Df-Sigma}]$$

To Show:
(1) $(S', H', K', L_I', L_E') \sim (DO, DD, DE)$
(2) $\emptyset, \emptyset, G \vdash_O e'$
(3) $G \vdash_{CT,H'} \Sigma'$

$\theta \vdash e_i; S; H; K; L_I; L_E \rightsquigarrow_G e_i'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of Irc-New
$(S', H', K', L_I', L_E') \sim (DO, DD, DE)$ By induction hypothesis
This proves (1).

$\theta \vdash e_i; S; H; K; L_I; L_E \rightsquigarrow_G e_i'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of Irc-New
$\emptyset, \emptyset, G \vdash_O e_i'$ By induction hypothesis
$\emptyset, \emptyset, G \vdash_O \texttt{new } C\!<\!\overline{p}\!>\!(v_{1..i-1}, e_i', e_{i+1..n})$ By Df-New
This proves (2).

$\theta \vdash e_i; S; H; K; L_I; L_E \rightsquigarrow_G e_i'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of Irc-New
$G \vdash_{CT,H'} \Sigma'$ By induction hypothesis, take $\Sigma' = \Sigma$
This proves (3).

Case Irc-Read: e $= e_0.f_k$, and $e' = e_0'.f_k$.
We have:

$$\frac{e_0 : C\!<\!\overline{p}\!> \qquad (T_k\ f_k) \in fieldDecls(C) \qquad G \vdash_O import(C\!<\!\overline{p}\!>, T_k) \qquad \Gamma, \Upsilon, G \vdash_O e_0}{\Gamma, \Upsilon, G \vdash_O e_0.f_k}[\textsc{Df-Read}]$$

$$\frac{\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'}{\theta \vdash e_0.f_i; S; H; K; L_I; L_E \rightsquigarrow_G e_0'.f_i; S'; H'; K'; L_I'; L_E'}[\textsc{Irc-Read}]$$

$$\frac{G = \langle DO, DD, DE\rangle \qquad \forall \ell \in dom(S), \Sigma[\ell] = C\!<\!\overline{p}\!> \qquad H[\ell] = O = \langle C\!<\!\overline{D}\!>\rangle \in DO \qquad \forall m.\ mbody(m, C\!<\!\overline{p}\!>) = (\overline{x} : \overline{T},\ e_R) \qquad \{\overline{x} : \overline{T},\ \texttt{this} : C\!<\!\overline{p}\!>\}, \emptyset, G \vdash_O e_R}{G \vdash_{CT,H} \Sigma}[\textsc{Df-Sigma}]$$

To Show:
(1) $(S', H', K', L_I', L_E') \sim (DO, DD, DE)$
(2) $\emptyset, \emptyset, G \vdash_O e'$
(3) $G \vdash_{CT,H'} \Sigma'$

$\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of IRC-READ

$(S', H', K', L_I', L_E') \sim (G)$ By induction hypothesis

This proves (1).

$\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of IRC-READ

$\emptyset, \emptyset, G \vdash_O e_0'$ By induction hypothesis

$\emptyset, \emptyset, G \vdash_O e_0'.f_k$ By DF-READ

This proves (2).

$\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of IRC-READ

$G \vdash_{CT,H'} \Sigma'$ By induction hypothesis, take $\Sigma' = \Sigma$

This proves (3).

Case Irc-Write-Rcv: e = $(e_0.f_k = e_1)$, and $e' = (e_0'.f_k = e_1)$.

We have:

$$\frac{\begin{array}{c} e_0 : C\langle\overline{p}\rangle \quad (T_k\ f_k) \in fields(C\langle\overline{p}\rangle) \\ e_1 : C_1\langle\overline{p''}\rangle \quad C_1\langle\overline{p''}\rangle <: T_k \\ G \vdash_O export(C\langle\overline{p}\rangle, C_1\langle\overline{p''}\rangle) \\ \Gamma, \Upsilon, G \vdash_O e_0 \quad \Gamma, \Upsilon, G \vdash_O e_1 \end{array}}{\Gamma, \Upsilon, G \vdash_O e_0.f_k = e_1}[\text{DF-WRITE}]$$

$$\frac{\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'}{\begin{array}{c}\theta \vdash e_0.f_i = e_1; S; H; K; L_I; L_E \rightsquigarrow_G \\ e_0'.f_i = e_1; S'; H'; K'; L_I'; L_E'\end{array}}[\text{IRC-WRITERCV}]$$

$$\frac{\begin{array}{c} G = \langle DO, DD, DE\rangle \quad \forall \ell \in dom(S), \Sigma[\ell] = C\langle\overline{p}\rangle \quad H[\ell] = O = \langle C\langle\overline{D}\rangle\rangle \in DO \\ \forall m.\ mbody(m, C\langle\overline{p}\rangle) = (\overline{x} : \overline{T},\ e_R) \quad \{\overline{x} : \overline{T},\ \texttt{this} : C\langle\overline{p}\rangle\}, \emptyset, G \vdash_O e_R \end{array}}{G \vdash_{CT,H} \Sigma}[\text{DF-SIGMA}]$$

To Show:

(1) $(S', H', K', L_I', L_E') \sim (DO, DD, DE)$

(2) $\emptyset, \emptyset, G \vdash_O e'$

(3) $G \vdash_{CT,H'} \Sigma'$

$\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of IRC-WRITE-RCV

$(S', H', K', L_I', L_E') \sim (DO, DD, DE)$ By induction hypothesis

This proves (1).

$\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of IRC-WRITE-RCV

$\emptyset, \emptyset, G \vdash_O e_0'$ By induction hypothesis

$\emptyset, \emptyset, G \vdash_O e_1$ By DF-WRITE

$\emptyset, \emptyset, G \vdash_O e_0'.f_k = e_1$ By DF-WRITE

This proves (2).

$\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of IRC-WRITE-RCV

$G \vdash_{CT,H'} \Sigma'$ By induction hypothesis, take $\Sigma' = \Sigma$

This proves (3).

Case Irc-Write-Arg: e $= (v.f_k = e_1)$, and $e' = (v.f_k = e_1')$.

We have:

$$\frac{\begin{array}{c} e_0 : C\overline{<p>} \qquad (T_k\ f_k) \in fields(C\overline{<p>}) \\ e_1 : C_1\overline{<p''>} \qquad C_1\overline{<p''>} <: T_k \\ G \vdash_O export(C\overline{<p>}, C_1\overline{<p''>}) \\ \Gamma, \Upsilon, G \vdash_O e_0 \qquad \Gamma, \Upsilon, G \vdash_O e_1 \end{array}}{\Gamma, \Upsilon, G \vdash_O e_0.f_k = e_1}[\textsc{Df-Write}]$$

$$\frac{\theta \vdash e_1; S; H; K; L_I; L_E \rightsquigarrow_G e_1'; S'; H'; K'; L_I'; L_E'}{\begin{array}{c}\theta \vdash v.f_i = e_1; S; H; K; L_I; L_E \rightsquigarrow_G \\ v.f_i = e_1'; S'; H'; K'; L_I'; L_E'\end{array}}[\textsc{Irc-Write-Arg}]$$

$$\frac{\begin{array}{c} G = \langle DO, DD, DE\rangle \qquad \forall \ell \in dom(S), \Sigma[\ell] = C\overline{<p>} \qquad H[\ell] = O = \langle C\overline{<D>}\rangle \in DO \\ \forall m.\ mbody(m, C\overline{<p>}) = (\overline{x} : \overline{T},\ e_R) \qquad \{\overline{x} : \overline{T},\ \texttt{this} : C\overline{<p>}\}, \emptyset, G \vdash_O e_R \end{array}}{G \vdash_{CT,H} \Sigma}[\textsc{Df-Sigma}]$$

To Show:

(1) $(S', H', K', L_I', L_E') \sim (DO, DD, DE)$

(2) $\emptyset, \emptyset, G \vdash_O e'$

(3) $G \vdash_{CT,H'} \Sigma'$

$\theta \vdash e_1; S; H; K; L_I; L_E \rightsquigarrow_G e_1'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of Irc-Write-Arg

$(S', H', K', L_I', L_E') \sim (DO, DD, DE)$ By induction hypothesis

This proves (1).

$\theta \vdash e_1; S; H; K; L_I; L_E \rightsquigarrow_G e_1'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of Irc-Write-Arg

$\emptyset, \emptyset, G \vdash_O e_1'$ By induction hypothesis

$\emptyset, \emptyset, G \vdash_O e_0'.f_k = e_1'$ By Df-Write

This proves (2).

$\theta \vdash e_1; S; H; K; L_I; L_E \rightsquigarrow_G e_1'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of Irc-Write-Arg

$G \vdash_{CT,H'} \Sigma'$ By induction hypothesis, take $\Sigma' = \Sigma$

This proves (3).

Case Irc-RecvInvk: e $= e_0.m(\overline{e})$, and $e' = e_0'.m(\overline{e})$.

We have:

$$\frac{\begin{array}{c} e_0 : C\overline{<p>} \qquad mtype(m, C\overline{<p>}) = \overline{T'} \to T_R' \qquad mtypeDecl(m, C) = \overline{T_f} \to T_R \\ G \vdash_O import(C\overline{<p>}, T_R) \\ \forall k \in 1..|\overline{e}|\ e_k : T_a \qquad T_a <: T_k' \qquad G \vdash_O export(C\overline{<p>}, T_a) \\ \Gamma, \Upsilon, G \vdash_O e_0 \qquad \Gamma, \Upsilon, G \vdash_O \overline{e} \end{array}}{\Gamma, \Upsilon, G \vdash_O e_0.m(\overline{e})}[\textsc{Df-Invk}]$$

$$\frac{\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'}{\begin{array}{c}\theta \vdash e_0.m(\overline{e}); S; H; K; L_I; L_E \rightsquigarrow_G \\ e_0'.m(\overline{e}); S'; H'; K'; L_I'; L_E'\end{array}}[\textsc{Irc-RecvInvk}]$$

$$\frac{\begin{array}{c} G = \langle DO, DD, DE\rangle \qquad \forall \ell \in dom(S), \Sigma[\ell] = C\overline{<p>} \qquad H[\ell] = O = \langle C\overline{<D>}\rangle \in DO \\ \forall m.\ mbody(m, C\overline{<p>}) = (\overline{x} : \overline{T},\ e_R) \qquad \{\overline{x} : \overline{T},\ \texttt{this} : C\overline{<p>}\}, \emptyset, G \vdash_O e_R \end{array}}{G \vdash_{CT,H} \Sigma}[\textsc{Df-Sigma}]$$

To Show:

(1) $(S', H', K', L_I', L_E') \sim (DO, DD, DE)$

(2) $\emptyset, \emptyset, G \vdash_O e'$

(3) $G \vdash_{CT,H'} \Sigma'$

$\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of Irc-RecvInvk
$(S', H', K', L_I', L_E') \sim (DO, DD, DE)$ By induction hypothesis
This proves (1).

$\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of Irc-RecvInvk
$\emptyset, \emptyset, G \vdash_O e_0'$ By induction hypothesis
$\emptyset, \emptyset, G \vdash_O \overline{e}$ By Df-Invk
$\emptyset, \emptyset, G \vdash_O e_0'.m(\overline{e})$ By Df-Invk
This proves (2).

$\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of Irc-RecvInvk
$G \vdash_{CT,H'} \Sigma'$ By induction hypothesis, take $\Sigma' = \Sigma$
This proves (3).

Case Irc-ArgInvk: e $= v.m(v_{1..i-1}, e_i, e_{i+1..n})$,and $e' = v.m(v_{1..i-1}, e_i', e_{i+1..n})$.
We have:

$$\frac{\begin{array}{c} e_0 : C\!<\!\overline{p}\!> \qquad mtype(m, C\!<\!\overline{p}\!>) = \overline{T'} \to T_R' \qquad mtypeDecl(m, C) = \overline{T_f} \to T_R \\ G \vdash_O import(C\!<\!\overline{p}\!>, T_R) \\ \forall k \in 1..|\overline{e}|\ e_k : T_a \qquad T_a <: T_k' \qquad G \vdash_O export(C\!<\!\overline{p}\!>, T_a) \\ \Gamma, \Upsilon, G \vdash_O e_0 \qquad \Gamma, \Upsilon, G \vdash_O \overline{e} \end{array}}{\Gamma, \Upsilon, G \vdash_O e_0.m(\overline{e})}\text{[Df-Invk]}$$

$$\frac{\theta \vdash e_i; S; H; K; L_I; L_E \rightsquigarrow_G e_i'; S'; H'; K'; L_I'; L_E'}{\begin{array}{c} \theta \vdash v.m(v_{1..i-1}, e_i, e_{i+1..n}); S; H; K; L_I; L_E \rightsquigarrow_G \\ v.m(v_{1..i-1}, e_i', e_{i+1..n}); S'; H'; K'; L_I'; L_E' \end{array}}\text{[Irc-ArgInvk]}$$

$$\frac{\begin{array}{c} G = \langle DO, DD, DE\rangle \qquad \forall \ell \in dom(S), \Sigma[\ell] = C\!<\!\overline{p}\!> \qquad H[\ell] = O = \langle C\!<\!\overline{D}\!>\rangle \in DO \\ \forall m.\ mbody(m, C\!<\!\overline{p}\!>) = (\overline{x} : \overline{T},\ e_R) \qquad \{\overline{x} : \overline{T},\ \texttt{this} : C\!<\!\overline{p}\!>\}, \emptyset, G \vdash_O e_R \end{array}}{G \vdash_{CT,H} \Sigma}\text{[Df-Sigma]}$$

To Show:
(1) $(S', H', K', L_I', L_E') \sim (DO, DD, DE)$
(2) $\emptyset, \emptyset, G \vdash_O e'$
(3) $G \vdash_{CT,H'} \Sigma'$

$\theta \vdash e_i; S; H; K; L_I; L_E \rightsquigarrow_G e_i'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of Irc-ArgInvk

$(S', H', K', L_I', L_E') \sim (DO, DD, DE)$ By induction hypothesis

This proves (1).

$\theta \vdash e_i; S; H; K; L_I; L_E \rightsquigarrow_G e_i'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of Irc-ArgInvk

$\emptyset, \emptyset, G \vdash_O e_i'$ By induction hypothesis

$\emptyset, \emptyset, G \vdash_O v.m(v_{1..i-1}, e_i', e_{i+1..n})$ By Df-Invk

This proves (2).

$\theta \vdash e_i; S; H; K; L_I; L_E \rightsquigarrow_G e_i'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of Irc-ArgInvk

$G \vdash_{CT,H'} \Sigma'$ By induction hypothesis, take $\Sigma' = \Sigma$

This proves (3).

Case Irc-Context: e $= \ell \triangleright e_0$, and $e' = \ell \triangleright e_0'$.

We have:

$$\frac{O_C = H[\ell] \qquad \Gamma, \Upsilon, G \vdash_{O_C} e}{\Gamma, \Upsilon, G \vdash_{O,H} \ell \triangleright e}[\text{Df-Context}]$$

$$\frac{\ell \vdash e; S; H; K; L_I; L_E \rightsquigarrow_G e'; S'; H'; K'; L_I'; L_E'}{\theta \vdash \ell \triangleright e; S; H; K; L_I; L_E \rightsquigarrow_G \ell \triangleright e'; S'; H'; K'; L_I'; L_E'}[\text{Irc-Context}]$$

$$\frac{\begin{array}{l} G = \langle DO, DD, DE\rangle \qquad \forall \ell \in dom(S), \Sigma[\ell] = C\!<\!\overline{p}\!> \qquad H[\ell] = O = \langle C\!<\!\overline{D}\!>\rangle \in DO \\ \quad \forall m.\ mbody(m, C\!<\!\overline{p}\!>) = (\overline{x} : \overline{T},\ e_R) \qquad \{\overline{x} : \overline{T},\ \texttt{this} : C\!<\!\overline{p}\!>\}, \emptyset, G \vdash_O e_R \end{array}}{G \vdash_{CT,H} \Sigma}[\text{Df-Sigma}]$$

To Show:

(1) $(S', H', K', L_I', L_E') \sim (DO, DD, DE)$

(2) $\emptyset, \emptyset, G \vdash_O e'$

(3) $G \vdash_{CT,H'} \Sigma'$

$\ell \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of Irc-Context

$(S', H', K', L_I', L_E') \sim (DO, DD, DE)$ By induction hypothesis

This proves (1).

$\ell \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of Irc-Context

$O_\ell = H[\ell]$ By induction hypothesis

$\emptyset, \emptyset, G \vdash_{O_\ell} e_0'$ By induction hypothesis

$\emptyset, \emptyset, G \vdash_O \ell \triangleright e_0'$ By Df-Context

This proves (2).

$\ell \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'$ By sub-derivation of Irc-Context

$G \vdash_{CT,H'} \Sigma'$ By induction hypothesis, take $\Sigma' = \Sigma$

This proves (3).

■

3.5 Theorem: Dataflow Progress

$$
\begin{array}{l}
If \\
\boxed{\emptyset, \Sigma, \theta \vdash e : T} \\
\boxed{\Sigma \vdash S} \\
G \vdash_{CT,H} \Sigma \\
\emptyset, \emptyset, G \vdash_O e \\
(S, H, K, L_I, L_E) \sim (DO, DD, DE) \\
then \\
either\ \boxed{e\ is\ a\ value} \\
or\ else\ \theta \vdash \boxed{e; S}; H; K; L_I; L_E \rightsquigarrow_G \boxed{e'; S'}; H'; K'; L'_I; L'_E
\end{array}
$$

Proof: We prove progress by derivation of $\emptyset, \emptyset, G \vdash_O e$, with a case analysis on the last typing rule used. The most interesting cases are Df-New, Df-Read (page 38), Df-Write (page 41), and Df-Invk (page 43).

Case DF-NEW : e = *new* $C\langle\overline{p}\rangle(\overline{e})$.

Subcase $\overline{e} = \overline{v}$ that is $e = new\ C\langle\overline{p}\rangle(\overline{v})$. Take $e' = \ell$, then Ir-New can apply.

$$
\frac{\begin{array}{c}\boxed{\ell \notin dom(S) \qquad S' = S[\ell \mapsto C\langle\overline{p}\rangle(\overline{v})]} \\ G = \langle DO, DD, DE\rangle \\ \overline{p} = \overline{\ell'.d} \qquad \forall i \in 1..|\overline{\ell'.d}|\ D_i = K[\ell'_i.d_i] \\ \ell_i \in dom(H) \text{ s.t. } H[\ell_i] = O_i \qquad D_i = DD[O_i, qual(\ell'_i.d_i)] \\ O_C = \langle C\langle\overline{D}\rangle\rangle \qquad O_C \in DO \qquad H' = H[\ell \mapsto O_C] \\ \forall(\texttt{domain}\ d_j) \in domains(C\langle\overline{p}\rangle) \qquad D_j = DD[(O_C, C{::}d_j)] \qquad K' = K[\ell.d_j \mapsto D_j]\end{array}}{\theta \vdash \boxed{\texttt{new}\ C\langle\overline{p}\rangle(\overline{v}); S}; H; K; L_I; L_E \rightsquigarrow_G \boxed{\ell; S'}; H'; K'; L_I; L_E}[\text{Ir-New}]
$$

$$
\frac{\Sigma[\ell] = C\langle\overline{\ell''.d}\rangle \qquad d \in domains(C\langle\overline{\ell''.d}\rangle)}{\Gamma; \Sigma; \theta \vdash qual(\ell.d) = C{::}d}
$$

To show:
(1) $\forall i \in 1..|\overline{\ell'.d}| \quad D_i = K[\ell'_i.d_i]$
(2) $O_C = \langle C\langle\overline{D}\rangle\rangle \quad O_C \in DO$
(3) $\forall i \in 1..|\overline{\ell'.d}| \quad H[\ell'_i] = O_i \quad D_i = DD[(O_i, qual(\ell'.d_i))]$
(4) $\forall d_j \in domains(C\langle\overline{\ell'.d}\rangle) \quad D_j = DD[(O_C, C{::}d_j)]$

$(S, H, K, L_I, L_E) \sim (DO, DD, DE)$ By assumption

$\forall \ell \in dom(S), \Sigma[\ell] = C\overline{<\ell'.d>}$ $\Sigma \vdash S$

$\forall \ell \in dom(S), \Sigma[\ell] = C\overline{<\ell'.d>}$

$\Longrightarrow$

$H[\ell] = O_C = \langle C\overline{<D>}\rangle \in DO$

and $\forall \ell'_j.d_j \in \overline{\ell'.d}\ K[\ell'_j.d_j] = D_j = \langle D_{id_j}, qual(\ell'_j.d_j)\rangle \in rng(DD)$

and $\forall d_i \in domains(C\overline{<\ell'.d>})$

$K[\ell.d_i] = D_i = \langle D_{id_i}, C::d_i\rangle\ \{(O_C, C::d_i) \mapsto D_i\} \in DD$

and $\forall \ell_{src} \in dom(H),\ fields(\Sigma[\ell_{src}]) = \overline{T_{src}}\ \overline{f}$

$\forall m.\ mtype(m, \Sigma[\ell_{src}]) = \overline{T} \to T_R$

$\forall T_k \in \{\overline{T_{src}}\} \cup \{T_R\}$

$H; K; L_I; L_E \vdash O_k \in irLookup(T_k)$

$E'_k \in L_I[(\ell_{src}, \ell)]\ E'_k = \langle H[\ell_{src}], H[\ell], O_k, Imp\rangle \in DE$

and $\forall \ell_{dst} \in dom(H),\ fields(\Sigma[\ell_{dst}]) = \overline{T_{dst}}\ \overline{f}$

$\forall m.\ mtype(m, \Sigma[\ell_{dst}]) = \overline{T} \to T_R$

$\forall T_k \in \{\overline{T_{dst}}\} \cup \{\overline{T}\}$

$H; K; L_I; L_E \vdash O_k \in irLookup(T_k)$

$E_k \in L_E[(\ell, \ell_{dst})]\ E_k = \langle H[\ell], H[\ell_{dst}], O_k, Exp\rangle \in DE$

This proves (1,2).

$\emptyset, \emptyset, G \vdash_O \texttt{new}\ C\overline{<\ell'.d>}(\overline{v})$	By assumption
$\forall i \in 1..\lvert\overline{\ell'.d}\rvert \quad G \vdash_O D_i \in findD(C_{this}::\ell'_i.d_i)$	By sub-derivation of Df-New
$O_C = \langle\ C\overline{<D>}\ \rangle \quad \{O_C\} \subseteq DO$	By sub-derivation of Df-New
$G \vdash_O dparams(C, O_C)$	By sub-derivation of Df-New
$\{(O_C, C::\alpha_i) \mapsto D_i\} \subseteq DD \quad \alpha_i \in params(C)$	By sub-derivation of AuxAlpha
$\{(O_C, qual(\ell'_i.d_i)) \mapsto D_i\} \subseteq DD$	By sub-derivation of Df-New
$G \vdash_O ddomains(C, O_C)$	By sub-derivation of Df-New
This proves (4).	By Df-Domains Lemma

Sub-subcase $\ell'_i \neq \theta$, $\ell'_i.d_i$ is a domain of some object in the context of θ That is, $\exists\ n \in dom(H)$ such that :

$DD[(H[n], C_n::d_i)] = D_i$	
$G \vdash_O H[n] \in lookup(\Sigma[n])$	by subderivation of Aux-Find-Public
This proves (3).	Take $\ell_i = n$

Sub-subcase $\ell'_i \neq \theta$, $\ell'_i.d_i$ substitutes the j^{th} formal domain parameters of C_{this}, where $\Sigma[\theta] = C_{this}<\overline{\ell_{this}.d}>$ That is:

$O = \langle C_{this}<\overline{D'}>\rangle$
$DD[(O, C_{this}::\alpha_i)] = D'_j$
$DD[(O, qual(\ell'_i.d_i))] = D'_j$ by subderivation of Df-New

$\forall i \in 1..|\overline{\ell'.d}| \quad G \vdash_O D_i \in findD(C_{this}::\ell'_i.d_i)$ By sub-derivation of Df-New
$D_i = DD[(O, qual(\ell'_i.d_i)] = DD[(O, C_{this}::\alpha_j)] = D'_j$ By sub-derivation of Aux-FindD
This proves (3). Take $\ell_i = \theta$

Sub-subcase $\ell'_i = \theta$, $\ell'_i.d_i$ is a domain of θ

$G \vdash_O D_i \in findD(C_{this}::\ell'_i.d_i)$ By above
$D_i = DD[(O, C::d_i))]$ By sub-derivation of Aux-FindThis
$d_i \in domains(\Sigma[\ell_i])$
$qual(\ell'_i.d_i) = C::d_i$ By inversion of Qual-Var
$D_i = DD[(O, qual(\ell'_i.d_i))]$ By above
This proves (3). Take $\ell_i = \theta$

Sub-subcase $d_i =$ ::shared

$G \vdash_O D_i \in findD(C_{this}::\ell'_i.d_i)$ By above
$D_i = D_{\texttt{shared}} = DD[(O_{world}, ::\texttt{shared})]$ By sub-derivation of Aux-FindShared
This proves (3).

Subcase $e = new\ C<\overline{p}>(v_{1..i-1}, e_i, e_{i+1..n})$. Then Irc-New can apply.

$$\frac{\theta \vdash e_i; S; H; K; L_I; L_E \rightsquigarrow_G e'_i; S'; H'; K'; L'_I; L'_E}{\theta \vdash \texttt{new}\ C<\overline{p}>(v_{1..i-1}, e_i, e_{i+1..n}); S; H; K; L_I; L_E \rightsquigarrow_G \texttt{new}\ C<\overline{p}>(v_{1..i-1}, e'_i, e_{i+1..n}); S'; H'; K'; L'_I; L'_E}[\text{IRC-New}]$$

$\Gamma, \Upsilon, G \vdash_O e_i$ By sub-derivation of Df-New
$\theta \vdash e_i; S; H; K; L_I; L_E \rightsquigarrow_G e'_i; S'; H'; K'; L'_I; L'_E$ By induction hypothesis
$\theta \vdash \texttt{new}\ C<\overline{p}>(v_{1..i-1}, e_i, e_{i+1..n}) S; H; K; L_I; L_E \rightsquigarrow_G$
$\texttt{new}\ C<\overline{p}>(v_{1..i-1}, e'_i, e_{i+1..n}); S'; H'; K'; L'_I; L'_E$ By inversion of Irc-New
This proves the case.

Case DF-VAR : $e = x$.
Not applicable since variable is not a closed term.

Case DF-LOC : $e = \ell$.
e is a value.

Case DF-READ : e = $e_0.f_i$. There are two subcases to consider depending on whether the receiver e_0 is a value.

Subcase $e_0 = \ell$. Then $e = \ell.f_i$

From Ir-Read:

$$\frac{\boxed{S[\ell] = C\langle\overline{p}\rangle(\overline{v}) \qquad fields(C\langle\overline{p}\rangle) = \overline{T}\ \overline{f}} \quad O = H[\theta] \quad O_\ell = H[\ell] \quad O_v = H[v_i] \quad T_i \in \overline{T} \quad E = \langle O_\ell, O, O_v, Imp\rangle \in DE \quad H; K; L_I; L_E \vdash O_v \in irLookup(T_i) \quad L'_I = L_I[(\ell,\theta) \mapsto_\cup \{E\}]}{\theta \vdash \boxed{\ell.f_i; S}; H; K; L_I; L_E \rightsquigarrow_G \boxed{v_i; S}; H; K; L'_I; L_E}$$

To show:
(1) $O = H[\theta]$
(2) $O_\ell = H[\ell]$
(3) $O_v = H[v_i] \quad T_i \in \overline{T} \quad H, K, L_I, L_E \vdash O_v \in irLookup(T_i) \quad E = \langle O_\ell, O, C_v, Imp\rangle \in DE$

$G \vdash_{CT,H} \Sigma$	By assumption
$\forall \ell' \in dom(S), \Sigma[\ell'] = C'\langle\overline{p}\rangle$	By sub-derivation of Df-Sigma
$H[\ell'] = O' = \langle C'\langle\overline{D'}\rangle\rangle \in DO$	By sub-derivation of Df-Sigma
$H[\theta] = O = \langle C\langle\overline{D}\rangle\rangle \in DO$	Since $\theta \in dom(S)$
$H[\ell] = O_\ell = \langle C_\ell\langle\overline{D_\ell}\rangle\rangle \in DO$	Since $\ell \in dom(S)$
$H[v_i] = O_v = \langle C_v\langle\overline{D_v}\rangle\rangle \in DO$	Since $v \in dom(S)$
this proves (1), and (2).	

$(S, H, K, L_I, L_E) \sim (DO, DD, DE)$	By assumption
$\forall \ell \in dom(S),\ \Sigma[\ell] = C\langle\overline{\ell'.d}\rangle$	Since $\Sigma \vdash S$
$H[\theta] = O_C = \langle C\langle\overline{D}\rangle\rangle \in DO$	
$and\ \forall \theta'_j.d_j \in \overline{\theta'.d}\ K[\theta'_j.d_j] = D_j = \langle D_{id_j}, qual(\theta'_j.d_j)\rangle \in rng(DD)$	
$and\ \forall d_i \in domains(C\langle\overline{\theta'.d}\rangle)$	
$K[\theta.d_i] = D_i = \langle D_{id_i}, C::d_i\rangle\ \{(O_C, C::d_i) \mapsto D_i\} \in DD$	By Df-Approx
$\forall \ell_{src} \in dom(H),\ fields(\Sigma[\ell_{src}]) = \overline{T_{src}}\ \overline{f}$	
$\forall m.\ mtype(m, \Sigma[\ell_{src}]) = \overline{T} \to T_R$	
$\forall T_k \in \{\overline{T_{src}}\} \cup \{T_R\}$	
$H; K; L_I; L_E \vdash O_k \in irLookup(T_k)$	
$E'_k \in L_I[(\ell_{src}, \theta)]\ E'_k = \langle H[\ell_{src}], H[\theta], O_k, Imp\rangle \in DE$	By Df-Approx
$\forall \ell_{dst} \in dom(H),\ fields(\Sigma[\ell_{dst}]) = \overline{T_{dst}}\ \overline{f}$	
$\forall m.\ mtype(m, \Sigma[\ell_{dst}]) = \overline{T} \to T_R$	
$\forall T_k \in \{\overline{T_{dst}}\} \cup \{\overline{T}\}$	
$H; K; L_I; L_E \vdash O_k \in irLookup(T_k)$	
$E_k \in L_E[(\theta, \ell_{dst})]\ E_k = \langle H[\theta], H[\ell_{dst}], O_k, Exp\rangle \in DE$	By Df-Approx

$\emptyset, \emptyset, G \vdash_O \ell.f_i$ By assumption

$fields(\Sigma[\ell]) = \overline{T'}\ \overline{f}$ By FDJ T-Store

Since $e_0 = \ell \in dom(H)$

$\ell : \Sigma[\ell] = C_\ell\texttt{<}\overline{p}\texttt{>}\ (T'_i\ f_i) \in fieldDecls(C_\ell)$ By sub-derivation of DF-READ

$G \vdash_O import(\Sigma[\ell], T'_i)$ By sub-derivation of DF-READ

$\emptyset, \emptyset, G \vdash_O \ell$ By sub-derivation of DF-READ

$G \vdash_O O_i \in lookup(\Sigma[\ell])$ By sub-derivation of AUX-IMPORT

$G \vdash_{O_i} O_j \in lookup(T'_i)$ By sub-derivation of AUX-IMPORT

$\langle O_i, O, O_j, Imp\rangle \subseteq DE$ By sub-derivation of AUX-IMPORT

$\emptyset, \Sigma, \theta \vdash \ell : \Sigma[\ell]$ By hypothesis

$\Sigma[\ell] = C_\ell\texttt{<}\overline{\ell'.d}\texttt{>} \quad H[\ell] = O_\ell = \langle C_\ell\texttt{<}\overline{D_\ell}\texttt{>}\rangle \in DO$

$G \vdash_{H[\theta]} H[\ell] \in lookup(\Sigma[\ell])$ By Lookup Lemma

$S[\ell] = C_\ell\texttt{<}\overline{\ell'.d}\texttt{>}(\overline{v}), \Sigma[v] = C_v\texttt{<}\overline{v'.d}\texttt{>} \quad H[v_i] = O_v = \langle C_v\texttt{<}\overline{D_v}\texttt{>}\rangle \in DO$

$T'_i = C'_v\texttt{<}\overline{p}\texttt{>} \quad C_v <: C'_v$ By sub-derivation of DF-READ

To show

$\forall p_k \in \overline{p} \quad G \vdash_{H[\ell]} D'_{vk} \in findD(C_\ell\texttt{::}p_k) \quad D'_{vk} = D_{vk} = K[v'_k.d_k]$

p_k is a domain parameter, local domain of ℓ or `shared`. Therefore we split the proof in cases

Case domain parameter: $p_k = \alpha_j$

$$\exists \ell'_j.d_j \in \overline{\ell'.d} \text{ s.t } K[v'_k.d_k] = K[\ell'_j.d_j] = D_j$$

$$D_{vk} = D_j = DD[(H[\ell], C_\ell\texttt{::}\alpha_j)] = DD[(H[\ell], qual(\ell'_j.d_j))] = K[\ell'_j.d_j] \qquad \text{By Params Lemma}$$

Case local domain: $p_k = \ell.d$

$$\exists \ell'_j.d_j \in \overline{\ell'.d} \text{ s.t } K[v'_k.d_k] = K[\ell'_j.d_j] = D_j$$

$$D_{vk} = D_j = DD[(H[\ell], C_\ell\texttt{::}d_j)] = K[\ell.d_j] \qquad \text{By sub-derivation of DF-NEW}$$

Case local domain: $p_k = \texttt{shared}$

$$D_{vk} = D_{\texttt{shared}} = DD[(H[\ell], \texttt{::shared})]$$

$G \vdash_{H[\ell]} D_{vk} \in findD(C_\ell\texttt{::}qual(v'_k.d_k))$ By inversion of AuxFindD

$G \vdash_{H[\ell]} H[v_i] \in lookup(T'_i)$ By inversion of AUX-LOOKUP.Take $O_i = H[\ell]$

$\langle H[\ell], H[\theta], H[v], Imp\rangle \subseteq DE$ By above. Take $O = H[\theta]\ O_i = H[\ell]\ O_j = H[v_i]$

$H[v] = O_{C_v} = \langle C_v{<}\overline{D}{>}\rangle \in DO$
and $\forall v'_j.d_j \in \overline{v'.d}\ K[v'_j.d_j] = D_j = \langle D_{id_j}, qual(v'_j.d_j)\rangle \in rng(DD)$
and $\forall d_i \in domains(C_v{<}\overline{v'.d}{>})$
$K[v.d_i] = D_i = \langle D_{id_i}, C_v{::}d_i\rangle\ \{(O_{C_v}, C_v{::}d_i) \mapsto D_i\} \in DD$ By Df-Approx
$H; K; L_I; L_E \vdash H[v] \in irLookup(\Sigma[v])$ By inversion of IR-Lookup
Take $T_k = T'_i \in \overline{T'}$, this proves (3).

Subcase $e_0 = e'_0.f_i$**.** That is, e_0 is not a value
From Irc-Read:

$$\frac{\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e'_0; S'; H'; K'; L'_I; L'_E}{\theta \vdash e_0.f_i; S; H; K; L_I; L_E \rightsquigarrow_G e'_0.f_i; S'; H'; K'; L'_I; L'_E}$$

$\theta \vdash e'_0; S; H; K; L_I; L_E \rightsquigarrow_G e''_0; S'; H'; K'; L'_I; L'_E$ By induction hypothesis
$\theta \vdash e'_0.f_i; S; H; K; L_I; L_E \rightsquigarrow_G e''_0.f_i; S'; H'; K'; L'_I; L'_E$ By Irc-Read
Take $e' = e''_0.f_i$.

Case DF-WRITE : e = $(e_0.f_i = e_1)$. There are three subcases to consider depending on whether the receiver e_0, and e_1 are values.

Subcase $e_0 = \ell$, and $e_1 = v$. Then $e = (\ell.f_i = v)$

From IR-WRITE

$$\frac{\boxed{S[\ell] = C\langle\overline{p}\rangle(\overline{v}) \qquad fields(C\langle\overline{p}\rangle) = \overline{T}\ \overline{f}} \quad \boxed{S' = S[\ell \mapsto C\langle\overline{p}\rangle([v/v_i]\overline{v})]} \quad O = H[\theta] \quad O_\ell = H[\ell] \quad O_v = H[v] \quad H;K;L_I;L_E \vdash O_v \in irLookup(T_i) \quad T_i \in \overline{T} \quad E = \langle O, O_\ell, O_v, Exp\rangle \in DE \quad L'_E = L_E[(\theta,\ell) \mapsto_\cup \{E\}]}{\theta \vdash \boxed{\ell.f_i = v; S}; H; K; L_I; L_E \rightsquigarrow_G \boxed{v; S'}; H; K; L_I; L'_E}$$

To show:
(1) $O = H[\theta]$
(2) $O_\ell = H[\ell]$
(3) $O_v = H[v_i]$ $H;K;L_I;L_E \vdash O_v \in irLookup(T_i)$ $E = \langle O, O_\ell, O_v, Exp\rangle \in DE$

$G \vdash_{CT,H} \Sigma$	By assumption
$\forall \ell' \in dom(S), \Sigma[\ell'] = C'\langle\overline{p}\rangle$	By sub-derivation of DF-SIGMA
$H[\ell'] = O' = \langle C'\langle\overline{D'}\rangle\rangle \in DO$	By sub-derivation of DF-SIGMA
$H[\theta] = O = \langle C\langle\overline{D}\rangle\rangle \in DO$	Since $\theta \in dom(S)$
$H[\ell] = O_\ell = \langle C_\ell\langle\overline{D_\ell}\rangle\rangle \in DO$	Since $\ell \in dom(S)$
$H[v] = O_v = \langle C_v\langle\overline{D_v}\rangle\rangle \in DO$	Since $v \in dom(S)$
this proves (1), and (2).	

$(S, H, K, L_I, L_E) \sim (DO, DD, DE)$	By assumption
$\forall \ell \in dom(S),\ \Sigma[\ell] = C\langle\overline{\ell'.d}\rangle$	Since $\Sigma \vdash S$

$H[\theta] = O_C = \langle C\langle\overline{D}\rangle\rangle \in DO$
and $\forall \theta'_j.d_j \in \overline{\theta'.d}\ K[\theta'_j.d_j] = D_j = \langle D_{id_j}, qual(\theta'_j.d_j)\rangle \in rng(DD)$
and $\forall d_i \in domains(C\langle\overline{\theta'.d}\rangle)$
$K[\theta.d_i] = D_i = \langle D_{id_i}, C::d_i\rangle\ \{(O_C, C::d_i) \mapsto D_i\} \in DD$

$\emptyset, \emptyset, G \vdash_O \ell.f_i = v$ By assumption:

Since $e_0 = \ell \in dom(H) \quad e_1 = v$:

$\ell : \Sigma[\ell] = C_\ell\langle\overline{p}\rangle \ (T_i' \ f_i) \in fields(C_\ell\langle\overline{p}\rangle) = \overline{T'} \ \overline{f} \ T_i' = C_i\langle\overline{p'}\rangle$ By sub-derivation of Df-Write

$v : \Sigma[v] = C_v\langle\overline{p''}\rangle \quad \Sigma[v] <: T_i'$ By sub-derivation of Df-Write

$G \vdash_O export(\Sigma[\ell], \Sigma[v])$ By sub-derivation of Df-Write

$G \vdash_O O_i \in lookup(\Sigma[\ell])$ By sub-derivation of Aux-Export

$G \vdash_O O_j \in lookup(\Sigma[v])$ By sub-derivation of Aux-Export

$\langle O, O_i, O_j, Exp\rangle \in DE$ By sub-derivation of Aux-Export

$\Sigma[\ell] = C_\ell\langle\overline{\ell'.d}\rangle \quad H[\ell] = O_\ell = \langle C_\ell\langle\overline{D_\ell}\rangle\rangle \in DO$

$\emptyset, \emptyset, G \vdash_O \ell$ By sub-derivation of Df-Write

$\emptyset; \Sigma; \theta \vdash \ell : \Sigma[\ell]$ By Hypothesis

$G \vdash_{H[\theta]} H[\ell] \in lookup(\Sigma[\ell])$ By Lookup Lemma

$\emptyset, \emptyset, G \vdash_O v$ By sub-derivation of Df-Write

$\emptyset; \Sigma; \theta \vdash v : \Sigma[v]$ By Hypothesis

$\Sigma[v] = C_v\langle\overline{v'.d}\rangle \quad H[v] = O_v = \langle C_v\langle\overline{D_v}\rangle\rangle \in DO$

$G \vdash_{H[\theta]} H[v] \in lookup(\Sigma[v])$ By Lookup Lemma

$\Sigma[v] = C_v\langle\overline{v'.d}\rangle \quad H[v] = O_v = \langle C_v\langle\overline{D_v}\rangle\rangle \in DO$

$\forall v_j'.d_j \in \overline{v'.d} \ K[v_j'.d_j] = D_j \in rng(DD)$ By Df-Approx

$\Sigma[v] <: T_i'$ By sub-derivation of Df-Write

$H; K; L_I; L_E \vdash H[v] \in irLookup(T_i')$ By inversion of IR-Lookup

Take $T_k = T_i' \in \overline{T'}$, this proves (3).

Subcase $e_0 = e_0'$. Then $e = (e_0'.f_i = e_1)$

From IRC-Write-Rcv:

$$\frac{\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e_0'; S'; H'; K'; L_I'; L_E'}{\theta \vdash e_0.f_i = e_1; S; H; K; L_I; L_E \rightsquigarrow_G e_0'.f_i = e_1; S'; H'; K'; L_I'; L_E'}$$

$\theta \vdash e_0'; S; H; K; L_I; L_E \rightsquigarrow_G e_0''; S'; H'; K'; L_I'; L_E'$ By induction hypothesis

$\theta \vdash e_0'.f_i = e_1; S; H; K; L_I; L_E \rightsquigarrow_G e_0''.f_i = e_1; S'; H'; K'; L_I'; L_E'$ By IRC-Write-Rcv

Take $e' = (e_0''.f_i = e_1)$.

Subcase $e_0 = v$, and $e_1 = e_1'$. Then $e = (v.f_i = e_1')$

From IRC-Write-Arg:

$$\frac{\theta \vdash e_1; S; H; K; L_I; L_E \rightsquigarrow_G e_1'; S'; H'; K'; L_I'; L_E'}{\theta \vdash v.f_i = e_1; S; H; K; L_I; L_E \rightsquigarrow_G v.f_i = e_1'; S'; H'; K'; L_I'; L_E'}$$

$\Gamma, \Upsilon, G \vdash_O e_1$ By sub-derivation of Df-Write

$\theta \vdash e_1; S; H; K; L_I; L_E \rightsquigarrow_G e_1'; S'; H'; K'; L_I'; L_E'$ By induction hypothesis

$\theta \vdash v.f_i = e_1; S; H; K; L_I; L_E \rightsquigarrow_G v.f_i = e_1'; S'; H'; K'; L_I'; L_E'$ By IRC-Write-Arg

Take $e' = (v.f_i = e_1')$.

Case DF-INVK : e = $e_0.m(\overline{e})$. There are three subcases to consider, depending on whether the receiver e_0, or the arguments $\overline{e}$ are values.

Subcase $e_0 = \ell$, and $\overline{e} = \overline{v}$ that is $e = \ell.m(\overline{v})$

From IR-INVK:

$$\frac{\boxed{S[\ell] = C\!<\!\overline{p}\!>\!(\overline{v}) \qquad mbody(m, C\!<\!\overline{p}\!>) = (\overline{x}, e_R)} \quad O = H[\theta] \quad O_\ell = H[\ell] \quad mtype(m, C\!<\!\overline{p}\!>) = \overline{T} \to T_R \quad H; K; L_I; L_E \vdash O_r \in irLookup(T_R) \quad E' = \langle O_\ell, O, O_r, Imp\rangle \in DE \quad L'_I = L_I[(\ell, \theta) \mapsto_\cup \{E'\}] \quad \forall k \in 1..|\overline{x}|\ O_k = H[v_k] \quad H; K; L_I; L_E \vdash O_k \in irLookup(T_k) \quad T_k \in \overline{T} \quad E_k = \langle O, O_\ell, O_k, Exp\rangle \in DE \quad L'_E = L_E[(\theta, \ell) \mapsto_\cup \{E_k\}]}{\theta \vdash \boxed{\ell.m(\overline{v}); S}; H; K; L_I; L_E \rightsquigarrow_G \boxed{\ell \triangleright [\overline{v}/\overline{x}, \ell/\texttt{this}]e_R; S}; H; K; L'_I; L'_E}$$

To show:
(1) $O = H[\theta]$
(2) $O_\ell = H[\ell]$
(3) $mtype(m, C_\ell\!<\!\overline{p}\!>) = \overline{T} \to T_R \quad \forall O_r.H; K; L_I; L_E \vdash O_r \in irLookup(T_R) \quad E' = \langle O_\ell, O, O_r, Imp\rangle \in DE$
(4) $\forall k \in 1..|\overline{T}|\ O_k = H[v_k] \quad H; K; L_I; L_E \vdash O_k \in irLookup(T_k) \quad E_k = \langle O, O_\ell, O_k, Exp\rangle \in DE$

$G \vdash_{CT,H} \Sigma$ — By assumption
$\forall \ell' \in dom(S), \Sigma[\ell'] = C'\!<\!\overline{p}\!>$ — By sub-derivation of DF-SIGMA
$H[\ell'] = O' = \langle C'\!<\!\overline{D'}\!>\rangle \in DO$ — By sub-derivation of DF-SIGMA
$H[\theta] = O = \langle C\!<\!\overline{D}\!>\rangle \in DO$ — Since $\theta \in dom(S)$
$H[\ell] = O_\ell = \langle C_\ell\!<\!\overline{D_\ell}\!>\rangle \in DO$ — Since $\ell \in dom(S)$
this proves (1), and (2).

$H[\theta] = O_C = \langle C\!<\!\overline{D}\!>\rangle \in DO$
and $\forall \theta'_j.d_j \in \overline{\theta'.d}\ K[\theta'_j.d_j] = D_j = \langle D_{id_j}, qual(\theta'_j.d_j)\rangle \in rng(DD)$
and $\forall d_i \in domains(C\!<\!\overline{\theta'.d}\!>)$
$K[\theta.d_i] = D_i = \langle D_{id_i}, C{::}d_i\rangle\ \{(O_C, C{::}d_i) \mapsto D_i\} \in DD$ — By DF-APPROX
$\forall \ell_{src} \in dom(H),\ fields(\Sigma[\ell_{src}]) = \overline{T_{src}}\ \overline{f}$
$\forall m.\ mtype(m, \Sigma[\ell_{src}]) = \overline{T} \to T_R$
$\forall T_k \in \{\overline{T_{src}}\} \cup \{T_R\}$
$H; K; L_I; L_E \vdash O_k \in irLookup(T_k)$
$E'_k \in L_I[(\ell_{src}, \theta)]\ E'_k = \langle H[\ell_{src}], H[\theta], O_k, Imp\rangle \in DE$ — By DF-APPROX
$\forall \ell_{dst} \in dom(H),\ fields(\Sigma[\ell_{dst}]) = \overline{T_{dst}}\ \overline{f}$
$\forall m.\ mtype(m, \Sigma[\ell_{dst}]) = \overline{T} \to T_R$
$\forall T_k \in \{\overline{T_{dst}}\} \cup \{\overline{T}\}$
$H; K; L_I; L_E \vdash O_k \in irLookup(T_k)$
$E_k \in L_E[(\theta, \ell_{dst})]\ E_k = \langle H[\theta], H[\ell_{dst}], O_k, Exp\rangle \in DE$ — By DF-APPROX

$\emptyset, \emptyset, G \vdash_O \ell.m(\overline{v})$ By assumption

$\ell : \Sigma[\ell] = C_\ell\langle\overline{\ell'.d}\rangle$ By sub-derivation of Df-Invk

$mtype(m, C_\ell\langle\overline{\ell'.d}\rangle) = \overline{T'} \to T'_R$ By sub-derivation of Df-Invk

$mtypeDecl(m, C_\ell) = \overline{T_f} \to T_R$ By sub-derivation of Df-Invk

$G \vdash_O import(\Sigma[\ell], T_R)$ By sub-derivation of Df-Invk

$G \vdash_O O_i \in lookup(\Sigma[\ell])$ By subderivation of Aux-Import

$G \vdash_{O_i} O_j \in lookup(T_R)$ By subderivation of Aux-Import

$E' = \langle O_i, O, O_j, Exp\rangle \in DE$ By subderivation of Aux-Import

$\theta \vdash H; K; L_I; L_E, \texttt{new}\ C_\ell\langle\overline{\ell'.d}\rangle(\overline{v}) \rightsquigarrow_G \ell; S'; K'; L'_I; L'_E$ By induction hypothesis

$\emptyset, \Sigma, \theta \vdash \ell : \Sigma[\ell]$ By hypothesis

$\emptyset, \emptyset, G \vdash_O \ell$ By sub-derivation of Df-Invk

$G \vdash_O H[\ell] \in lookup(\Sigma[\ell])$ By Lookup Lemma

$T'_R = \Sigma[\ell''] = C_R\langle\overline{\ell'''.d}\rangle \quad T_R = C_R\langle\overline{p}\rangle \quad C_R\langle\overline{\ell'''.d}\rangle = [\ldots \ell'''_j.d_j/p_j \ldots]C_R\langle\overline{p}\rangle$ By definition of $mtype, mtypeDecl$

$\forall H; K; L_I; L_E \vdash O_r \in irLookup(T'_R)$

$G \vdash_{H[\ell]} O_r \in lookup(T_R)$ To Show

$O_r \in rng(H) \quad O_r = \langle C'\langle\overline{D'}\rangle\rangle \quad C' <: C_R$ By subderivation of IR-Lookup

$\forall \ell'''_j.d_j \in \overline{\ell''.d} \quad D''_j = K[\ell'''_j.d_j] \quad D''_j = D'_j$ By subderivation of IR-Lookup

To show

$\forall p_j \in \overline{p} \quad G \vdash_{H[\ell]} D''_j \in findD(C_\ell{::}p_j) \quad D''_j = D'_j = K[\ell'''_j.d_j]$

p_j is a domain parameter, local domain of ℓ or `shared`. Therefore we split the proof in cases

Case domain parameter: $p_j = \alpha_i$

$$\exists \ell'_i.d_i \in \overline{\ell'.d} \text{ s.t } K[\ell'''_j.d_j] = K[\ell'_i.d_i] = D_{\ell i} \quad D_{\ell i} \in \overline{D_\ell} \quad H[\ell] = O_\ell = \langle C_\ell\langle\overline{D_\ell}\rangle\rangle$$

$$D'_j = D_{\ell i} = DD[(H[\ell], C_\ell{::}\alpha_i)] = DD[(H[\ell], qual(\ell'_i.d_i))] = K[\ell'_i.d_i] \quad \text{By Params Lemma}$$

Case local domain: $p_k = \ell.d_j$

$$D_j = DD[(H[\ell], C_\ell{::}d_j)] = K[\ell.d_j] \quad \text{By sub-derivation of Df-New}$$

Case local domain: $p_k = \texttt{shared}$

$$D_{vk} = D_{\texttt{shared}} = DD[(H[\ell], {::}\texttt{shared})]$$

We showed

$\forall p_j \in \overline{p} \quad G \vdash_{H[\ell]} D''_j \in findD(C_\ell{::}p_j) \quad D''_j = D'_j$

$G \vdash_{H[\ell]} O_r \in lookup(T_R)$ By inversion of Aux-Lookup

$\forall O_r, E' = \langle H[\ell], H[\theta], O_r, Imp\rangle \in DE$ By above. Take $O_i = H[\ell], O = H[\theta]$

$\forall i \in 1..|\overline{v}|\ v_k : \Sigma[v_k]\ \Sigma[v_k] <: T'_k\ G \vdash_O export(\Sigma[\ell], \Sigma[v_k])$ By sub-derivation of Df-Invk

$\emptyset, \emptyset, G \vdash_O \overline{v}$ By sub-derivation of Df-Invk

$\emptyset, \emptyset, G \vdash_O \ell$ By sub-derivation of Df-Invk

$G \vdash_O H[\ell] \in lookup(\Sigma[\ell])$ By Lookup Lemma

$\forall v_k \in \overline{v} \quad \Sigma[v_k] = C_k{<}\overline{v'.d}{>}$

$\emptyset, \emptyset, G \vdash_O v_k$ By sub-derivation of Df-Invk

$G \vdash_O H[v_k] \in lookup(\Sigma[v_k])$ By Lookup Lemma

$E' = \langle H[\ell], H[\theta], H[v_k], Exp\rangle \in DE$ By subderivation of Aux-Import

$\Sigma[v_k] = C_v{<}\overline{v'.d}{>} \quad H[v_k] = O_v = \langle C_v{<}\overline{D_v}{>}\rangle \in DO$

$\forall v'_j.d_j \in \overline{v'.d}\ K[v'_j.d_j] = D_j \in rng(DD)$ By Df-Approx

$\Sigma[v_k] <: T'_k$ By sub-derivation of Df-Invk

$H; K; L_I; L_E \vdash H[v_k] \in irLookup(T'_k)$ By inversion of IR-Lookup

This proves (4).

Subcase $e_0 = e'_0$ that is $e = e'_0.m(\overline{e})$.

From IRC-RecvInvk:

$$\frac{\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e'_0; S'; H'; K'; L'_I; L'_E}{\begin{array}{c}\theta \vdash e_0.m(\overline{e}); S; H; K; L_I; L_E \rightsquigarrow_G \\ e'_0.m(\overline{e}); S'; H'; K'; L'_I; L'_E\end{array}}$$

$\theta \vdash e'_0; S; H; K; L_I; L_E \rightsquigarrow_G e''_0; S'; H'; K'; L'_I; L'_E$ By induction hypothesis

$\theta \vdash e'_0.m(\overline{e}); S; H; K; L_I; L_E \rightsquigarrow_G e''_0.m(\overline{e}); S'; H'; K'; L'_I; L'_E$ By IRC-RecvInvk

Take $e' = e''_0.m(\overline{e})$.

Subcase $e_0 = v$ that is $e = v.m(v_{1..i-1}, e_i, e_{i+1..n})$.

From IRC-ArgInvk:

$$\frac{\theta \vdash e_i; S; H; K; L_I; L_E \rightsquigarrow_G e'_i; S'; H'; K'; L'_I; L'_E}{\begin{array}{c}\theta \vdash v.m(v_{1..i-1}, e_i, e_{i+1..n}); S; H; K; L_I; L_E \rightsquigarrow_G \\ v.m(v_{1..i-1}, e'_i, e_{i+1..n}); S'; H'; K'; L'_I; L'_E\end{array}}$$

$\Gamma, \Upsilon, G \vdash_O e_i$ By sub-derivation of Df-Invk

$\theta \vdash e_i; S; H; K; L_I; L_E \rightsquigarrow_G e'_i; S'; H'; K'; L'_I; L'_E$ By induction hypothesis

$\theta \vdash v.m(v_{1..i-1}, e_i, e_{i+1..n}); S; H; K; L_I; L_E \rightsquigarrow_G$
$v.m(v_{1..i-1}, e'_i, e_{i+1..n}); S'; H'; K'; L'_I; L'_E$ By IRC-ArgInvk

Take $e' = v.m(v_{1..i-1}, e'_i, e_{i+1..n})$.

Case DF-CONTEXT : e = $\ell \triangleright e_0$. there are two subcases to consider, depending on whether e_0 is a value

Subcase e_0 **is a value** that is $e = \ell \triangleright v$.

From Ir-Context:

$$\frac{}{\theta \vdash \boxed{\ell \triangleright v; S}; H; K; L_I; L_E \rightsquigarrow_G \boxed{v;\ S}; H; K; L_I; L_E}$$

Then Ir-Context can apply. Take $e' = v$.

Subcase e_0 **is not a value** that is $e = \ell \triangleright e'_0$.

From IRC-CONTEXT:

$$\frac{\ell \vdash e; S; H; K; L_I; L_E \rightsquigarrow_G e'; S'; H'; K'; L'_I; L'_E}{\theta \vdash \ell \triangleright e; S; H; K; L_I; L_E \rightsquigarrow_G \ell \triangleright e'; S'; H'; K'; L'_I; L'_E}$$

$\ell \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e'_0; S'; H'; K'; L'_I; L'_E$	By induction hypothesis
$\theta \vdash \ell \triangleright e_0; S; H; K; L_I; L_E \rightsquigarrow_G \ell \triangleright e'_0; S'; H'; K'; L'_I; L'_E$	By IRC-CONTEXT
Take $e' = \ell \triangleright e'_0$.	

■

$$\frac{}{\theta \vdash e; S; H; K; L_I; L_E \rightsquigarrow^*_G e; S; H; K; L_I; L_E}[\text{Df-Reflex}]$$

$$\frac{\theta \vdash e; S; H; K; L_I; L_E \rightsquigarrow^*_G e''; S''; H''; K''; L''_I; L''_E \quad \theta \vdash e''; S''; H''; K''; L''_I; L''_E \rightsquigarrow_G e'; S'; H'; K'; L'_I; L'_E}{\theta \vdash e; S; H; K; L_I; L_E \rightsquigarrow^*_G e'; S'; H'; K'; L'_I; L'_E}[\text{Df-Trans}]$$

Figure 11: Reflexive, transitive closure of the instrumented evaluation relation

3.6 Theorem: Object Graph Soundness

$$\begin{array}{l} \textit{If} \\ \quad G = \langle DO, DD, DE \rangle \\ \quad DO, DD, DE \vdash (CT, e_{root}) \\ \quad \forall e,\ \theta_0 \vdash e; \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rightsquigarrow^*_G e; S; H; K; L_I; L_E \\ \quad \Sigma \vdash S \\ \textit{then} \\ \quad DO, DD, DE \vdash_{CT,H} \Sigma \\ \quad (S, H, K, L_I, L_E) \sim (DO, DD, DE) \end{array}$$

where $\rightsquigarrow^*_G$ relation is the reflexive and transitive closure of $\rightsquigarrow_G$ relation (Fig. 11). θ_0 is the location of the first object instantiated by e_{root}.

To prove the Object Graph Soundness theorem, we need to show:

(1) $DO, DD, DE \vdash_{CT,H} \Sigma$

(2) $(S, H, K, L_I, L_E) \sim (DO, DD, DE)$

Proof: The proof is by induction on the $\rightsquigarrow^*_G$ relation. There are two cases to consider: [1]

Case Df-Reflex :

Since $S = \emptyset$:

$(S, H, K, L_I, L_E) \sim G$

Immediately, from Df-Sigma store constraint with $S = \emptyset$:

$DO, DD, DE \vdash_{CT,H} \Sigma$

Case Df-Trans :

By assumption:

$\theta_0 \vdash e; \emptyset; \emptyset; \emptyset; \emptyset; \emptyset \rightsquigarrow^*_G e; S; H; K; L_I; L_E$

Since $S = \emptyset$:

$(\emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \sim G$

By inversion of Df-Trans:

$\theta_0 \vdash e; \emptyset; \emptyset; \emptyset; \emptyset; \emptyset \rightsquigarrow^*_G e'; S'; H'; K'; L'_I; L'_E$

By induction hypothesis:

$(S'; H'; K'; L'_I; L'_E) \sim G$

By inversion of Df-Trans:

$\theta_0 \vdash e'; S'; H'; K'; L'_I; L'_E \rightsquigarrow_G e; S; H; K; L_I; L_E$

By preservation:

$(S; H; K; L_I; L_E) \sim G$

By assumption:

[1] The soundness proof follows similar steps to the one of points-to analysis [1].

$\theta_0 \vdash e; \emptyset; \emptyset; \emptyset; \emptyset; \emptyset \rightsquigarrow^*_G e; S; H; K; L_I; L_E$
Since $S = \emptyset$:
$(\emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \sim G$
By inversion of Df-Trans:
$\theta_0 \vdash e; \emptyset; \emptyset; \emptyset; \emptyset; \emptyset \rightsquigarrow^*_G e'; S'; H'; K'; L'_I; L'_E$
By induction hypothesis:
$DO, DD, DE \vdash_{CT,H'} \Sigma'$
By inversion of Df-Trans:
$\theta_0 \vdash e'; S'; H'; K'; L'_I; L'_E \rightsquigarrow_G e; S; H; K; L_I; L_E$
By preservation:
$DO, DD, DE \vdash_{CT,H} \Sigma$

■

3.6.1 Lemmas

To prove the Progress and Preservation theorems, we use the following lemmas. We intended to use the first four lemmas, (i.e. the import and export lemmas) in the Progress theorem proof. However, we complete the Progress proof without their use. We keep them for backward compatibility with the previous version of this report.

Df-Substitution Lemma.
If
$\Gamma \cup \{\overline{x} : \overline{T_f}\}, \Sigma, \theta \vdash e : T$
$\Gamma \cup \{\overline{x} : \overline{T_f}\}, \Upsilon, G \vdash_O e$
$\Gamma, \Sigma, \theta \vdash \overline{v} : \overline{T_a}$ where $\overline{T_a} <: [\overline{v}/\overline{x}]\overline{T_f}$
then
$\Gamma, \Sigma, \theta \vdash [\overline{v}/\overline{x}]e : T'$ *for some* $T' <: [\overline{v}/\overline{x}]T$
$\Gamma, \Upsilon, G \vdash_O [\overline{v}/\overline{x}]e$
Proof: By induction on the $\Gamma, \Upsilon, G \vdash_O e$ relation. ■

Df-Weakening Lemma.
If
$\Gamma, \Upsilon, G \vdash_O e$
then
$\Gamma, \Upsilon \cup \{C\overline{<D>}\}, G, \vdash_O e$
Proof: By induction on the $\Gamma, \Upsilon, G \vdash_O e$ relation. ■

Df-Strengthening Lemma.
If
$\Gamma, \emptyset, G \vdash_O \texttt{new}\ C\overline{<p>}(v)$
$\forall i \in 1..|\overline{p}| \quad D_i = DD[(O, p_i)]$
$\Gamma, \Upsilon \cup \{C\overline{<D>}\}, G, \vdash_{O'} e'$
then
$\Gamma, \Upsilon, G, \vdash_O e$
Proof: By induction on the $\Gamma, \Upsilon, G \vdash_O e$ relation. ■

Df-Domains Lemma.
If
$\emptyset, \Sigma, \theta \vdash e : T$
$\Sigma \vdash S$
$G \vdash_{CT,H} \Sigma$
$\emptyset, \emptyset, G \vdash_O$ *new* $C\overline{<p>}(\overline{v})$
$(S, H, K, L_I, L_E) \sim (DO, DD, DE)$
$G \vdash_O ddomains(C, O_C)$

$\forall i \in 1..|\overline{p}| \quad D_i = DD[(O_C, qual(p_i))]$
$O_C = \langle C\!<\!\overline{D}\!>\rangle \quad \{O_C\} \subseteq DO$
then
$\forall d_j \in domains(C\!<\!\overline{p}\!>) \quad D_j = DD[(O_C, C::d_j)]$

Proof: By induction on the $G \vdash_O ddomains(C, O_C)$ relation. ■

Lookup Lemma

If $\emptyset, \Sigma, \theta \vdash \ell : \Sigma[\ell]$
$\Sigma \vdash S$
$G \vdash_{CT,H} \Sigma$
$(S, H, K, L_I, L_E) \sim (DO, DD, DE)$
then
$G \vdash_{H[\theta]} H[\ell] \in lookup(\Sigma[\ell])$

Proof:

$\Sigma[\ell] = C\!<\!\overline{\ell'.d}\!>$ By $\Sigma \vdash S$
$H[\ell] = \langle C_\ell\!<\!\overline{D}\!>\rangle$

To Show:

Take $H[\theta] = C_\theta\!<\!\overline{D_\theta}\!>$ $\theta \in domain(H)$
$\forall \ell'_j.d_j \in \overline{\ell'.d}, G \vdash_{H[\theta]} D'_j \in findD(C_\theta :: qual(\ell'_j.d_j)) \quad D'_j = D_j \quad D_j \in \overline{D} \quad K[\ell'_j.d_j] = D_j$

Proof by generalized induction.

We first prove two base cases: (1) when d_j is a locally declared domain and (2) when d_j is a locally declared domain of ℓ in the presence of recursive types.

Case $\ell'_j = \theta$. Local domains.

$H[\theta] = \langle C_\theta\!<\!\overline{D_\theta}\!>\rangle \in DO$
$\forall d_j \in domains(\Sigma[\theta]) \quad K[\theta.d_j] = D_j = \langle D_{id_j}, C_\theta::d_j\rangle \quad \{(H[\theta], C_\theta::d_j) \mapsto D_j\} \in DD$ By Df-Approx
$D'_j = DD[(H[\theta], C_\theta::d_j)] = K[\theta.d_j]$ By above
$G \vdash_{H[\theta]} D'_j \in findD(C_\theta :: \texttt{this}.d_j)$ By inversion of Aux-FindThis
$G \vdash_{H[\theta]} D'_j \in findD(C_\theta :: qual(\ell'_j.d_j))$ Since$\ell'_j.d_j$
$D'_j = K[\theta.d_j] = K[\ell'_j.d_j] = D_j$ By hypothesis and $\ell'_j = \theta$

Case $\ell'_j = \ell$. Recursive types.

$H[\ell] = O_C = \langle C\!<\!\overline{D}\!>\rangle \in DO$
and $\forall \ell'_j.d_j \in \overline{\ell'.d}\ K[\ell'_j.d_j] = D_j = \langle D_{id_j}, qual(\ell'_j.d_j)\rangle \in rng(DD)$
and $\forall d_i \in domains(C\!<\!\overline{\ell'.d}\!>)$
$K[\ell.d_i] = D_i = \langle D_{id_i}, C::d_i\rangle\ \{(O_C, C::d_i) \mapsto D_i\} \in DD$
$DD[(H[\ell], \ell.d_j)] = D_j = K[\ell.d_j]$ By inversion of Df-New

The induction step.

Case $\ell'_j = n$. d_j is a public domain of n, but not a local domain of θ.

Assume that $\forall \ell' \in domain(H), G \vdash_{H[\theta]} H[\ell'] \in lookup(\Sigma[\ell'])$ but $\ell \notin dom(H)$. Where *domain* means the set of all keys stored in H.

To show:
$K[n.d_j] = D_j \quad G \vdash_{H[\theta]} D_j \in findD(C_\theta{::}qual(n.d_j))$

$n : \Sigma[n]$	
$DD[(H[n], n.d_j)] = D_j = K[n.d_j]$	By Df-Approx, since $d_j \in domains(\Sigma[n])$
$G \vdash_{H[\theta]} H[n] \in lookup(\Sigma[n])$	By induction hypothesis
$G \vdash_{H[\theta]} D_j \in findD(C_\theta{::}qual(n.d_j))$	By inversion of AUX-FIND-PUBLIC

$G \vdash_{H[\theta]} H[\ell] \in lookup(\Sigma[\ell])$ By induction

■

Params Lemma.

If $\emptyset, \Sigma, \theta \vdash \ell : \Sigma[\ell]$
$\Sigma \vdash S$
$\emptyset, \emptyset, G \vdash_O$ `new` $C\langle\overline{\ell'.d}\rangle(\overline{v})$
$G \vdash_{CT,H} \Sigma$
$(S, H, K, L_I, L_E) \sim (DO, DD, DE)$
$\ell \in domain(S) \quad H[\ell] = C\langle\overline{D}\rangle \quad \Sigma[\ell] = C\langle\overline{\ell'.d}\rangle \quad S[\ell] = C\langle\overline{\ell'.d}\rangle(\overline{v})$
$CT(C) =$ `class` $C\langle\overline{\alpha}\rangle$ .... $\{\ \overline{T}\ \overline{f};\ \overline{dom};\ \ldots;\ \overline{md};\ \}$
then $\forall \ell'_i.d_i \in \overline{\ell'.d} \quad DD[(H[\ell], qual(\ell'_i.d_i))] = DD[(H[\ell], C{::}\alpha_i)] = D_i = K[\ell'_i.d_i]$

Proof:
By induction on the $G \vdash_O dparams(C, O_C)$ relation using subderivation of DF-NEW IR-NEW.

■

Differences with points-to analysis. Our formalization is similar to the one for the points-to analysis [1, Section 3.2 and 3.3]. The two analyses create the same object-domain hierarchy, but the analysis in this paper shows additional edges that are missing from an OOG with points-to edges. The key differences in the formalization deal with generating the dataflow edges and the soundness proof. The previous work made a simplistic assumption about dataflow edges, namely that they can be approximated by reverting points-to edges, but the assumption turned out to be imprecise.

4 Extensions `lent` and `unique`

A variable declared `unique` refers to an object to which there is only one reference, such as a newly created object, and can be passed linearly from one domain to another. For example, in a factory method the actual domain of the object created by the factory is known only by the client code. Therefore, the method returns a reference declared as `unique`, which may be pass to an actual domain. Fields are not usually declared `unique` because a field read expression is not usually followed by an object destruction. A variable declared `lent` refers to an object that is temporarily lent from one domain to another as long as an object in the second domain does not create a persistent reference to the borrowed object, e.g., by storing it in a field. Only method formal parameters and local variables can be `lent`.

The client code can assign a `unique` variable to a variable in a locally declared domain, in a domain parameter, in `shared`, or `lent`. The converse is prohibited: a variable declared `lent` can only be assigned to other variables declared `lent`. Therefore, variables declared `unique` are universal sources and can flow to all other variables, while variables declared `lent` are universal sinks [3]. Because a local variable that refers to a newly created object is a source, it cannot be declared `lent`. Since the OOG is an over-approximation, we extract all such objects such that the OOG has a representative for all the objects that are created in any possible execution.

Several variables declared in different domains can alias the same object. An extraction analysis resolves `unique` by finding a destination variable that is declared in either a domain parameter, a locally declared domain, `shared` or `lent` that the variable declared `unique` may flow into. Similarly, for a variable declared `lent`, the extraction analysis resolves `lent` by finding a source variable that is declared in either a domain parameter, a locally declared domain, `shared` or `unique`.

Flow Object. We define a *flow object* to be an object that is in a domain that corresponds to a `unique` annotation that the analysis cannot resolve to an actual domain. One flow object can be referred to from zero or more dataflow edges, or can be the source or the destination of a dataflow edge. For example, a `unique` variable that refers to a newly created object can be passed as a `lent` parameter of a method; the called method can pass on the object as a `lent` parameter to other methods but cannot return it or store it in a field, and the analysis cannot find an actual domain.

In the following, we revise the FDJ abstract syntax to support `lent` and `unique`. To resolve `lent`and `unique`, the analysis needs to compute a value flow graph, and reason about information flow between variables. To keep track of all the intermediate values, and easy the reasoning about complex expressions,

we revise the FDJ abstract syntax using three-address code. Next, we extend the data type declarations to include the declaration of the value flow graph. We also extend the formalization to describe how the analysis attempts to resolve `lent` and `unique` and creates flow objects.

4.1 Revised Abstract Syntax

We formally describe our static analysis using a three-address code language based on Featherweight Domain Java (FDJ), which models a core of the Java language with Ownership Domains [2]. To keep the language easier to reason about, FDJ ignores advanced Java language constructs such as interfaces and static code.

In Fig. 12, the meta-variable C ranges over class names; T ranges over types; f ranges over field names. As a shorthand, an overbar denotes a sequence. Γ maps variables to their types a store S maps locations ℓ to their contents; the set of variables includes the distinguished variable `this` of type T_{this} used to refer to the receiver of a method; the result of the computation is a location ℓ, which is sometimes referred to as a value v; $S[\ell]$ denotes the store entry of ℓ; $S[\ell, i]$ denotes the value of i^{th} field of $S[\ell]$; $S[\ell \mapsto C<\overline{\ell'.d}>(\overline{v})]$ denotes adding an entry for location ℓ to S; α and β range over formal domain parameters; the expression form $\ell \triangleright e$ represents a method body e executing with a receiver ℓ; a program is a tuple (CT, e_{root}) of a class table and an expression that starts the program.

The syntax includes `lent` and `unique`. According to FDJ [3], only the first domain parameter (`owner` domain) of a type can be `lent` or `unique` because the class `Object` takes one domain parameter, and according to *cdef*, only the first domain is mandatory for every type. An object creation expression can have the first domain parameter `unique` (but not `lent`), thus the syntax uses the meta-variable A for a `new` expression. A type T consists of a class C parameterized with a list of domains that are of the following form: a domain parameter α, a declared domain $n.d$, or `shared`. The syntax also includes the meta-variable T_A for types in which the `owner` domain can also be `unique`, and T_B for types in which the `owner` domain can be also `lent`. For example, the type of fields in the class definition cannot be `lent` because a borrowed object cannot be stored in a field, hence the field type is T_A. On the other hand, the first domain in the type of a parameter in a method declaration can be `lent` or `unique` and the parameter type is T_B.

4.2 Extended Data Type Declarations

The analysis extracts a hierarchical object graph (OGraph) with nodes that represent abstract objects (OObjects), groups of objects (ODomains), and OEdges that represent dataflow communication between

$$
\begin{array}{rcl}
CT & ::= & \overline{cdef} \qquad \textit{table of class declarations} \\
cdef & ::= & \texttt{class}\ C\texttt{<}\overline{\alpha},\overline{\beta}\texttt{>}\ \texttt{extends}\ C'\texttt{<}\overline{\alpha}\texttt{>} \\
 & & \{\ \overline{dom};\ \ \overline{T_A\ f};\ \ C(\overline{T'_A\ f'},\overline{T_A\ f}) \\
 & & \{\texttt{super}(f');\texttt{this}.\overline{f}=\overline{f};\}\ \ \overline{md}\ \} \qquad \textit{class decl.} \\
dom & ::= & [\texttt{public}]\ \texttt{domain}\ d; \qquad \textit{domain decl.} \\
md & ::= & T_A\ ret\ m(\overline{T_B\ x})\ \ T_{this}\ \{ret = e_R; \texttt{return}\ ret;\} \\
 & & \qquad \textit{method decl.} \\
e & ::= & \mid x = \texttt{new}\ C\texttt{<}A,\overline{p}\texttt{>}(\overline{y}) \\
 & & \mid x = y.f\ \mid x.f = y \mid x = y\ \mid x = r.m(\overline{y}) \\
 & & \mid \ell\ \mid \ell \triangleright e \qquad \textit{expressions} \\
n & ::= & x\ \mid\ v \qquad \textit{values or variable names} \\
p & ::= & \alpha\ \mid\ n.d\ \mid\ \texttt{SHARED} \qquad \textit{domain name} \\
A & ::= & \texttt{unique}\ \mid\ p \qquad \textit{domain may be unique} \\
B & ::= & \texttt{lent}\ \mid\ A \qquad \textit{domain may be lent or unique} \\
T & ::= & C\texttt{<}\overline{p}\texttt{>} \qquad \textit{precise type} \\
T_A & ::= & C\texttt{<}A,\overline{p}\texttt{>} \qquad \textit{owner domain may be unique} \\
T_B & ::= & C\texttt{<}B,\overline{p}\texttt{>} \qquad \textit{owner domain may be lent or unique} \\
v,\ell,\theta & \in & \textit{locations} \\
x,y,r,a & \in & \textit{variables} \\
S & ::= & \ell\ \rightarrow\ C\texttt{<}\overline{\ell'.d}\texttt{>}(\overline{v}) \qquad \textit{location store} \\
\Sigma & ::= & \ell\ \rightarrow\ T \qquad \textit{store typing} \\
\Gamma & ::= & x\ \rightarrow\ T_B \qquad \textit{type environment}
\end{array}
$$

Figure 12: FDJ syntax, extended using `lent` and `unique` [2]

$$
\begin{array}{ll}
G \in \mathsf{OGraph} & ::= \langle\ \mathbf{Objects} = DO,\ \mathbf{DomainMap} = DD, \mathbf{Edges} = DE\ \rangle \\
D \in \mathsf{ODomain} & ::= \langle\ \mathbf{Id} = D_{id},\ \mathbf{Domain} = C{::}d\ \rangle \\
O \in \mathsf{OObject} & ::= \langle\ \mathbf{Type} = C\texttt{<}\overline{D}\texttt{>}\ \rangle \\
E \in \mathsf{OEdge} & ::= \langle\ \mathbf{From} = O_{src},\ \mathbf{To} = O_{dst}, \\
 & \quad \mathbf{Label} = O_{label}, \mathbf{Flag} = \mathit{Imp}\ |\mathit{Exp}\ \rangle \\
DD & ::= \emptyset\ \mid\ DD \cup \{\ (O, C{::}d) \mapsto D\ \} \qquad \textit{Domain map} \\
DO & ::= \emptyset\ \mid\ DO \cup \{\ O\ \} \qquad \textit{Dataflow Object} \\
DE & ::= \emptyset\ \mid\ DE \cup \{\ E\ \} \qquad \textit{Dataflow Edge} \\
\Upsilon & ::= \emptyset\ \mid\ \Upsilon \cup \{\ C\texttt{<}\overline{D}\texttt{>}\ \} \qquad \textit{Stack of visited OObjects} \\
FG & ::= \emptyset\ \mid\ FG \cup \{\ (O_{src}, x, B_{src})\ \overset{annot}{\rightsquigarrow}\ (O_{dst}, y, B_{dst})\ \} \\
 & \quad \textit{Value Flow Graph} \\
annot & ::= (_i\ \mid\ )_i\ \mid \bullet\ \mid \star \qquad \textit{value flow annotations}
\end{array}
$$

Figure 13: Data type of OGraph, and value flow graph

abstract objects (Fig. 13). Each OEdge is a directed edge from a source O_{src} to a destination O_{dst}. The label of an OEdge is the OObject that the dataflow refers to. The flag states whether the OEdge represents an import or an export dataflow communication. The OGraph is a multi-graph, where multiple edges with different labels might exist between the same source and destination.

A value flow graph (FG) represents information flow between two variables x and y. A node in FG is a triplet (O, x, B) that denotes a variable x part of an expression that the analysis interprets in the context of O, and x is of a type T_B where the owner domain B is a domain p, `unique`, or `lent` as defined in Fig. 12. A

shorthand the notation $(O, \overline{x}, \overline{B})$ means a list of j triplets (O, x_1, B_1), (O, x_2, B_2), ..., (O, x_j, B_j) An edge in FG has an *annot* label to track if the value flow is due to a method invocation ($\overset{(_i}{\rightsquigarrow}$), a method return ($\overset{)_i}{\rightsquigarrow}$). The label $\bullet$ denotes an empty annotation on a value flow edge due an assignment ($\rightsquigarrow$). The label $\star$ denotes an information flow due to a field write. The label on value flow edges tracks call-site sensitivity [5, 4]. By considering O as a part of the node, the flow analysis is also domain-sensitive and has different nodes for the same variable x analyzed in different contexts.

4.3 Extended Formalization

The extraction analysis starts by creating the OObject O_{world} and its owning ODomain $D_{\texttt{SHARED}}$, which constitutes the root of the OGraph. It uses the following initial values:

$$D_{\texttt{SHARED}} = \langle D_0, ::\texttt{SHARED} \rangle, O_{world} = \langle C_{dummy} < . > \rangle$$
$$FG_0 = \emptyset,\ DO_0 = \{O_{world}\}$$
$$DD_0 = \{(O_{world},\ ::\texttt{shared}) \mapsto D_{\texttt{shared}}\}, DE_0 = \emptyset$$

Then, the analysis abstractly interprets e_{root} in the context of O_{world}:

$$\emptyset, \emptyset, FG_0, DO_0, DD_0, DE_0 \vdash_{O_{world}} e_{root}$$

We describe the analysis using rules of the following form:

$$\Gamma, \Upsilon, FG, G \vdash_O e$$

The rules consider the types of the variable in scope as provided by Γ, a stack Υ of visited OObjects to avoid non-termination, the value flow graph FG and the OGraph G, from which we can refer to its constituent parts DO, DD, and DE. The O subscript on the turnstile captures the context-sensitivity, and represents the context that the analysis uses to abstractly interpret an expression e.

The analysis uses expressions given in the form of three-address code, where x represents the left-hand-side of the expression. In *Df-New*, the analysis interprets an object allocation expression in the context of O. The analysis first ensures that DO contains an OObject O_C for the newly allocated object. Then, using *dparams*, *Df-New* ensures that each of the actual domain parameters p_i maps to an actual domain D_i in the context of O, where the corresponding formal domain parameter α_i maps to the same D_i but in the

$$\frac{\begin{array}{c} CT(C) = \texttt{class}\ C\texttt{<}\overline{\alpha},\overline{\beta}\texttt{>}\ \texttt{extends}\ C'\texttt{<}\overline{\alpha}\texttt{>}\ \{\ \overline{T}\ \overline{f};\ \overline{dom};\ \ldots;\ \overline{md};\ \} \qquad G = \langle DO, DD, DE\rangle \\ O = C_{\texttt{this}}\texttt{<}\overline{D_O}\texttt{>} \qquad \forall i \in 1..|\overline{p}| \qquad FG, G \vdash_O D_i \in findD(C_{\texttt{this}}::p_i) \\ O_C = \langle\ C\texttt{<}\overline{D}\texttt{>}\ \rangle \qquad \{O_C\} \subseteq DO \\ dparams(C, O_C) \qquad \{(O_C, qual(p_i)) \mapsto D_i\} \subseteq DD \qquad G \vdash_O ddomains(C, O_C) \\ \Gamma[\overline{a}] = \overline{T_a} \qquad \{(O, x, p_1) \rightsquigarrow (O_C, \texttt{this}, \alpha_0), (O, \overline{a}, owner(\overline{T_a})) \rightsquigarrow (O_C, \texttt{this}.\overline{f}, owner(\overline{T}))\} \subseteq FG \\ \forall m \in \overline{md}.\ mbody(m, C\texttt{<}\overline{p}\texttt{>}) = (\overline{x} : \overline{T},\ e_R) \\ C\texttt{<}\overline{D}\texttt{>} \notin \Upsilon \Longrightarrow \{\overline{x} : \overline{T},\ \texttt{this} : C\texttt{<}\overline{p}\texttt{>}\}, \Upsilon \cup \{C\texttt{<}\overline{D}\texttt{>}\}, FG, G \vdash_{O_C} e_R \end{array}}{\Gamma, \Upsilon, FG, G \vdash_O x = \texttt{new}\ C\texttt{<}\overline{p}\texttt{>}(\overline{a})}\ [\textsc{Df-New}]$$

$$\frac{\begin{array}{c} CT(C) = \texttt{class}\ C\texttt{<}\overline{\alpha},\overline{\beta}\texttt{>}\ \texttt{extends}\ C'\texttt{<}\overline{\alpha}\texttt{>}\ \{\ \overline{T}\ \overline{f};\ \overline{dom};\ \ldots;\ \overline{md};\ \} \qquad G = \langle DO, DD, DE\rangle \\ O = C_{\texttt{this}}\texttt{<}\overline{D_O}\texttt{>} \qquad FG, G \vdash_O D_1 \in uniqueDomains(C) \qquad \forall i \in 2..|\overline{p}|\ FG, G \vdash_O D_i \in findD(C_{\texttt{this}}::p_i) \\ O_C = \langle\ C\texttt{<}\overline{D}\texttt{>}\ \rangle \qquad \{O_C\} \subseteq DO \\ dparams(C, O_C) \qquad \{(O_C, qual(p_i)) \mapsto D_i\} \subseteq DD \qquad G \vdash_O ddomains(C, O_C) \\ \Gamma[\overline{a}] = \overline{T_a} \qquad \{(O, x, p_1) \rightsquigarrow (O_C, \texttt{this}, \alpha_0), (O, \overline{a}, owner(\overline{T_a})) \rightsquigarrow (O_C, \texttt{this}.\overline{f}, owner(\overline{T}))\} \subseteq FG \\ \forall m \in \overline{md}.\ mbody(m, C\texttt{<}\overline{p}\texttt{>}) = (\overline{x} : \overline{T},\ e_R) \\ C\texttt{<}\overline{D}\texttt{>} \notin \Upsilon \Longrightarrow \{\overline{x} : \overline{T},\ \texttt{this} : C\texttt{<}\overline{p}\texttt{>}\}, \Upsilon \cup \{C\texttt{<}\overline{D}\texttt{>}\}, FG, G \vdash_{O_C} e_R \end{array}}{\Gamma, \Upsilon, FG, G \vdash_O x = \texttt{new}\ C\texttt{<}\texttt{unique}, \overline{p}\texttt{>}(\overline{a})}\ [\textsc{Df-New-Unique}]$$

$$\frac{\begin{array}{c} CT(C_r) = \texttt{class}\ C_r\texttt{<}\overline{\alpha},\overline{\beta}\texttt{>}\ \texttt{extends}\ C'_r\texttt{<}\overline{\alpha}\texttt{>}\ \{\ \overline{T}\ \overline{f};\ \overline{dom};\ \ldots;\ \overline{md};\ \} \qquad G = \langle DO, DD, DE\rangle \\ \Gamma[r] = T_r = C_r\texttt{<}\overline{p}\texttt{>} \qquad (T_k\ f_k) \in fields(T_r) \qquad FG, G \vdash_O import(T_r, T_k) \\ FG, G \vdash_O O_r \in lookup(T_r) \qquad (T'_k\ f_k) \in \overline{Tf} \\ \{(O, r, owner(T_r)) \rightsquigarrow (O_r, \texttt{this}, \alpha_0), (O_r, f_k, owner(T'_k)) \rightsquigarrow (O, x, owner(T_k))\} \subseteq FG \end{array}}{\Gamma, \Upsilon, FG, G \vdash_O x = r.f_k}\ [\textsc{Df-Read}]$$

$$\frac{\begin{array}{c} CT(C_x) = \texttt{class}\ C_x\texttt{<}\overline{\alpha},\overline{\beta}\texttt{>}\ \texttt{extends}\ C'_x\texttt{<}\overline{\alpha}\texttt{>}\ \{\ \overline{T}\ \overline{f};\ \overline{dom};\ \ldots;\ \overline{md};\ \} \qquad G = \langle DO, DD, DE\rangle \\ \Gamma[x] = T_x = C_x\texttt{<}\overline{p}\texttt{>} \qquad (T_k\ f_k) \in fields(T_x) \qquad \Gamma[r] = T_r \qquad T_r <: T_k \qquad FG, G \vdash_O export(T_x, T_r) \\ FG, G \vdash_O O_x \in lookup(C_x\texttt{<}\overline{p}\texttt{>}) \qquad (T'_k\ f_k) \in \overline{Tf} \qquad \{(O, r, owner(T_r)) \overset{\star}{\rightsquigarrow} (O_x, f_k, owner(T'_k))\} \subseteq FG \end{array}}{\Gamma, \Upsilon, FG, G \vdash_O x.f_k = r}\ [\textsc{Df-Write}]$$

$$\frac{\begin{array}{c} CT(C) = \texttt{class}\ C\texttt{<}\overline{\alpha},\overline{\beta}\texttt{>}\ \texttt{extends}\ C'\texttt{<}\overline{\alpha}\texttt{>}\ \{\ \overline{T}\ \overline{f};\ \overline{dom};\ \ldots;\ \overline{md};\ \} \qquad G = \langle DO, DD, DE\rangle \\ \Gamma[r_0] = C\texttt{<}\overline{p}\texttt{>} \qquad mtype(m, C\texttt{<}\overline{p}\texttt{>}) = \overline{T} \to T_R \qquad FG, G \vdash_O import(C\texttt{<}\overline{p}\texttt{>}, T_R) \\ \Gamma[\overline{a}] = \overline{T_a} \qquad \overline{T_a} <: \overline{T} \qquad FG, G \vdash_O export(C\texttt{<}\overline{p}\texttt{>}, \overline{T_a}) \\ FG, G \vdash_O O_r \in lookup(C\texttt{<}\overline{p}\texttt{>}) \qquad T'_R\ ret\ m(\overline{T_B}\ \overline{x})\ \ T_{this}\ \{ret = e_R; \texttt{return}\ ret;\} \in \overline{md} \qquad i = fresh_i(O, x_0 = r_0.m(\overline{a})) \\ \Gamma[\overline{x}] = \overline{T_x} \qquad \{(O, r_0, p_0) \overset{(_i}{\rightsquigarrow} (O_r, \texttt{this}, \alpha_0), (O, \overline{a}, owner(\overline{T_a})) \overset{(_i}{\rightsquigarrow} (O_r, \overline{x}, owner(\overline{T_x}))\} \subseteq FG \\ \Gamma[ret] = T'_R \qquad \{(O_r, ret, owner(T'_R)) \overset{)_i}{\rightsquigarrow} (O, x_0, owner(T_R))\} \subseteq FG \end{array}}{\Gamma, \Upsilon, FG, G \vdash_O x_0 = r_0.m(\overline{a})}\ [\textsc{Df-Invk}]$$

$$\frac{\Gamma[r] = T_r \qquad \Gamma[x] = T_x \qquad \{(O, r, owner(T_r)) \rightsquigarrow (O, x, owner(T_x))\} \subseteq FG}{\Gamma, \Upsilon, FG, G \vdash_O x = r}\ [\textsc{Df-Assign}]$$

Figure 14: Static semantics of the extraction analysis. We highlight the parts that construct the value flow graph.

context of O_C (Fig. 14). *Df-New* also ensures that the object hierarchy is created such that new ODomains are created for each domain declarations in C according to the auxiliary judgment *ddomains*. Both *dparams* and *ddomains* are recursive auxiliary judgments that consider inheritance (Fig. 16), i.e., the domain may be declared by a class C' that C extends. The base case for the recursion is the class `Object`.

The rule *Df-New* also ensures that FG includes an edge from x to `this` and edges from each of the

object allocation arguments $\overline{a}$ to the corresponding fields `this`.$\overline{f}$ (Fig. 14). Next, *Df-New-Unique* handles the case when the owner is `unique`. The rule is similar to *Df-New*, except that it uses the auxiliary judgment *solveUnique* to find the actual owner domain of O_C. We describe *solveUnique* later in Fig. 15.

The rules *Df-Read*, *Df-Write* and *Df-Invk* ensure that import and export edges are created using the context O, the receiver O_r and the dataflow OObject as determined by *lookup* auxiliary judgment (Fig. 15). If the owner domain parameter of T_{label} is `unique`, and *lookup* cannot find an actual ODomain, the analysis ensures that an OObject is created in a fresh ODomain, as a child of O. All the child OObjects of such an ODomain are flow objects.

Next, *Df-Read*, *Df-Write*, *Df-Invk*, and *Df-Assign* ensure value flow edges are created in FG. For example, *Df-Read*, ensures that the flow graph contains one edge from the receiver r in the context O to the context variable `this` in the context O_r, and a second edge from the field f_k in the context of O_r to x in the context of O. For a method invocation, the rule *Df-Invk* adds annotations to the value flow edges that correspond to the arguments of the invocation and to the return value. For *Df-Read*, *Df-Write*, and *Df-Invk*, the context OObject of the source is different from the context OObject of the destination of a flow edge. On the other hand, for *Df-Assign*, the context OObject remains unchanged for the source and destination.

The precision of the analysis is provided by the auxiliary judgments *Df-Lookup* (Fig. 15). For a given type $C'<\overline{p}>$ that includes a list of actual domain parameters, *lookup* returns those OObjects O_k in DO such the class of O_k is C' or one of its subclasses and each domain D_i of O_k corresponds to D'_i, the domain associated with the pair $(O, C_{this}::p_i)$ in DD. The second condition increases the precision of the analysis, because *lookup* selects all the objects in DO of a class C' or a subclass thereof that are in reachable domains, as opposed to all the objects of a given class in DO.

We introduce the rules *Df-Lookup-Lent*, and *Df-Lookup-Unique* for *lookup* where the owner domain is `lent` or `unique`. These rules use *solveLent* and *solveUnique* to determine the actual domain p' and the context O' where p' is defined. We need to include the context O' in the result to be able to determine the actual domain D'_1 corresponding to `lent` or `unique` because the context might be different from the current context O. For example, a class might be instantiated in the context of O where the owner is `unique`. Next, the reference x in the context O is assigned to y and in another context O'. The destination y is declared in an actual domain p'. To determine the actual ODomain for the domain parameter p', the extraction analysis uses the context of y, namely O', not the context of x.

The analysis is sound. Flow objects maintain the unique representative invariant since the analysis creates a fresh ODomain for each flow object. Multiple dataflow edges can then refer to the same flow object.

$$\frac{G = \langle DO, DD, DE\rangle \quad O = C_{\texttt{this}}\overline{<D>} \quad O_k \in DO \quad O_k = \langle C\overline{<D>}\rangle \quad C <: C' \quad \forall i \in 1..|\overline{p'}| \quad FG, G \vdash_O D'_i \in findD(C_{\texttt{this}}::p'_i) \quad D'_i = D_i}{FG, G \vdash_O O_k \in lookup\ (C'\overline{<p'>})}\ [\textsc{Df-Lookup}]$$

$$\frac{\begin{array}{c} G = \langle DO, DD, DE\rangle \quad O = C_{\texttt{this}}\overline{<D>} \quad O_k \in DO \quad O_k = \langle C<D_1, \overline{D}>\rangle \quad C <: C' \\ \forall i \in 2..|\overline{p'}| \quad FG, G \vdash_O D'_i \in findD(C_{\texttt{this}}::p'_i) \quad D'_i = D_i \\ FG \vdash (O', x, p') \in solveLent(O, C') \quad O' = C'_{\texttt{this}}\overline{<D'>} \\ A = \texttt{unique} \implies FG \vdash (O'', x, \texttt{unique}) \in findSrcUnique(O', C') \wedge D'_1 = DD[(O'', C'::\texttt{unique})] \wedge D'_1 = D_1 \\ A = p' \implies FG, G \vdash_{O'} D'_1 \in findD(C'_{\texttt{this}}::p') \wedge D'_1 = D_1 \end{array}}{FG, G \vdash_O O_k \in lookup\ (C'\overline{<\texttt{lent}, p'>})}\ [\textsc{Df-Lookup-lent}]$$

$$\frac{\begin{array}{c} G = \langle DO, DD, DE\rangle \quad O = C_{\texttt{this}}\overline{<D>} \quad O_k \in DO \quad O_k = \langle C<D_1, \overline{D}>\rangle \quad C <: C' \\ \forall i \in 2..|\overline{p'}| \quad FG, G \vdash_O D'_i \in findD(C_{\texttt{this}}::p'_i) \quad D'_i = D_i \\ FG \vdash (O', y, A) \in solveUnique(O, C') \quad O' = \langle C'_{\texttt{this}}\overline{<D'>}\rangle \\ A = \texttt{unique} \implies FG \vdash (O'', x, \texttt{unique}) \in findSrcUnique(O', C') \wedge D'_1 = DD[(O'', C'::\texttt{unique})] \wedge D'_1 = D_1 \\ A = p' \implies FG, G \vdash_{O'} D'_1 \in findD(C'_{\texttt{this}}::p') \wedge D'_1 = D_1 \end{array}}{FG, G \vdash_O O_k \in lookup\ (C'\overline{<\texttt{unique}, p'>})}\ [\textsc{Df-Lookup-Unique}]$$

$$\frac{\begin{array}{c} FG_P = propagateAll(FG) \\ (O, s, \texttt{unique}) \rightsquigarrow (O', y, \texttt{unique}) \in FG_P \quad s : C \quad C <: C' \quad \nexists (O'', x, \texttt{unique}) \rightsquigarrow (O, s, \texttt{unique}) \in FG_P \end{array}}{FG \vdash (O, s, \texttt{unique}) \in findSrcUnique(O', C')}\ [\textsc{Aux-Find-Unique}]$$

$$\frac{FG_P = propagateAll(FG) \quad (O, x, \texttt{unique}) \rightsquigarrow (O', y, A) \in FG_P \quad y : C \quad C <: C'}{FG \vdash (O', y, A) \in solveUnique(O, C')}\ [\textsc{Aux-Resolve-Unique}]$$

$$\frac{FG_P = propagateAll(FG) \quad (O', x, A) \rightsquigarrow (O, y, \texttt{lent}) \in FG_P \quad x : C \quad C <: C'}{FG \vdash (O', x, A) \in solveLent(O, C)}\ [\textsc{Aux-Resolve-Lent}]$$

$$\frac{\begin{array}{c} G = \langle DO, DD, DE\rangle \quad FG \vdash (O', y, A) \in solveUnique(O, C) \\ A = \texttt{unique} \implies (D = \langle D_{id}, C::\texttt{unique}\rangle \wedge \{(O, C::\texttt{unique}) \mapsto D\} \subseteq DD) \\ A = p' \implies (O' = \langle C'_{\texttt{this}}\overline{<D'>}\rangle \wedge FG, G \vdash_{O'} D \in findD(C'_{\texttt{this}}::p')) \end{array}}{FG, G \vdash_O D \in uniqueDomains(C)}\ [\textsc{Aux-UniqueDom}]$$

Figure 15: Rules for resolving `lent`, and `unique`. Auxiliary judgments *import* and *export* ensure dataflow edges are created.

$$\frac{O = \langle C\overline{<D_O>}\rangle \quad n : C_n\overline{<p>} \quad FG, G \vdash_O O_i \in lookup\ (C_n\overline{<p>}) \quad D_i = DD[(O_i, C_n::d)]}{FG, G \vdash_O D_i \in findD(C::n.d)}\ [\textsc{Df-FindD-Public}]$$

$$\frac{O = \langle C\overline{<D_O>}\rangle \quad D_i = DD[(O, C::d_i)]}{FG, G \vdash_O D_i \in findD\ (C::\texttt{this}.d_i)}\ [\textsc{Df-FindD-This}]$$

$$\frac{O = \langle C\overline{<D_O>}\rangle \quad D_i = DD[(O, C::\alpha_i)]}{FG, G \vdash_O D_i \in findD\ (C::\alpha_i)}\ [\textsc{Df-FindD}]$$

$$\frac{}{FG, G \vdash_O D_{\texttt{SHARED}} \in findD\ (::\texttt{shared})}\ [\textsc{Df-FindD-Shared}]$$

Figure 16: Auxiliary judgments for inference rules in Fig. 14.

function *summarize*(FG)
  $FG* = FG$
  $WL = \{(O_1, x_1, B_1) \overset{a}{\rightsquigarrow} (O_2, x_2, B_2) \in FG \text{ s.t. } a \text{ is } (_i\}$
  while $WL \neq \emptyset$ **do**
    remove $e_1 : (O_1, x_1, B_1) \overset{a_1}{\rightsquigarrow} (O_2, x_2, B_2)$ from WL
    if a_1 is $(_i$ **then**
      for $e_2 : (O_2, x_2, B_2) \overset{a_2}{\rightsquigarrow} (O_3, x_3, B_3) \in FG*$ **do**
        if $e_3 = concat(e_1, e_2) \notin FG*$ **then**
          add e_3 to $FG*$ and WL
    else
      if a_1 is $\bullet$ or $\star$ **then**
        for $e_2' : (O_0, x_0, B_0) \overset{a_2'}{\rightsquigarrow} (O_1, x_1, B_1) \in FG*$ **do**
          if $e_3' = concat(e_2', e_1) \notin FG*$ **then**
            add e_3' to $FG*$ and WL
  return $FG*$

$concat((O_1, x, B_1) \overset{(_i}{\rightsquigarrow} (O_2, y, B_2), (O_2, y, B_2) \rightsquigarrow (O_2, z, B_3)) =$
$= (O_1, x, B_1) \overset{(_i}{\rightsquigarrow} (O_2, z, B_3)$

$concat((O_1, x, B_1) \overset{(_i}{\rightsquigarrow} (O_2, y, B_2), (O_2, y, B_2) \overset{)_i}{\rightsquigarrow} (O_1, z, B_3)) =$
$(O_1, x, B_1) \rightsquigarrow (O_1, z, B_3)$

$concat((O_1, x, B_1) \overset{(_i}{\rightsquigarrow} (O_2, y, B_2), (O_2, y, B_2) \overset{\star}{\rightsquigarrow} (O_3, z, B_3)) =$
$(O_1, x, B_1) \overset{\star}{\rightsquigarrow} (O_3, z, B_3)$

Figure 17: The algorithm *summarize* inspired from [4, Fig. 4.16].

We conjecture but do not prove that the analysis that extracts an OOG with flow objects is also sound.

4.4 Flow Graph Analysis

Since a value might pass linearly through several assignments, the rules *solveLent* and *solveUnique* use another flow graph , FG_P, where the transitive flow is propagated, as direct flow edges may not exist in FG. FG_P is computed in two steps. First, an algorithm summarizes FG into a summary graph $FG*$, then another algorithm propagates the transitive flow for nodes in $FG*$ and computes FG_P. The algorithm *summarize* does not consider the order of the assignments and the Flow Graph Analysis is therefore flow-insensitive. Still, by using the annotations of the flow edges, the *summarize* matches the parentheses with the same value i and the Flow Graph Analysis is call-site context-sensitive.

In order to better understand the Flow Graph Analysis, we introduce three similar running examples that each creates objects of the same type. The first two examples show how the Flow Graph Analysis distinguishes between these objects, and the extraction analysis avoids creating false positive dataflow edges. The last example highlights one limitation of the extraction analysis if developers overuse `lent` and `unique`. For each example, Figure 18 also shows the extracted OGraph that has OObjects such as ⟨`A<DATA,DOM1>`⟩ and ⟨`A<DATA,DOM2>`⟩. The flow graph FG (not shown) has nodes such as (⟨`A<DATA,DOM1>`⟩, `f`, `F`) and (⟨`A<DATA,DOM2>`⟩, `f`, `F`) and the following flow edges:

$$(\langle \texttt{Main<SHARED>}\rangle, \texttt{a1}, \texttt{DATA}) \overset{(_{11}}{\rightsquigarrow} (\langle \texttt{A<DATA,DOM1>}\rangle, \texttt{this}, \texttt{owner})$$

$$(\langle \texttt{Main<SHARED>}\rangle, \texttt{n1}, \texttt{DOM1}) \overset{(_{11}}{\rightsquigarrow} (\langle \texttt{A<DATA,DOM1>}\rangle, \texttt{num}, \texttt{F})$$

$$(\langle \texttt{A<DATA,DOM1>}\rangle, \texttt{num}, \texttt{F}) \overset{\star}{\rightsquigarrow} (\langle \texttt{A<DATA,DOM1>}\rangle, \texttt{f}, \texttt{F})$$

$$(\langle \texttt{Main<SHARED>}\rangle, \texttt{a2}, \texttt{DATA}) \overset{(_{14}}{\rightsquigarrow} (\langle \texttt{A<DATA,DOM2>}\rangle, \texttt{this}, \texttt{owner})$$

$$(\langle \texttt{Main<SHARED>}\rangle, \texttt{n2}, \texttt{DOM2}) \overset{(_{14}}{\rightsquigarrow} (\langle \texttt{A<DATA,DOM2>}\rangle, \texttt{num}, \texttt{F})$$

$$(\langle \texttt{A<DATA,DOM2>}\rangle, \texttt{num}, \texttt{F}) \overset{\star}{\rightsquigarrow} (\langle \texttt{A<DATA,DOM2>}\rangle, \texttt{f}, \texttt{F})$$

$$(\langle \texttt{A<DATA,DOM2>}\rangle, \texttt{f}, \texttt{F}) \rightsquigarrow (\langle \texttt{A<DATA,DOM2>}\rangle, \texttt{ret}, \texttt{F})$$

$$(\langle \texttt{A<DATA,DOM2>}\rangle, \texttt{ret}, \texttt{F}) \overset{)_{15}}{\rightsquigarrow} (\langle \texttt{Main<SHARED>}\rangle, \texttt{dest}, \texttt{lent})$$

The algorithm *summarize* (Fig. 17) computes $FG*$ such that it concatenates two edges where the first edge has the same destination and the source of the second edge. The algorithm matches pair of edges $(_i$ and $)_i$ that have the same value for i. If the invocation $(_i$ is followed by an assignment, the algorithm propagates the invocation. The $\star$ annotation means that a method stores a value in a field. Because other methods can use the value of the field, the $\star$ annotation cancels the effect of an $(_i$ annotation and the concatenated edge keeps the $\star$ annotation.

For the example in Fig. 18, *summarize* adds the following edges.

$$(\langle \texttt{Main<SHARED>}\rangle, \texttt{n1}, \texttt{DOM1}) \overset{\star}{\rightsquigarrow} (\langle \texttt{A<DATA,DOM1>}\rangle, \texttt{f}, \texttt{F})$$

$$(\langle \texttt{Main<SHARED>}\rangle, \texttt{n2}, \texttt{DOM2}) \overset{\star}{\rightsquigarrow} (\langle \texttt{A<DATA,DOM2>}\rangle, \texttt{f}, \texttt{F})$$

The flow graph $FG*$ has a transitive flow:

$$(\langle \texttt{A<DATA,DOM2>}\rangle, \texttt{f}, \texttt{F}) \rightsquigarrow \ldots \rightsquigarrow (\langle \texttt{Main<SHARED>}\rangle, \texttt{dest}, \texttt{lent})$$

Therefore, by distinguishing between OObjects of the same type, but with different lists of ODomains $\overline{D}$, the Flow Graph Analysis avoids a false positive.

$$(\langle \texttt{A<DATA,DOM1>}\rangle, \texttt{f}, \texttt{F}) \rightsquigarrow \ldots \rightsquigarrow (\langle \texttt{Main<SHARED>}\rangle, \texttt{dest}, \texttt{lent})$$

According to *Aux-Resolve-Lent*, the Flow Graph Analysis resolves `lent` to `F` in the context of $\langle \texttt{A<DATA,DOM2>}\rangle$. Then, according to *Aux-Lookup-Lent*, *lookup* returns only $\langle \texttt{Integer<DOM2>}\rangle$ and not $\langle \texttt{Integer<DOM1>}\rangle$, which would introduce a false positive dataflow edge in the OGraph.

Related work [4, Fig. 4.16] treats fields differently from local variables, and requires a separate may-alias

analysis for finding variables that may alias the same receiver object to substitute a field `this.f` to `a1.f` or `a2.f`. In a flow node (O, `this.f`, B), the receiver `this` refers to O, so no separate may-alias analysis is required.

To compute the index i used on the value flow edges, a naive flow analysis could use the line number of the method invocation expression in the code. If our analysis were to use such a value for i in the rule *Df-Invk*, it would use the same value of i for different values of O, which would create false positive flow edges. Instead, *Df-Invk* uses $fresh_i$ to generate distinct values for i based on the pair (O, $x = r.m(\overline{y})$) and allows the analysis to distinguish between the same method invocation but in different contexts.

For example, consider Fig. 18 that shows code fragments where an object of type `B` creates an object of type `A` and invokes the method `set` to assign the value for the field f. The analysis creates two objects `a:A` in different domains for the same object allocation expression `new A()`. The assignment of the field value also occurs at the same method invocation `a.set(n)`. Due to different values returned by $fresh_i$, the analysis considers the method invocation `a.set(n)` twice, first in the context of `b1:B` and second in the context of `b2:B`. If the analysis were to consider only the line number of the method invocation as the value of i, $FG*$ would have a false positive transitive flow from (⟨`A<DATA,DOM1>`⟩,`f`,`F`) to (⟨`Main<SHARED>`⟩,`dest`,`lent`).

To compute the transitive flow, the analysis uses the algorithm *propagate* (Fig. 19), which takes as input the flow graph $FG*$ returned by the *summarize* algorithm and a source s. The output of *propagate* is a flow graph FG_P with the same nodes as FG and $FG*$, but more edges. To compute FG_P, *propagate* uses a worklist algorithm and the method $concat'$, which concatenates two edges where the destination of the first edge is the source of the first edge. For concatenation, *propagate* uses two value flow annotation *Call* and *nCall*. During the initialization, the annotation *Call* corresponds to flow edges with annotation $(_i$, and *nCall* correspond to the other annotations. If the second argument of $concat'$ is an edge annotated $(_i$, the result is an edge annotated *Call*. Otherwise, either no edge is added, or the concatenation propagates the *nCall* annotation.

Since we are using the result of the algorithm to resolve `lent` and `unique`, the propagation occurs only if any node of the two edges have a domain B that is `lent`, `unique`, or a public domain $n.d$. We also include $n.d$ because a transitive value flow may exist from (O_1, x,$n.d$) to (O_1, y,`lent`), and further to (O_2, z,`lent`), where the variables n and z are in different contexts O_1 and O_2. Here, the analysis needs to perform an extra step and find the edge from (O,ret,`this`.d) to (O_1, x,$n.d$) such that `this`.d is a domain of O.

For Fig. 18, where the variable `n2` is declared `unique`, the algorithm *propagate* adds the flow edges:

```
class A<owner,F> {
  Integer<F> f;
  void set(Integer<F> num) {
    f = num;
  }
  Integer<F> get() { return f; }
}
class Main<owner>{
  void main() {
   domain DATA,DOM1,DOM2;
   Integer<DOM1> n1= new Integer(-1);
   A<DATA,DOM1> a1 = new A();
   a1.set(n1);
   Integer<DOM2> n2 = new Integer(2);
   A<DATA,DOM2> a2 = new A();
   a2.set(n2);
   Integer<lent> dest = a2.get();
   dest.compareTo(n1);
}
```

```
class A<owner,F> {
  Integer<F> f;
  void set(Integer<F> num) {
    f = num;
  }
  Integer<F> get() { return f; }
}
class B<owner, F> {
  domain OWNED;
  A<OWNED,F> A a = new A();
  void assign(Integer<F> n) {
    a.set(n);
  }
}
class Main<owner>{
  void main() {
   domain DATA,DOM1,DOM2;
   Integer<DOM1> n1 = new Integer(-1);
   B<DATA,DOM1> b1 = new B();
   b1.assign(n1);
   Integer<unique> n2 = new Integer(2);
   B<DATA,DOM2> b2 = new B();
   b2.assign(n2);
   Integer<lent> dest = b2.a.get();
   dest.compareTo(n1);
  }
}
```

```
class A<owner,F> {
  Integer<F> f;
  void set(Integer<F> num) {
    f = num;
  }
  Integer<F> get() { return f; }
}
class B<owner, F> {
  domain OWNED;
  A<OWNED,F> A a = new A();
  void assign(Integer<F> n) {
    a.set(n);
  }
}
class Main<owner>{
  void main() {
   domain DATA,DOM1,DOM2;
   Integer<unique> n1 = new Integer(-1);
   B<DATA,DOM1> b1 = new B();
   b1.assign(n1);
   Integer<unique> n2 = new Integer(2);
   B<DATA,DOM2> b2 = new B();
   b2.assign(n2);
   Integer<lent> dest = b2.a.get();
   dest.compareTo(n1);
  }
}
```

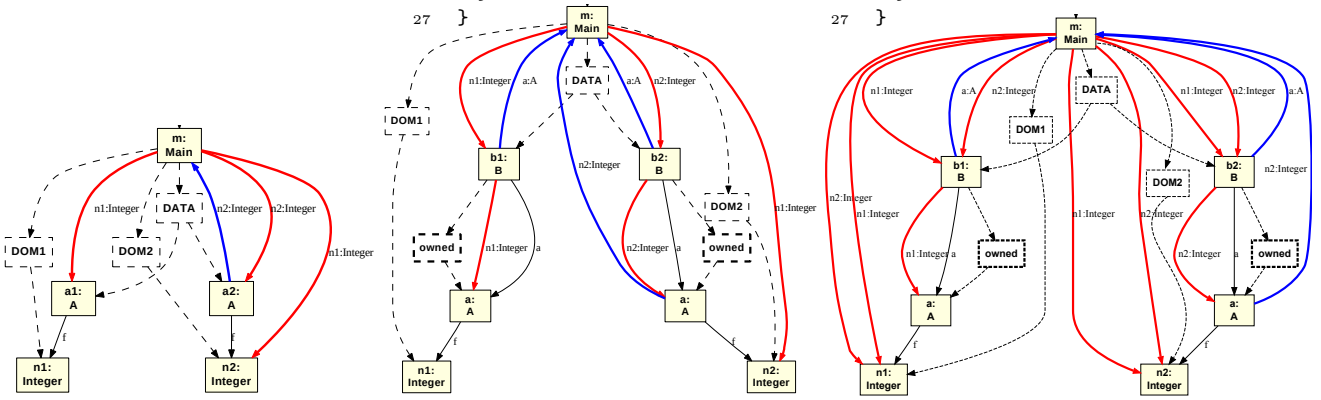


Figure 18: The analysis distinguishes between different instances of the same type `A`, and show a dataflow edge that refers to `n2:Integer` from `a2:A` to `m:Main` and not from `a1:A` to `m:Main`. It is not necessary that the objects of type `A` are created for different object allocation expressions in the code (left vs. middle). However, if developers overuse `lent` and `unique`, the analysis may add false positive edges (right).

function *propagate*($FG*$, s)
  $FG_P = \emptyset$
  $WL = \emptyset$
  for $e : (O_1, s, B_1) \overset{a}{\rightsquigarrow} (O_2, x_2, B_2) \in FG*$ **do**
    if $\{B1, B2\} \cap \{\texttt{lent}, \texttt{unique}, n.d\} \neq \emptyset$ **then**
      if a_1 is $(_i$ **then**
        add $(O_1, s, B_1) \overset{Call}{\rightsquigarrow} (O_2, x_2, B_2)$ to FG_P and WL
      else
        add $(O_1, s, B_1) \overset{nCall}{\rightsquigarrow} (O_2, x_2, B_2)$ to FG_P and WL
  while $WL \neq \emptyset$ **do**
    remove $e_1 : (O_1, x_1, B_1) \overset{nCall|Call}{\rightsquigarrow} (O_2, x_2, B_2)$ from WL
    for $e_2 : (O_2, x_2, B_2) \overset{a}{\rightsquigarrow} (O_3, x_3, B_3) \in FG*$ **do**
      if $e_3 = concat'(e_1, e_2) \notin FG_P$ **then**
        add e_3 to $FG*$ and WL
  return FG_P

function *propagateAll*(FG)
  $FG* = summarize(FG)$;
  for $(O, x, B) \in FG*$ **do**
    $FG_P = FG_P \cup propagate(FG*, x)$
  return FG_P

$concat'((O_1, x, B_1) \overset{Call}{\rightsquigarrow} (O_2, y, B_2), (O_2, y, B_2) \overset{(_i}{\rightsquigarrow} (O_2, z, B_3)) =$
$= (O_1, x, B_1) \overset{Call}{\rightsquigarrow} (O_2, z, B_3)$

$concat'((O_1, x, B_1) \overset{Call}{\rightsquigarrow} (O_2, y, B_2), (O_2, y, B_2) \overset{\star|\bullet|)_i}{\rightsquigarrow} (O_1, z, B_3))$
NO Edge

$concat'((O_1, x, B_1) \overset{nCall}{\rightsquigarrow} (O_2, y, B_2), (O_2, y, B_2) \overset{(_i}{\rightsquigarrow} (O_3, z, B_3)) =$
$= (O_1, x, B_1) \overset{Call}{\rightsquigarrow} (O_3, z, B_3)$

$concat'((O_1, x, B_1) \overset{nCall}{\rightsquigarrow} (O_2, y, B_2), (O_2, y, B_2) \overset{\star|\bullet|)_i}{\rightsquigarrow} (O_3, z, B_3)) =$
$= (O_1, x, B_1) \overset{nCall}{\rightsquigarrow} (O_3, z, B_3)$

Figure 19: . The algorithm *propagate* adds more edges for nodes with variables declared `lent` or `unique`.

$$(\langle \texttt{Main<SHARED>} \rangle, \texttt{n2}, \texttt{unique}) \overset{Call}{\rightsquigarrow} (\langle \texttt{A<DATA,DOM2>} \rangle, \texttt{num}, \texttt{F})$$
$$(\langle \texttt{Main<SHARED>} \rangle, \texttt{n2}, \texttt{unique}) \overset{Call}{\rightsquigarrow} (\langle \texttt{A<DATA,DOM2>} \rangle, \texttt{f}, \texttt{F})$$
$$(\langle \texttt{Main<SHARED>} \rangle, \texttt{n2}, \texttt{unique}) \overset{Call}{\rightsquigarrow} (\langle \texttt{A<DATA,DOM2>} \rangle, \texttt{ret}, \texttt{F})$$
$$(\langle \texttt{Main<SHARED>} \rangle, \texttt{n2}, \texttt{unique}) \overset{Call}{\rightsquigarrow} (\langle \texttt{Main<SHARED>} \rangle, \texttt{dest}, \texttt{lent})$$

Then, *Df-New-Unique* invokes *findD*(`Main::unique`) that in turn invokes *solveUnique*(⟨`Main<SHARED>`⟩,`Integer`). Here, `unique` is resolved to the domain parameter `F` that is bound to `DOM2` in the context of ⟨`B<DATA,DOM2>`⟩. Therefore, *Df-New-Unique* creates the OObject ⟨`Integer<DOM2>`⟩.

5 Conclusion

The paper presents a static analysis that uses abstract interpretation to extract an OOG with dataflow edges that refer to objects. The analysis also extracts flow objects that are passed linearly. The analysis uses the Ownership Domains type system to achieve aliasing precision and extracts a hierarchy of objects. It also uses a domain-sensitive value flow analysis to resolve the domains of variables declared as `lent` or `unique`. We evaluate the extraction analysis on an open-source Android application and the results indicate that flow objects exist in practice.

This paper is part of ongoing work on extracting a sound approximation of the runtime architecture. In a related paper [10], we use the extracted OOG to support Architectural Risk Analysis [6] and find architectural flaws such as information disclosure. We also plan to evaluate the analysis on more systems.

References

[1] M. Abi-Antoun. *Static Extraction and Conformance Analysis of Hierarchical Runtime Architectural Structure*. PhD thesis, CMU, 2010.

[2] J. Aldrich and C. Chambers. Ownership Domains: Separating Aliasing Policy from Mechanism. In *ECOOP*, 2004.

[3] J. Aldrich, V. Kostadinov, and C. Chambers. Alias Annotations for Program Understanding. In *OOPSLA*, 2002.

[4] Y. Liu. *Practical Static Analysis Framework For Inference Of Security-Related Program Properties*. PhD thesis, Rensselaer Polytechnic Institute, 2010.

[5] Y. Liu and A. Milanova. Static analysis for inference of explicit information flow. In *Proceedings of the 8th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pages 50–56, 2008.

[6] G. McGraw. *Software Security: Building Security In*. Addison-Wesley Professional, 2006.

[7] A. Spiegel. *Automatic Distribution of Object-Oriented Programs*. PhD thesis, FU Berlin, 2002.

[8] F. Swiderski and W. Snyder. *Threat Modeling*. Microsoft Press, 2004.

[9] R. Vanciu and M. Abi-Antoun. Ownership Object Graphs with Dataflow Edges. In *WCRE*, 2012.

[10] R. Vanciu and M. Abi-Antoun. Finding architectural flaws using constraints. In *ASE*, 2013.

A Auxiliary judgements

Figure 21 shows the definitions of many auxiliary judgments used earlier in the semantics. These definitions are the auxiliary judgments from ownership domains [2]. The Aux-Public rule checks whether a domain is public. The next few rules define the *domains*, and $fields$ functions by looking up the declarations in the class and adding them to the declarations in the base classes. The *owner* function just returns the first domain parameter (which represents the owning domain in our formal system).

The *mtype* function looks up the type of a method in the class; if the method is not present, it looks in the superclass instead. The *mbody* function looks up the body of a method in a similar way. Finally, the *override* function verifies that if a superclass defines method m, it has the same type as the definition of m in a subclass.

We also define $fieldDecls$ and *mtypeDecl* as the equivalent of $fields$, and *mtype* without substitution of formal domain parameters with actual domain parameters. Since actual domain parameter of the type are not consider, $fieldDecls$ and *mtypeDecl* take only the class as an argument, which they use to lookup for the class declaration in the class table CT.

$$CT(C) = \texttt{class}\ C\texttt{<}\overline{\alpha}, \overline{\beta}\texttt{>}\ \texttt{extends}\ C'\texttt{<}\overline{\alpha}\texttt{>}\ \texttt{assumes}\ \overline{\gamma} \to \overline{\delta}\ \{\ \overline{D};\ \overline{L};\ \overline{F};\ K\ \overline{M};\ \}$$

$$\frac{(\texttt{public domain}\ d) \in \overline{D}}{public(d)}\ \textit{Aux-Public}$$

$$\frac{\overline{D} = \overline{\texttt{public}_{opt}\ \texttt{domain}\ d_C} \qquad domains(C'\texttt{<}\overline{p}\texttt{>}) = \overline{d'}}{domains(C\texttt{<}\overline{p}, \overline{p'}\texttt{>}) = \overline{this.d_C}, \overline{d'}}\ \textit{Aux-Domains}$$

$$\frac{}{domains(\texttt{Object<}\overline{\alpha_0}\texttt{>}) = \emptyset}\ \textit{Aux-Domains-Obj}$$

$$\frac{\texttt{class}\ C\texttt{<}\overline{\alpha}\texttt{>}}{params(C) = \overline{\alpha}}\ \textit{Aux-Params}$$

$$\frac{\overline{F} = \overline{T}\ \overline{f} \qquad fields(C'\texttt{<}\overline{p}\texttt{>}) = \overline{T'}\ \overline{f'}}{fields(C\texttt{<}\overline{p}, \overline{p'}\texttt{>}) = ([\overline{p}/\overline{\alpha}, \overline{p'}/\overline{\beta}]\ \overline{T}\ \overline{f}), \overline{T'}\ \overline{f'}}\ \textit{Aux-Fields}$$

$$\frac{}{fields(\texttt{Object<}\overline{\alpha_0}\texttt{>}) = \emptyset}\ \textit{Aux-Fields-Obj}$$

$$\frac{}{owner(C\texttt{<}\overline{p}\texttt{>}) = p_1}\ \textit{Aux-Owner}$$

$$\frac{(T_R\ m(\overline{T}\ \overline{x})\ \{\ \texttt{return}\ e;\ \}) \in \overline{M}}{mtype(m, C\texttt{<}\overline{p}\texttt{>}) = [\overline{p}/\overline{\alpha}]\ \overline{T} \to T_R}\ \textit{Aux-MType1}$$

$$\frac{m\ is\ not\ defined\ in\ \overline{M}}{mtype(m, C\texttt{<}\overline{p}, \overline{p'}\texttt{>}) = mtype(m, C'\texttt{<}\overline{p}\texttt{>})}\ \textit{Aux-MType2}$$

$$\frac{(T_R\ m(\overline{T}\ \overline{x})\ \{\ \texttt{return}\ e;\ \}) \in \overline{M}}{mbody(m, C\texttt{<}\overline{p}\texttt{>}) = [\overline{p}/\overline{\alpha}]\ (\overline{x},\ e)}\ \textit{Aux-MBody1}$$

$$\frac{m\ is\ not\ defined\ in\ \overline{M}}{mbody(m, C\texttt{<}\overline{p}, \overline{p'}\texttt{>}) = mbody(m, C'\texttt{<}\overline{p}\texttt{>})}\ \textit{Aux-MBody2}$$

$$\frac{(mtype(m, C\texttt{<}\overline{p}\texttt{>}) = \overline{T'} \to T') \Longrightarrow (\overline{T} = \overline{T'} \wedge T = T')}{override(m, C\texttt{<}\overline{p}\texttt{>}, \overline{T} \to T)}\ \textit{Aux-Override}$$

Figure 20: Auxiliary Judgments. Source: [2].

$$CT(C) = \texttt{class}\ C\texttt{<}\overline{\alpha},\overline{\beta}\texttt{>}\ \texttt{extends}\ C'\texttt{<}\overline{\alpha}\texttt{>}\ \texttt{assumes}\ \overline{\gamma} \to \overline{\delta}\ \{\ \overline{D};\ \overline{L};\ \overline{F};\ K\ \overline{M};\ \}$$

$$\frac{\overline{F} = \overline{T}\ \overline{f} \qquad fieldDecls(C') = \overline{T'}\ \overline{f'}}{fieldDecls(C) = (\overline{T}\ \overline{f}), \overline{T'}\ \overline{f'}}\ \textit{Aux-Fields-Decl}$$

$$\frac{}{fieldDecls(\texttt{Object}) = \emptyset}\ \textit{Aux-Fields-Decl-Obj}$$

$$\frac{(T_R\ m(\overline{T}\ \overline{x})\ \{\ \texttt{return}\ e;\ \}) \in \overline{M}}{mtypeDecl(m, C) = \overline{T} \to T_R}\ \textit{Aux-MTypeDecl1}$$

$$\frac{m\ is\ not\ defined\ in\ \overline{M}}{mtypeDecl(m, C) = mtypeDecl(m, C')}\ \textit{Aux-MTypeDecl2}$$

Figure 21: Auxiliary judgments without substitution of actual with formal domain parameters.