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

1 Introduction 3

2 Dataflow Communication 4

3 Formalization 5
   3.1 Abstract Syntax ................................................................. 5
   3.2 Data Type Declarations ....................................................... 6
   3.3 Soundness ........................................................................... 16
   3.4 Theorem: Dataflow Preservation (Subject reduction) .................. 20
   3.5 Theorem: Dataflow Progress .................................................. 35
   3.6 Theorem: Object Graph Soundness ......................................... 47
      3.6.1 Lemmas ........................................................................ 48

4 Extensions lent and unique 52
   4.1 Revised Abstract Syntax ....................................................... 53
   4.2 Extended Data Type Declarations .......................................... 53
   4.3 Extended Formalization ......................................................... 55
   4.4 Flow Graph Analysis .............................................................. 59

5 Conclusion 64

A Auxiliary judgements 65
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Example of dataflow communication.</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Field write semantics.</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Simplified FDJ abstract syntax [2].</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>Data type declarations for the OGraph.</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>Static semantics.</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>Auxiliary judgments for static semantics.</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>Static semantics (continued).</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>Qualify domains rules.</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>Instrumented dynamic semantics (core rules).</td>
<td>14</td>
</tr>
<tr>
<td>10</td>
<td>Instrumented dynamic semantics (congruence rules).</td>
<td>15</td>
</tr>
<tr>
<td>11</td>
<td>Reflexive, transitive closure of the instrumented evaluation relation</td>
<td>47</td>
</tr>
<tr>
<td>12</td>
<td>FDJ syntax, extended using \texttt{len} and \texttt{unique} [2]</td>
<td>54</td>
</tr>
<tr>
<td>13</td>
<td>Data type of OGraph, and value flow graph</td>
<td>54</td>
</tr>
<tr>
<td>14</td>
<td>Static semantics of the extraction analysis. We highlight the parts that construct the value flow graph.</td>
<td>56</td>
</tr>
<tr>
<td>15</td>
<td>Rules for resolving \texttt{len} and \texttt{unique}. Auxiliary judgments \texttt{import} and \texttt{export} ensure dataflow edges are created.</td>
<td>58</td>
</tr>
<tr>
<td>16</td>
<td>Auxiliary judgments for inference rules in Fig. 14.</td>
<td>58</td>
</tr>
<tr>
<td>17</td>
<td>The algorithm \texttt{summarize} inspired from [4, Fig. 4.16].</td>
<td>59</td>
</tr>
<tr>
<td>18</td>
<td>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 \texttt{len} and \texttt{unique}, the analysis may add false positive edges (right).</td>
<td>62</td>
</tr>
<tr>
<td>19</td>
<td>. The algorithm \texttt{propagate} adds more edges for nodes with variables declared \texttt{len} or \texttt{unique}.</td>
<td>63</td>
</tr>
<tr>
<td>20</td>
<td>Auxiliary Judgments. Source: [2].</td>
<td>66</td>
</tr>
<tr>
<td>21</td>
<td>Auxiliary judgments without substitution of actual with formal domain parameters.</td>
<td>67</td>
</tr>
</tbody>
</table>
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 \cite{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 \textit{lent} for objects borrowed in method invocations, and \textit{unique} for objects passed linearly \cite{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 \textit{lent} or \textit{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 \textit{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.
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...
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; \( \ell \) and \( \theta \) ranges over locations in a store; \( \theta \) represents the value of \texttt{this}; a store \( S \) maps locations \( \ell \) to their contents; the set of variables includes the distinguished variable \texttt{this} of type \( T_{\texttt{this}} \) used to refer to the receiver of a method; the result of the computation is a location \( \ell \), which is sometimes referred to as a value \( v \); \( S[\ell] \) denotes the store entry of \( \ell \); \( S[\ell, i] \) denotes the value of \( i^{\text{th}} \) field of \( S[\ell] \); \( S[\ell \mapsto C_{<\mathcal{P}>}(\mathcal{D})] \) denotes adding an entry for location \( \ell \) to \( S \); \( \alpha \) and \( \beta \) range over formal domain parameters; \( m \) ranges over method names; \( p \) ranges over formal domain parameters, actual domains,
or the special domain \texttt{SHARED}; the expression form $\ell \triangleright e$ represents a method body $e$ executing with a receiver $\ell$; 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; $\Gamma$ is the typing context; and $\Sigma$ 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 $G = (DO, DD, DE)$, 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 $C<\overline{D}>$. 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 $\Upsilon$ records the combination of class and domain parameters $C<\overline{D}>$.

Figure 3: Simplified FDJ abstract syntax [2].
$G \in \text{OGraph} ::= (\text{Objects} = DO, \text{DomainMap} = DD, \text{Edges} = DE)$

$D \in \text{ODomain} ::= (\text{Id} = D_{id}, \text{Domain} = C::d)$

$O \in \text{OObject} ::= (\text{Type} = C<\mathcal{D}>)$

$E \in \text{OEdge} ::= (\text{From} = O_{src}, \text{To} = O_{dst}, \text{Label} = O_{label}, \text{Flag} = \text{Imp} | \text{Exp})$

$DD ::= \emptyset | DD \cup \{(O, C::d) \mapsto \rightarrow D\} \cup \{(O, C::\alpha) \mapsto \rightarrow D\}$

$DO ::= \emptyset | DO \cup \{O\}$

$DE ::= \emptyset | DE \cup \{E\}$

$\Upsilon ::= \emptyset | \Upsilon \cup \{C<\mathcal{D}>\}$

$H ::= \emptyset | H \cup \{\ell \mapsto \rightarrow O\}$

$K ::= \emptyset | K \cup \{\ell.d \mapsto \rightarrow D\}$

$L_I ::= \emptyset | L_I \cup \{(\ell_{src}, \ell_{dst}) \mapsto \{E\}\}$

$L_E ::= \emptyset | L_E \cup \{(\ell_{src}, \ell_{dst}) \mapsto \{E\}\}$

**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 $O = (C<\mathcal{D}>)$ is a child of $D_1$, or that $O$ owns a domain $D$. Given a mapping $\{(O, C''::d) \mapsto \rightarrow D\}$ 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 $\ell$ and $\ell'$ 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 ODoms $D_i$ in the OGraph and the fresh identifier $D_{id}$ ensures that multiple ODoms 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 $D_{\text{shared}}$, the root of the OGraph. A developer picks a root class, $C_{\text{root}}$, and the analysis creates $O_{\text{root}}$ in $D_{\text{shared}}$. The analysis also requires an initial context. We use a dummy OObject $O_{\text{world}}$, which does not correspond to an actual runtime object. Next, the analysis changes the context from $O_{\text{world}}$ to $O_{\text{root}}$, 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 $\ell$ to the corresponding $\text{OObject}$, and $K$ maps a runtime domain $\ell.d$ to an $\text{ODomain}$. The multi-valued maps $L_I$ and $L_E$ map a pair of locations $(\ell_{src}, \ell_{dst})$ to a set of $\text{OEdges}$ $\{E\}$. 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 $\text{OGraph}$ is sound, i.e., it has all the required $\text{OObjects}$, $\text{ODomains}$, and $\text{OEdges}$.

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

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 $\text{OObject}$ $O_{world}$ and its owning $\text{ODomain}$ $D_{\text{SHARED}}$, which constitutes the root of the $\text{OGraph}$,

$$D_{\text{SHARED}} = \langle D_0, ::\text{SHARED} \rangle \quad O_{world} = \langle C_{dummy}<$ . $>$ \rangle$$
then abstractly interprets $e_{\text{root}}$ in the context of $O_{\text{world}}$:

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

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 $CT(\text{Object})$ represent a lookup of a class $C$ and the class $\text{Object}$ 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 $O\text{Object} O_C$ 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 $\alpha_i$ 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 $O\text{Domains}$ 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 $\text{java.lang.Object}$ class (Fig. 5).

Then, Df-New uses the auxiliary judgement $\text{Aux-Dom}$ to ensure that $DD$ has an $O\text{Domain}$ corresponding to each domain that $C$ locally declares (($(O_C, C::d_j) \mapsto D_j$). $\text{Aux-Dom}$ recursively includes inherited domains from base classes as well. $\text{Aux-Obj}$, the base case of the recursion, deals with the class $\text{Object}$, for which $\text{Aux-Obj}$ does nothing, because $\text{Object}$ 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 $O\text{Object} O_C$. 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 $\Upsilon$. If this combination does not exist, Df-New extends $\Upsilon$ with the current combination. As a side note, $\Upsilon$ tracks previously analyzed $O\text{Objects}$ 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.
\[ CT(C) = \text{class } C^{<\alpha, \beta>} \text{ extends } C'^{<\gamma>} \{ \langle \tau \rangle; \text{ dom}; \ldots; \text{ ind}; \} \quad CT(\text{Object}) = \text{class } \text{Object}^{<\alpha_0>} \{ \} \]

\[ G = \langle DO, DD, DE \rangle \\
O = C_{\text{this}}^{<DO>} \quad \forall i \in 1..|\beta| \quad G \vdash_O D_i \in \text{findD}(C_{\text{this}}::p_i) \\
O_C = \langle C^{<\overline{D}>} \rangle \quad \{O_C\} \subseteq DO \\
G \vdash_O \text{dparams}(C, O_C) \quad \{(O_C, \text{qual}(p_i)) \rightarrow D_i\} \subseteq DD \\
G \vdash_O \text{ddomains}(C, O_C) \\
\forall m \in \overline{\text{md}} \text{mbody}(m, C^{<\overline{D}>}) = (\overline{\tau}, e_R) \\
C^{<\overline{D}>} \not\in \Upsilon \implies \{\tau: \overline{\tau}, \text{this}: C^{<\overline{D}>}\}, \Upsilon \cup \{C^{<\overline{D}>}\}, G \vdash_O e_R \]

\[ \Gamma, \Upsilon, G \vdash_O \text{new } C^{<\overline{D}>}(\overline{\tau}) \]

\[ e_0 : C^{<\overline{D}>} \quad (T_k, f_k) \in \text{fieldDecs}(C) \\
G \vdash_O \text{import}(C^{<\overline{D}>}, T_k) \\
\Gamma, \Upsilon, G \vdash_O e_0 \quad \text{[DF-READ]} \]

\[ e_0 : C^{<\overline{D}>} \quad (T_k, f_k) \in \text{fields}(C^{<\overline{D}>}) \\
e_1 : C_1^{<\overline{D}>} \quad C_1^{<\overline{D}>} \prec C^{<\overline{D}>} \\
G \vdash_O \text{export}(C^{<\overline{D}>}, C_1^{<\overline{D}>}) \\
\Gamma, \Upsilon, G \vdash_O e_0 \quad \Gamma, \Upsilon, G \vdash_O e_1 \quad \text{[DF-WRITE]} \]

\[ G = \langle DO, DD, DE \rangle \\
G \vdash_O O_i \in \text{lookup}(T_{\text{src}}) \\
G \vdash_O O_j \in \text{lookup}(T_{\text{label}}) \\
\{(O_i, O_j, \text{Imp})\} \subseteq DE \\
G \vdash_O \text{import}(T_{\text{src}}, T_{\text{label}}) \quad \text{[AUX-IMPORT]} \]

\[ G \vdash_O \text{import}(C^{<\overline{D}>}, T_R) \\
\forall k \in 1..|\beta| \quad e_k : T_a \quad T_a \prec T_k \\
G \vdash_O \text{export}(C^{<\overline{D}>}, T_a) \\
\Gamma, \Upsilon, G \vdash_O e_0 \\
\Gamma, \Upsilon, G \vdash_O e_1 \quad \text{[DF-INVK]} \]

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 \( \text{lookup} \) that returns the set of the \( \text{OObject}s \) \( O_k \) 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 \( \text{lookup} \) 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 \( \text{OObject}s \), and finds the flow object of an \( \text{OEdge} \). Due to subtyping, the number of actual domain parameters \( \overline{\beta} \) is smaller than or equal to the number of actual \( \text{ODomains} \) \( \overline{D} \).

The auxiliary judgements AUX-IMPORT and AUX-EXPORT ensure import and export edges between the context \( \text{OObject} \) \( O \) and the \( \text{OObject}s \) \( O_i \), where \( O_i \) is the result of \( \text{lookup} \) \( (T_{\text{src}}) \), and \( \text{lookup} \) \( (T_{\text{dst}}) \), 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 \( \text{lookup} \) in the context of \( O \), while AUX-IMPORT calls the second \( \text{lookup} \) in the context of \( O_i \). As a result, there could be multiple edges with
\[ G = \langle DO, DD, DE \rangle \quad O_C = C\langle D \rangle \quad \forall \alpha_j \in \text{params}(C) \{(O_C, C::\alpha_j) \mapsto D_j\} \subseteq DD \]
\[ G \vdash_O \text{dparams}(C', O_C) \]
\[ G \vdash_O \text{dparams}(C, O_C) \]
\[ \text{CT}(\text{Object}) = \text{class Object} <\alpha_o> \{ \} \]
\[ G \vdash_O \text{dparams}(\text{Object}, O_C) \]
\[ G = \langle DO, DD, DE \rangle \quad O = C\langle D_O \rangle \quad n : C_n\langle \rho \rangle \quad G \vdash_O O_i \in \text{lookup}(C_n\langle \rho \rangle) \]
\[ D_i = DD[(O_i, C_n::d)] \]
\[ G \vdash_O D_i \in \text{findD}(C::n.d) \]
\[ G = \langle DO, DD, DE \rangle \quad O = C\langle D_O \rangle \quad D_i = DD[(O, C::d_i)] \]
\[ G \vdash_O D_i \in \text{findD}(C::this.d_i) \]
\[ G = \langle DO, DD, DE \rangle \quad O = C\langle D_O \rangle \quad D_i = DD[(O, C::\alpha_i)] \]
\[ G \vdash_O D_i \in \text{findD}(C::\alpha_i) \]
\[ G \vdash_O D_{SHARED} \in \text{findD}(::\text{shared}) \]
\[ G = \langle DO, DD, DE \rangle \quad \forall (\text{domain } d_j) \in \text{dom} \quad D_j = (D_{d_j}, C::d_j) \quad \{(O_C, C::d_j) \mapsto D_j\} \subseteq DD \]
\[ G \vdash_O \text{ddomains}(C', O_C) \]
\[ G \vdash_O \text{ddomains}(C, O_C) \]
\[ G \vdash_O \text{ddomains}(\text{Object}, O_C) \]
\[ G = \langle DO, DD, DE \rangle \quad O = C_{\text{this}}\langle D \rangle \quad O_k \in DO \quad O_k = \langle C\langle D \rangle \rangle \quad C' < C' \]
\[ \forall i \in 1..|\rho'| \quad G \vdash_O D'_i \in \text{findD}(C_{\text{this}}::\rho'_i) \quad D'_i = D_i \]
\[ G \vdash_O O_k \in \text{lookup}(C'\langle \rho' \rangle) \]

**Figure 6:** Auxiliary judgments for static semantics.

different labels between the same two \texttt{OObject}s, depending on what \texttt{lookup} returns.

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

\texttt{Df-Read} and \texttt{Df-Write} abstractly interpret field read and field write expressions, and use \texttt{Aux-Import}.
\[
\begin{align*}
\frac{\Gamma, \Upsilon, G \vdash O \, x}{\Gamma, \Upsilon, G \vdash O \, x} & \quad [\text{DF-Var}] \\
\frac{\Gamma, \Upsilon, G \vdash O \, \ell}{\Gamma, \Upsilon, G \vdash O \, \ell} & \quad [\text{DF-Loc}] \\
\frac{O_C = H[\ell]}{\Gamma, \Upsilon, G \vdash O, \ell \triangleright e} & \quad [\text{DF-Context}] \\
G = \langle DO, DD, DE \rangle & \quad \forall \ell \in \text{dom}(S), \Sigma[\ell] = C<\overrightarrow{p}> \\
H[\ell] = O = \langle C<\overrightarrow{p}> \rangle \in DO & \quad \{x : T, \epsilon_R\} = \langle x : T, \text{this} : C<\overrightarrow{p}>\}, \emptyset, G \vdash O, e_R \\
G \vdash C_T, H \Sigma & \quad [\text{DF-Sigma}]
\end{align*}
\]

Figure 7: Static semantics (continued).

\[
\begin{align*}
\frac{\Gamma; \Sigma; \theta \vdash n : C<\overrightarrow{p}>}{\Gamma; \Sigma; \theta \vdash \text{qual}(n.d) = C:d} & \quad [\text{QUAL-VAR}] \\
\frac{\Gamma; \Sigma; \theta \vdash \text{this} : C\text{this}<\overrightarrow{p}>}{\Gamma; \Sigma; \theta \vdash \text{qual}(\alpha) = C\text{this}::\alpha} & \quad [\text{QUAL-PARAM}] \\
\frac{\Gamma; \Sigma; \theta \vdash \text{qual}(\text{shared}) = ::\text{shared}}{\text{[QUAL-SHARED]}}
\end{align*}
\]

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. 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 \(\ell\) 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 \(\ell\) in the store, and that every \(\ell\) 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 \(\Gamma\), \(\Sigma\), and \(\theta\), Qual-Var qualifies \(n.d\) as \(C:d\). This judgement also applies to the case when \(n\) is \text{this} and \(p = \text{this}.d\). Qual-Param qualifies a formal domain parameter \(\alpha\) 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 \(\theta\), 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 \(\ell\) to the store \(S\), where \(\ell\) maps to an object of type \(C\) with the specified ownership domain parameters, and the fields set to the values \(\mathcal{v}\) passed to the constructor. The rule extends \(H\) by mapping \(\ell\) 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 \(O_\ell\). They use \(\theta\) and \(\ell\) to lookup these \(OObject\)s in \(H\). They also ensure that the edge label \(O_v\) is of a subclass of the return class \(C_R\). IR-Invk also ensures that an export \(OEdge\) \(E_k\) exist from \(O\) to \(O_\ell\) for every parameter, having as edge label an \(OObject\) of a subclass of the method’s parameter class \(C_k\). The rule uses \(\theta\) and \(\ell\) to lookup \(O\) and \(O_\ell\) in \(H\). It extends both \(L_I\) and \(L_E\) by adding \(E\) to the set of edges associated with \((\ell, \theta)\) in \(L_I\), and \((\theta, \ell)\) in \(L_E\).

The IR-Lookup auxiliary judgment returns the set of \(OObject\) found by looking up each actual domain \(\ell', d_i\) 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-
\[
\ell \notin \text{dom}(S) \quad S' = S[\ell \mapsto C<\overline{\rho}>(\overline{\nu})] \\
G = \langle DO, DD, DE \rangle \\
\overline{\rho} = \overline{\tau.d} \quad \forall i \in 1..|\overline{\tau.d}| \quad D_i = K[\ell_i.d_i] \\
\ell_i \in \text{dom}(H) \quad \text{s.t.} \quad H[\ell_i] = O_i \quad D_i = DD[O_i, \text{qual}(\ell_i.d_i)] \\
O_C = \langle C<\overline{D}>, \overline{\nu} \rangle \quad O_C \in DO \quad H' = H[\ell \mapsto OC] \\
\forall (\text{domain } d_i) \in \text{domains}(C<\overline{\rho}>) \quad D_j = DD[O_C, C::d_j] \quad K' = K[\ell.d_j \mapsto D_j]
\]

[IR-NEW]

\[
\theta \vdash \text{new } C<\overline{\rho}>(\overline{\nu}); S \quad H; K; L_I; L_E \rightsquigarrow_G \ell; S' \quad H'; K'; L_I; L_E
\]

\[
S[\ell] = C<\overline{\rho}>(\overline{\nu}) \quad \text{fields}(C<\overline{\rho}>) = T \quad T \quad T
\]

\[E = \langle O_i, O_v, \text{Imp} \rangle \in DE \quad H; K; L_I; L_E \vdash O_v \in \text{irLookup}(T_i) \quad L'_i = L_i[(\ell, \theta) \mapsto \cup \{E\}]
\]

[IR-READ]

\[
\theta \vdash \ell.f_i; S; H; K; L_I; L_E \rightsquigarrow_G \nu; S' \quad H; K; L_I; L_E
\]

\[
S[\ell] = C<\overline{\rho}>(\overline{\nu}) \quad \text{fields}(C<\overline{\rho}>) = T \quad T \\
S' = S[\ell \mapsto \tau<\overline{\nu}>([\nu/v_i])]
\]

\[E = \langle O_i, O_v, \text{Exp} \rangle \in DE \quad H; K; L_I; L_E \vdash O_v \in \text{irLookup}(T_i) \quad T_i \in T
\]

[IR-WRITE]

\[
\theta \vdash \ell.f_i = v; S; H; K; L_I; L_E \rightsquigarrow_G v; S' \quad H; K; L_I; L_E
\]

\[
S[\ell] = C<\overline{\rho}>(\overline{\nu}) \quad \text{mbody}(m, C<\overline{\rho}>) = (\overline{\pi}, e_R) \\
O = H[\theta] \quad O_i = H[\ell] \quad O_v = H[v_i] \quad H; K; L_I; L_E \vdash \text{irLookup}(T_i) \quad T_i \in T
\]

\[E = \langle O_i, O_v, \text{Exp} \rangle \in DE \quad L'_i = L_i[(\ell, \theta) \mapsto \cup \{E\}]
\]

[IR-INVK]

\[
\theta \vdash \ell.m(\overline{\pi}); S; H; K; L_I; L_E \rightsquigarrow_G \ell \uparrow \overline{\pi}[\overline{\nu}, \ell/\text{this}]e_R; S \quad H; K; L_I; L_E
\]

\[
\theta \vdash \ell.v; S; H; K; L_I; L_E \rightsquigarrow_G v; S \quad H; K; L_I; L_E
\]

\[
O_k \in \text{rng}(H) \quad O_k = \langle C'<\overline{\rho}'> \rangle \quad C' <: C \\
\forall i \in 1..|\overline{\rho'}| \quad D_i = K[\ell_i.d_i] \quad D'_i = D_i \\
H; K; L_I; L_E \vdash O_k \in \text{irLookup}(C<\overline{\rho}'>)
\]

[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 \(e'_1\).
\[
\begin{align*}
\theta \vdash e_i; S; H; K; L_I; L_E \rightsquigarrow_G e'_i; S'; H'; K'; L'_I; L'_E & \quad \text{[IRC-NEW]} \\
\theta \vdash \text{new } C<p>v_1; \ldots; v_{i-1}, e_i; e_{i+1}; \ldots; e_n; S; H; K; L_I; L_E \rightsquigarrow_G \text{new } C<p>v_1; \ldots; v_{i-1}, e'_i; e_{i+1}; \ldots; e_n; S'; H'; K'; L'_I; L'_E & \\
\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e'_0; S'; H'; K'; L'_I; L'_E & \quad \text{[IRC-READ]} \\
\theta \vdash e_0.f_i; S; H; K; L_I; L_E \rightsquigarrow_G e'_0.f_i; S'; H'; K'; L'_I; L'_E & \\
\theta \vdash e_i; S; H; K; L_I; L_E \rightsquigarrow_G e'_i; S'; H'; K'; L'_I; L'_E & \quad \text{[IRC-WRITE-RCV]} \\
\theta \vdash e_i; S; H; K; L_I; L_E \rightsquigarrow_G e'_i; S'; H'; K'; L'_I; L'_E & \\
\theta \vdash v.m(\tau); S; H; K; L_I; L_E \rightsquigarrow_G e'_0.m(\tau); S'; H'; K'; L'_I; L'_E & \quad \text{[IRC-WRITE-ARG]} \\
\theta \vdash e_0; S; H; K; L_I; L_E \rightsquigarrow_G e'_0; S'; H'; K'; L'_I; L'_E & \\
\theta \vdash e_0.m(\tau); S; H; K; L_I; L_E \rightsquigarrow_G e'_0.m(\tau); S'; H'; K'; L'_I; L'_E & \quad \text{[IRC-RECVINVK]} \\
\theta \vdash e_i; S; H; K; L_I; L_E \rightsquigarrow_G e'_i; S'; H'; K'; L'_I; L'_E & \\
\theta \vdash v.m(v_1; \ldots; v_{i-1}, e_i; e_{i+1}; \ldots; e_n); S; H; K; L_I; L_E \rightsquigarrow_G v.m(v_1; \ldots; v_{i-1}, e'_i; e_{i+1}; \ldots; e_n); S'; H'; K'; L'_I; L'_E & \quad \text{[IRC-ARGINVK]} \\
\theta \vdash \ell \triangleright e; S; H; K; L_I; L_E \rightsquigarrow_G \ell \triangleright e'; S'; H'; K'; L'_I; L'_E & \quad \text{[IRC-CONTEXT]} \\
\end{align*}
\]

\textbf{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 $\ell$ 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 $\ell_1$ to $\ell_2$ 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 $(\ell_1, \ell_2)$ 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).

∀ Σ ⊩ S, (S, H, K, LI, LE) ∼ (DO, DD, DE)

⇐⇒

∀ ℓ ∈ dom(S), Σ[ℓ] = C<\overrightarrow{\ell.d}>

⇒

H[ℓ] = OC = ⟨C<\overrightarrow{D}⟩⟩ ∈ DO

and ∀ ℓ′, dj ∈ \overrightarrow{\ell.d} K[ℓ′, dj] = Dj = ⟨D_{idj}, qual(ℓ′, dj)⟩ ∈ rng(DD)

and ∀ dj ∈ domains(C<\overrightarrow{\ell.d}>)

K[ℓ.d_i] = Di = ⟨D_{idj}, C::d_i⟩ {⟨OC, C::d_i⟩ ↦ Dj} ∈ DD

and ∀ ℓ_{src} ∈ dom(H), fields(Σ[ℓ_{src}]) = \overrightarrow{T_{src} T}

∀ m. mtype(m, Σ[ℓ_{src}]) = T → TR

∀ T_k ∈ (\overrightarrow{T_{src}}) \cup \{TR\}

H; K; LI; LE ⊩ O_k ∈ irLookup(T_k)

E'_k ∈ LI[(ℓ_{src}, ℓ)]

E'_k = (H[ℓ_{src}], H[ℓ], O_k, Imp) ∈ DE

and ∀ ℓ_{dst} ∈ dom(H), fields(Σ[ℓ_{dst}]) = \overrightarrow{T_{dst} T}

∀ m. mtype(m, Σ[ℓ_{dst}]) = T → TR

∀ T_k ∈ (\overrightarrow{T_{dst}}) \cup \{T\}

H; K; LI; LE ⊩ O_k ∈ irLookup(T_k)

E_k ∈ LE[(ℓ, ℓ_{dst})]

E_k = (H[ℓ], H[ℓ_{dst}], O_k, Exp) ∈ DE

Df-Approx states that given a well-typed store S of a program and an OGraph G = (DO, DD, DE) of the same program, there are maps H, K, LI, and LE, such that H maps each runtime object ℓ in the store to a unique OObject OC from DO, K maps each runtime domain ℓ.d_i in the store to a unique ODomain D_i, and LI and LE 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. LI maps a runtime
dataflow communication from a runtime object \( \ell_{\text{src}} \) to another runtime object \( \ell \) back to an import \( \text{OEdge} \) \( E'_k \) 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 \( \ell_{\text{src}} \) as its receiver. The condition also ensures that the edge’s label is an \( \text{OObject} \) of a subtype of \( T_k \). \( T_k \) is the type of a field of \( \ell_{\text{src}} \), or the return type of a method of \( \ell_{\text{src}} \).

Similarly, \( L_E \) maps a runtime dataflow communication from a runtime object \( \ell \) to another runtime object \( \ell_{\text{dst}} \) back to an export \( \text{OEdge} \) \( E_k \) 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 \( \ell_{\text{dst}} \) as its receiver. The condition also ensures that the edge’s label is an \( \text{OObject} \) of a subtype of \( T_k \). \( T_k \) is the type of a field of \( \ell_{\text{dst}} \), or the type of a parameter of \( \ell_{\text{dst}} \)’s methods.

**Theorem: Dataflow Object Graph Soundness.**

If \( G = (DO, DD, DE) \)

\[
G \vdash (CT, e_{\text{root}})
\]

\[
\forall e, \theta_0 \vdash e; \emptyset, \emptyset, \emptyset, \emptyset \leadsto_G^* e; S; H; K; L_I; L_E
\]

\[
\Sigma \vdash S
\]

then \( G \vdash_{CT, H} \Sigma \)

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

where \( \leadsto_G^* \) relation is the reflexive and transitive closure of \( \leadsto_G \) relation, and \( \theta_0 \) is the location of the first object instantiated by \( e_{\text{root}} \). 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).

If \( \emptyset, \Sigma, \theta \vdash e : T \)
\[ \Sigma \vdash S \]
\[ G = \langle DO, DD, DE \rangle \]
\[ G \vdash_{CT,H} \Sigma \]
\[ \emptyset, \emptyset, G \vdash_O e \]
\[ (S, H, K, L_I, L_E) \sim (DO, DD, DE) \]
\[ \theta \vdash \boxed{c ; S} ; H ; K ; L_I ; L_E \sim_G \boxed{e ; S} ; H' ; K' ; L_I' ; L_E' \]
then there exists \( \Sigma' \supseteq \Sigma \) and \( T' < T \) such that
\[ \emptyset, \Sigma', \theta \vdash e' : T' \text{ and } \Sigma' \vdash S' \]
\[ (S', H', K', L_I', L_E) \sim (DO, DD, DE) \]
\[ \emptyset, \emptyset, G \vdash_O e' \]
and \( G \vdash_{CT,H} \Sigma' \)

Theorem: Dataflow Progress.

If \( \emptyset, \Sigma, \theta \vdash e : T \)
\[ \Sigma \vdash S \]
\[ G = \langle DO, DD, DE \rangle \]
\[ G \vdash_{CT,H} \Sigma \]
\[ \emptyset, \emptyset, G \vdash_O e \]
\[ (S, H, K, L_I, L_E) \sim (DO, DD, DE) \]
then either \( e \) is a value
or else \( \theta \vdash \boxed{c ; S} ; H ; K ; L_I ; L_E \sim_G \boxed{e' ; S'} ; H' ; K' ; L_I' ; L_E' \)
3.4 Theorem: Dataflow Preservation (Subject reduction)

If

\[\emptyset, \Sigma, \theta \vdash e : T\]
\[\Sigma \vdash S\]

\[G = \langle DO, DD, DE \rangle\]
\[\emptyset, \emptyset, G \vdash_O e\]

\[(S, H, K, L_I, L_E) \sim \langle DO, DD, DE \rangle\]
\[\theta \vdash \langle c; S \rangle H; K; L_I; L_E \sim_G \langle c'; S' \rangle H'; K'; L'_I; L'_E\]

then

\[\exists \Sigma' \supseteq \Sigma \text{ and } T' \triangleleft T \text{ such that } \emptyset, \Sigma', \theta \vdash e : T' \text{ and } \Sigma' \vdash S'\]

\[(S', H', K', L'_I, L'_E) \sim \langle DO, DD, DE \rangle\]
\[\emptyset, \emptyset, G \vdash_O e'\]
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: \(e; S \sim e'; S'\).

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

\[\emptyset \vdash e; S; H; K; L; L_E \sim_G e'; S'; H'; K'; L'_I; L'_E\]

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

**Case Ir-New:** \(e = \text{new } C<\ell.d>(\tau)\), and \(e' = \ell\). We have:

\[G = \langle DO, DD, DE \rangle\]
\[O = C_{\text{this}}<\tau_d>\]
\[O \vdash_{CT,H} D_i \in \text{findD}(C_{\text{this}} : \ell.d_i)\]
\[O_C = \langle C<\ell.d> \rangle\]
\[O_{C_G} \subseteq DO\]
\[G \vdash_{O} \text{dparams}(C, O_C)\]
\[\{(O_C, \text{qual}(\ell.d_i)) \mapsto D_i\} \subseteq DD\]
\[G \vdash_{O} \text{domains}(C, O_C)\]
\[\forall m \in \text{mbody}(m, C<\ell.d>) = (x : \mathcal{T}, \epsilon_R)\]
\[C<\mathcal{T}> \not\in \Upsilon \implies \Upsilon \vdash (\mathcal{T} : \mathcal{T}, \text{this} : C<\ell.d>, \Upsilon \cup C<\ell.d>, G \vdash_{O} \epsilon_R)\]
\[\Gamma, \Upsilon, G \vdash_{O} \tau\]

\[\Upsilon \vdash \text{new } C<\ell.d>(\tau)\]

\[G = \langle DO, DD, DE \rangle\]
\[O = \text{C\text{\_this}}<\tau_d>\]
\[\forall i \in \llbracket \ell.d \rrbracket, D_i = K[\ell.d_i]\]
\[\ell_i \in \text{dom}(H) \text{ s.t. } H[\ell_i] = K, D_i = \text{DD}(H, \text{qual}(\ell.d_i))\]
\[O_C = \langle C<\ell.d> \rangle\]
\[O_{C_G} \subseteq \text{DO}\]
\[H' = H[\ell \mapsto O_C]\]
\[\forall \text{domain } d_j \in \text{domains}(C<\tau.d>)\]
\[D_j = \text{DD}(\langle O_C, C_{\text{this}} : d_j \rangle)\]
\[K' = K[\ell.d_j \mapsto D_j]\]

\[\theta \vdash \text{new } C<\ell.d>(\tau); S; H; K; L; L_E \sim_G \ell; S' \vdash H'; K'; L'_I; L'_E\]

To Show:

1. \((S', H', K', L'_I, L'_E) \sim \langle DO, DD, DE \rangle\)
2. \(\emptyset, \emptyset, G \vdash_{O} e'\)
3. \(G \vdash_{CT,H} \Sigma'\)
\[\theta \vdash v; S; H; K; L_1; L_E \rightsquigarrow_G e'; S'; H'; K'; L'_1; L'_E (S, H, K, L_1, L_E) \sim (DO, DD, DE)\]
\[\forall \ell \in \text{dom}(S), \Sigma[\ell] = C_{\varnothing,d}\]
\[H[\theta] = O_C = (C_{\varnothing,d}) \in DO\]
and \[\forall \ell',d_j \in \varnothing.d K[\theta,d_j] = D_j = (D_{id}, \text{qual}(\theta',d_j)) \in \text{rng}(DD)\]
and \[\forall d_i \in \text{domains}(C_{\varnothing,d})\]
\[K[\theta,d_i] = D_i = (D_{id}, C::d_i) \quad \{(O_C, C::d_i) \mapsto D_i\} \in DD\]
\[\forall \ell_{src} \in \text{dom}(H), \text{fields}(\Sigma[\ell_{src}]) = T_{src} f\]
\[\forall m. \text{mtype}(m, \Sigma[\ell_{src}]) = T \rightarrow T_R\]
\[\forall T_k \in \{T_{src}\} \cup \{T_R\}\]
\[H; K; L_1; L_E \vdash O_k \in \text{irLookup}(T_k)\]
\[E_k^\theta = L_1[\ell_{src}, \theta] \quad E_k^\theta = (H[\ell_{src}], H[\theta], O_k, \text{Imp}) \in DE\]
\[\forall \ell_{dst} \in \text{dom}(H), \text{fields}(\Sigma[\ell_{dst}]) = T_{dst} f\]
\[\forall m. \text{mtype}(m, \Sigma[\ell_{dst}]) = T \rightarrow T_R\]
\[\forall T_k \in \{T_{dst}\} \cup \{T\}\]
\[H; K; L_1; L_E \vdash O_k \in \text{irLookup}(T_k)\]
\[E_k^\ell_{dst} = L_1[\ell_{dst}, \theta] \quad E_k^\ell_{dst} = (H[\theta], H[\ell_{dst}], O_k, \text{Exp}) \in DE\]
\[O_C = (C_{\ell_{src}}) \in DO\]
\[S' = S[ \ell \rightarrow C_{\ell_{src}} \varnothing.d(\varnothing)]\]
\[H' = H[ \ell \rightarrow O_C]\]
\[\forall i \in 1..\varnothing.d\ D_i = K[\ell_i,d_i]\]
\[\forall (\text{domain } d_i) \in \text{domains}(C_{\varnothing,d})\quad D_j = DD[(O_C, C::d_j)]\]
\[K' = K[\ell_1,d_j \mapsto D_j]\]
\[L'_1 = L_1 \quad L'_E = L_E\]
\[\exists \Sigma' \supseteq \Sigma \quad \text{and } T' < : T \quad \text{s.t. } \emptyset, \Sigma', \theta \vdash e' : T' \quad \text{and } \Sigma' \vdash S'\]
\[\Sigma'[\ell] = C_{\ell_{src}}\]
\[S', H', K', L'_1, L'_E \sim (DO, DD, DE)\]
This proves (1).
\[\emptyset, \emptyset, G \vdash_O e'\]
This proves (2).
\[G \vdash_{CT,H} \Sigma\]
\[\forall \ell \in \text{dom}(S), \Sigma[\ell] = C_{\ell_{src}}\]
\[H[\ell] = O_{\ell} = (C_{\ell_{src}}) \in DO\]
\[\forall m. \text{mbody}(m, C_{\ell_{src}}) = (\pi : T, e_R)\]
\[\{\pi : T, \text{this } : C_{\ell_{src}}\}, \emptyset, G \vdash_{O_{\ell}} e_R\]
\[O_C = (C_{\varnothing,d}) \in DO\]
\[S' = S[ \ell \rightarrow C_{\varnothing,d}(\varnothing)]\]
\[H' = H[ \ell \rightarrow O_C]\]
\[\emptyset, \emptyset, G \vdash_e e\]
\[e = \text{new } C_{\varnothing,d}(\varnothing), \quad \text{and } \emptyset = \emptyset\]
By assumption
By sub-derivation of Ir-New
By sub-derivation of Ir-New
By sub-derivation of Ir-New
By sub-derivation of Ir-New
By sub-derivation of Ir-New
By sub-derivation of Ir-New
By sub-derivation of Ir-New
By sub-derivation of Ir-New
By assumption with e, \emptyset below
By assumption
∀m. mbody(m, C<\texttt{p}>) = (\texttt{t}: T, e_R) 
By sub-derivation of Df-New

C<\texttt{D}> /∈ \Upsilon \implies 
\{\texttt{t}: T, this: C<\texttt{p}>\}, \Upsilon \cup \{C<\texttt{D}>\}, G \vdash_{\text{O}_C} e_R 
By sub-derivation of Df-New

\{\texttt{t}: T, this: C<\texttt{p}>\}, \emptyset, G \vdash_{\text{O}_C} e_R 
By Df-Strengthening Lemma

∀\ell \in \text{dom}(\textit{S}'), \Sigma'[\ell] = C_\ell<\texttt{p}> 

H'[\ell] = O_\ell = \langle C_\ell<\texttt{D}>\rangle \in DO 

∀m. mbody(m, C_\ell<\texttt{p}>) = (\texttt{t}: T, e_R) 
By above

\{\texttt{t}: T, this: C_\ell<\texttt{p}>\}, \emptyset, G \vdash_{\text{O}_\ell} e_R 
By Df-Sigma with above H' and \Sigma'

This proves (3).
Case Ir-Read: \( e = \ell.f_i \) and \( e' = v_i \). We have:

\[
\begin{align*}
    e_0 : C\langle \overline{p} \rangle & \quad (T_k f_k) \in \text{fieldDecls}(C) \\
    G \vdash_O \text{import}(C\langle \overline{p} \rangle, T_k) \\
    \Gamma, \Upsilon, G \vdash_O e_0 & \quad [\text{DF-READ}] \\
\end{align*}
\]

\[
\begin{align*}
    \{S[\ell] = C\langle \overline{p} \rangle \mid \text{fields}(C\langle \overline{p} \rangle) = \mathcal{T} \mathcal{J} \} \\
    O = H[\theta] & \quad O_{\ell} = H[\ell] \quad O_v = H[v_i] \\
    T_i \in \mathcal{T} & \quad E = (O_{\ell}, O_v, \text{Imp}) \in \text{DE} \\
    H; K; L_{1i}; L_E \vdash_O e_0 \in \text{irLookup}(T_i) & \quad L'_i = L_i[(\ell, \theta) \mapsto \cup \{E\}] \\
    \theta \vdash S[\ell]; S \vdash H; K; L_{1i}; L_E \Rightarrow_G v_i; S \vdash H; K; L'_i; L_E & \quad [\text{IR-READ}] \\
\end{align*}
\]

\[
\begin{align*}
    G = \langle D_O, D_D, D_E \rangle & \quad \forall \ell \in \text{dom}(S), \Sigma[\ell] = C\langle \overline{p} \rangle \\
    H[\ell] = O = \langle C\langle \overline{D} \rangle \rangle \in \text{DO} & \quad \forall m. \text{mbody}(m, C\langle \overline{p} \rangle) = (\overline{\mathcal{T}}, \overline{e_R}) \\
    \{\overline{\mathcal{T}}, \text{this} : C\langle \overline{p} \rangle \}, \theta, G \vdash_O e_R & \quad [\text{DF-SIGMA}] \\
\end{align*}
\]

To show:

1. \( (S', H', K', L'_1, L'_E) \sim (D_O, D_D, D_E) \)
2. \( \emptyset, \emptyset, G \vdash \ e' \)
3. \( G \vdash_{CT,H} \Sigma' \)  

\[
\theta \vdash e; S; H; K; L_{1i}; L_E \Rightarrow_G e'; S'; H'; K'; L'_1; L'_E \\
(S, H, K, L, L_E) \sim (D_O, D_D, D_E) \\
\forall \ell \in \text{dom}(S), \Sigma[\ell] = C\langle \overline{p} \rangle \\
H[\theta] = O = \langle C\langle \overline{D} \rangle \rangle \in \text{DO} \\

\]

and \( \forall \theta', d_j \in \overline{\mathcal{D}}, \mathcal{K}[\theta', d_j] = D_j = \langle D_{id_j}, \text{qual}(\theta', d_j) \rangle \in \text{rng}(DD) \)

and \( \forall d_i \in \text{domains}(C\langle \overline{D} \rangle) \)

\[
K[\theta, d_i] = D_i = \langle D_{id_i}, C\cdot d_i \rangle \quad \{(O_C, C\cdot d_i) \mapsto D_i\} \in DD \\
\forall \ell_{src} \in \text{dom}(H), \text{fields}(\Sigma[\ell_{src}] = \mathcal{T}_{src} \mathcal{J} \\
\forall m. \text{mtype}(m, \Sigma[\ell_{src}] = \mathcal{T} \mapsto \mathcal{T}_R \\
\forall T_k \in \{\mathcal{T}_{src}\} \cup \{\mathcal{T}_R\} \\
\quad H; K; L_{1i}; L_E \vdash_O e_0 \in \text{irLookup}(T_k) \\
\quad E'_k \in L_{1i}[(\ell_{src}, \theta)] E'_k = \langle H[\ell_{src}], H[\theta], O_k, \text{Imp} \rangle \in \text{DE} \\
\forall \ell_{dst} \in \text{dom}(H), \text{fields}(\Sigma[\ell_{dst}] = \mathcal{T}_{dst} \mathcal{J} \\
\forall m. \text{mtype}(m, \Sigma[\ell_{dst}] = \mathcal{T} \mapsto \mathcal{T}_R \\
\forall T_k \in \{\mathcal{T}_{dst}\} \cup \{\mathcal{T}\} \\
\quad H; K; L_{1i}; L_E \vdash_O e_0 \in \text{irLookup}(T_k) \\
\quad E_k \in L_{1i}[(\ell, \ell_{dst})] E_k = \langle H[\theta], H[\ell_{dst}], O_k, \text{Exp} \rangle \in \text{DE} \\
S' = S, H' = H, K' = K, L'_E = L_E \\
S[\ell] = C\ell\langle \overline{p} \rangle \quad \text{fields}(C\ell\langle \overline{p} \rangle) = \mathcal{T} \mathcal{J} \\
O = H[\theta] \quad O_{\ell} = H[\ell] \quad O_v = H[v_i] \\
T_i \in \mathcal{T} \\
E' = (O_{\ell}, O_v, \text{Imp}) \in \text{DE} \\
H; K; L_{1i}; L_E \vdash_O e_0 \in \text{irLookup}(T_i) \\
L'_i = L_i[(\ell, \theta) \mapsto \cup \{E'\}] \\
\]

By assumption

By assumption

Since \( \Sigma \vdash S \)

By Df-Approx

By Df-Approx

By Df-Approx

By sub-derivation of Ir-Read

By sub-derivation of Ir-Read

By sub-derivation of Ir-Read

By sub-derivation of Ir-Read

By sub-derivation of Ir-Read

By sub-derivation of Ir-Read

By sub-derivation of Ir-Read

By sub-derivation of Ir-Read
∀ℓ_{src} ∈ dom(H'), fields(Σ'[ℓ_{src}]) = \overrightarrow{T_{src}I},
∀m. mtype(m, Σ'[ℓ_{src}]) = \overrightarrow{T} \rightarrow \overrightarrow{T_R}
∀T_k \in \{T_{src}\} \cup \{T_R\}
∀H'; K'; L'_I; L'_E ⊢ O_k ∈ irLookup(T_k)
E'_k ∈ L'_I[(ℓ_{src}, θ)] = \langle H'[ℓ_{src}], H'[θ], O_k, Imp \rangle ∈ DE

By above, since Σ' = Σ

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

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 \(\Sigma' = Σ\)
This proves (3).
Case IR-Write: $e = \ell.f_i = v$, and $e' = v$

We have:

\[ e_0 : C<\overline{p}> (T_k \ f_k) \in \text{fields}(C<\overline{p}>) \]
\[ e_1 : C_1<\overline{p}'> C_1<\overline{p}'> \prec T_k \]
\[ G \vdash_O \text{export}(C<\overline{p}>, C_1<\overline{p}'>) \]
\[ \Gamma, \ Y, G \vdash_O e_0 \quad \Gamma, \ Y, G \vdash_O e_1 \quad [\text{DF-WRITE}] \]

\[ S[\ell] = C<\overline{p}>(\overline{p}) \quad \text{fields}(C<\overline{p}>) = \overline{t} \overline{f} \]

\[ O = H[v] \quad O_v = H[v] \quad O_v \vdash \text{irLookup}(T_i) \quad T_i \in \mathcal{T} \]
\[ E = (O, O_t, O_v, Exp) \in \text{DE} \quad L_E = L_E[[\theta, \ell] \mapsto \lambda (E)] \quad [\text{IR-WRITE}] \]

\[ G = \langle DO, DD, DE \rangle \quad \forall \ell \in \text{dom}(S), \Sigma[\ell] = C<\overline{p}> \quad H[\ell] = O \wedge (C<\overline{D}>) \in DO \]
\[ \forall m. \ \text{mbody}(m, C<\overline{p}>) = (\overline{t}, \overline{f}, e_R) \quad \{\overline{t}, \overline{f}, \text{this} : C<\overline{p}>\}, \theta, G \vdash_O e_R \quad [\text{DF-SIGMA}] \]

To Show:

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

\[ \emptyset \vdash e; S; H; K; L_i; L'_E \quad \text{By assumption} \]

\[ (S, H, K, L_i, L'_E) \sim (DO, DD, DE) \quad \text{By assumption} \]

\[ \emptyset \vdash e; S; H; K; L_i; L'_E \quad \text{By assumption} \]

$H'[\ell] = O_C = (C<\overline{D}>) \in DO$

and $\forall \theta, k \in \overline{D} K[\theta, d_j] = D_j = (D_{id}, \text{qual}(\theta, d_j)) \in \text{rng}(DD)$

and $\forall d_i \in \text{domains}(C<\overline{D}>)$

\[ K[\theta, d_i] = D_i = (D_{id}, C::d_i) \quad \{(O_C, C::d_i) \rightarrow D_i\} \in DD \quad \text{By DF-APPROX} \]

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

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

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

\[ H; K; L_i; L'_E \vdash O_k \in \text{irLookup}(T_k) \]

$E_k \in L_i[[\ell_{src}, \theta]] \quad E_k = (H[\ell_{src}], H[\theta], O_k, Imp) \in \text{DE} \quad \text{By DF-APPROX} \]

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

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

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

\[ H; K; L_i; L'_E \vdash O_k \in \text{irLookup}(T_k) \]

$E_k \in L_E[[\theta, \ell_{dst}]] \quad E_k = (H[\theta], H[\ell_{dst}], O_k, Exp) \in \text{DE} \quad \text{By DF-APPROX} \]
\[ H' = H, K' = K, L'_1 = L_1 \]
\[ S[\ell] = C_\ell < \varphi > (\varphi) \quad \text{fields}(C_\ell < \varphi >) = \bar{T} \bar{T} \]
\[ S' = S[\ell \mapsto C_\ell < \varphi > ([v/v_1] \varphi)] \]
\[ O = H[\theta] \quad O_\ell = H[\ell] \quad O_v = H[v] \quad T_i < T \]
\[ E = (O, O_\ell, O_v, \text{Exp}) \in \text{DE} \quad H; K; L_1; L_E \vdash O_v \in \text{irLookup}(T_i) \]
\[ L_E' = L_E[(\theta, \ell) \mapsto \cup \{ E \}] \]
\[ \exists \Sigma' \supset \Sigma \quad \text{and} \quad T' < T \quad \exists \emptyset, \Sigma', \emptyset \vdash e' : T' \quad \text{and} \quad \Sigma' \vdash S' \]
\[ \Sigma'[\ell] = C < \bar{T} \bar{d} > \]
\[ \forall \ell_{\text{dst}} \in \text{dom}(H'), \text{fields}(\Sigma'[\ell_{\text{dst}}]) = \bar{T}_{\text{dst}} \bar{T}, \]
\[ \forall m. \text{mtype}(m, C < \bar{T} \bar{p} >) = \bar{T} \to T_R \]
\[ \forall O_k. H'; K'; L_1'; L_E' \vdash O_k < \text{irLookup}(T_k) \]
\[ E_k \in L_E'[(\theta, \ell_{\text{dst}})] = (H'[\theta], H'[\ell_{\text{dst}}], O_k, \text{Exp}) \in \text{DE} \]
\[ (S', H', K', L'_1, L'_E) \sim (\text{DO}, \text{DD}, \text{DE}) \]
\[ \emptyset, \emptyset, G \vdash O e' \]
This proves (2).

\[ G \vdash C_T, H \Sigma \]
\[ \forall \ell \in \text{dom}(S), \Sigma[\ell] = C_\ell < \varphi > \]
\[ H[\ell] = O_\ell = (C_\ell < \bar{T}_\ell >) \in \text{DO} \]
\[ \forall m. \text{mbody}(m, C_\ell < \varphi >) = (\varphi : T, e_R) \]
\[ (\varphi : T, \text{this} : C_\ell < \varphi >), \emptyset, G \vdash O_\ell e_R \]
\[ H' = H \]
\[ S[\ell] = C_\ell < \varphi > (\varphi) \quad \text{fields}(C_\ell < \varphi >) = \bar{T} \bar{T} \]
\[ S' = S[\ell \mapsto C_\ell < [v/v_1] \varphi >] \]
\[ G \vdash C_T, H' \Sigma' \]
This proves (3).
Case Ir-Invk: \( e = \ell.m(\tau) \), and \( e' = \ell \triangleright [\tau/\ell, \ell/\text{this}] e_R \)

We have:

\[
\ell : C<\ell.d> \quad \text{mtype}(m, C<\ell.d>) = T \rightarrow T_R' \quad \text{mtypeDec}(m, C) = T_f \rightarrow T_R
\]

\[
G \vdash_o \text{import}(C<\ell.d>, T_R)
\]

\[
\forall k \in 1..|\ell| \quad v_k : T_a \quad T_a <: T_k' \quad G \vdash_o \text{export}(C<\ell.d>, T_a)
\]

\[
\Gamma, \ell, G \vdash_o \ell \quad \ell, \ell, G \vdash_o \tau
\]  

[DF-INVK]

\[
[S][\ell] = C<\ell.d> \quad \text{mbody}(m, C<\ell.d>) = (\tau, e_R)
\]

\[
O = H[e] \quad O_t = H[\ell] 
\]

\[
\text{mtype}(m, C<\ell.d>) = T \rightarrow T_R
\]

\[
H; K; L_I; L_E \vdash O_t \in \text{irLookup}(T_R) \quad E' = \{ O_t, O_t, \text{Imp} \} \in DE \quad L_I' = L_I[[\ell, \theta] \mapsto \cup \{ E' \}]
\]

\[
\forall k \in 1..|\ell| \quad O_k = H[v_k] \quad H; K; L_I; L_E \vdash O_k \in \text{irLookup}(T_k) \quad T_k = \overline{T}
\]

\[
E_k = \{ O, O_t, O_k, \text{Exp} \} \in DE \quad L_E' = L_E[[\ell, \theta] \mapsto \cup \{ E_k \}]
\]

[IR-INVK]

\[
G = \{ D_O, D_D, D_E \} \quad \forall \ell \in \text{dom}(S), \Sigma[\ell] = C<\tau> \quad H[\ell] = O = \{ C<\overline{D}> \} \in DO
\]

\[
\forall m. \Sigma[\ell] = C<\tau> \quad \{ \tau, \text{this}; C<\tau> \}, \theta, G \vdash_o e_R
\]  

[DF-SIGMA]

To Show:

1. \((S', H', K', L_I', L_E') \sim (DO, DD, DE)\)
2. \(\theta, \theta, G \vdash_o e'\)
3. \(G \vdash_o \tau; \theta; S \quad \theta \vdash e; S; H; K; L_I; L_E \sim G\)

By assumption

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

By assumption

Since \(\Sigma \vdash S\)

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

and \(\forall \ell, d_j \in \overline{\ell.d} K[\ell, d_j] = D_j = (D, \text{qual} (\ell, d_j)) \in \text{rng}(DD)\)

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

\[
K[\ell, d_i] = D_i = (D, \text{qual} (\ell, d_i)) \} \{ (O_C, C::d_i) \rightarrow D_i \} \in DD
\]

By Df-Approx

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

\[
\forall m. \text{mtype}(\ell, \Sigma[\ell_{\text{src}}]) = \overline{T} \rightarrow T_R
\]

\[
\forall \ell_k \in \{ \overline{T_{\text{src}}} \} \cup \{ \overline{T_R} \}
\]

\[
H; K; L_I; L_E \vdash O_k \in \text{irLookup}(T_k)
\]

\[
E_k' = L_I[[\ell_{\text{src}}, \theta]] \quad E_k' = \{ H[\ell_{\text{src}}], H[\ell], O_k, \text{Imp} \} \in DE
\]

By Df-Approx

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

\[
\forall m. \text{mtype}(\ell, \Sigma[\ell_{\text{dst}}]) = \overline{T} \rightarrow T_R
\]

\[
\forall \ell_k \in \{ \overline{T_{\text{dst}}} \} \cup \{ \overline{T} \}
\]

\[
H; K; L_I; L_E \vdash O_k \in \text{irLookup}(T_k)
\]

\[
E_k = L_E[[\ell, \ell_{\text{dst}}]] \quad E_k = \{ H[\ell], H[\ell_{\text{dst}}], O_k, \text{Exp} \} \in DE
\]

By Df-Approx
\[ S' = S \quad H' = H \quad K' = K \]
\[ S[\ell] = C_l <\ell, \varphi > (\varphi, e_R) \]
\[ H[\ell] = O \quad H[\ell] = O_\ell \]
\[ mtype(m, C_l <\ell, \varphi >) = T \to T_R \quad T_R = C_K <\varphi> \]
\[ \forall \ell, H, K, L_i, L_E \models O, \text{ irLookup}(T_R) \quad E' = \langle O_\ell, O_i, Imp \rangle \in DE \]
\[ L_i = L_i[\langle \ell, \theta \rangle \mapsto \{ E' \}] \]
\[ \forall k \in L_k[\langle \ell, \theta \rangle \mapsto \{ E_k \}] \]
\[ \exists \Sigma' \supseteq \Sigma \text{ and } T' <: T \text{ s.t. } \emptyset, \Sigma', \theta \vdash e' : T' \text{ and } \Sigma' \vdash S' \]
\[ \Sigma'[\ell] = C_l \ell \]
\[ \forall \ell_{src} \in \text{dom}(H') \quad \text{fields}(\Sigma'[\ell_{src}]) = T_{src} \]
\[ \forall m, mtype(m, \Sigma'[\ell_{src}]) = T \to T_R \]
\[ \forall \ell_{k} \in \{ T_{src} \} \cup \{ T_R \} \]
\[ \forall O_k, H', K'; L_i'; L_E' \vdash O_k \in \text{ irLookup}(T_k) \]
\[ E_k \in L_k'[\langle \ell_{src}, \theta \rangle] = \langle H'[\ell_{src}], H'[\theta], O_k, Imp \rangle \in DE \]
\[ \forall \ell_{dst} \in \text{dom}(H') \quad \text{fields}(\Sigma'[\ell_{dst}]) = T_{dst} \]
\[ \forall m, mtype(m, \Sigma'[\ell_{dst}]) = T \to T_R \]
\[ \forall \ell_{k} \in \{ T_{dst} \} \cup \{ T \} \]
\[ \forall O_k, H', K'; L_i'; L_E' \vdash O_k \in \text{ irLookup}(T_k) \]
\[ E_k \in L_k'[\langle \theta, \ell_{dst} \rangle] = \langle H'[\theta], H'[\ell_{dst}], O_k, Exp \rangle \in DE \]
\[ (S', H', K', L_i', L_E') \sim (DO, DD, DE) \]

This proves (1).

\[ \emptyset, \emptyset, G \vdash \emptyset \]
\[ e = \ell.m(\varphi) \quad e_0 = \ell \quad \varphi = \varphi \]
\[ e' = \ell \triangleright \varphi/\varphi, \ell/\text{this}\]e_R
\[ \emptyset, \Sigma, \emptyset \vdash e' : T \]
\[ \exists \Sigma' \supseteq \Sigma \text{ and } T' <: T \text{ s.t. } \emptyset, \Sigma', \emptyset \vdash e' : T' \text{ and } \Sigma' \vdash S' \]
\[ e_0 : T_0 \quad T_0 = C_l <\varphi> \]
\[ mtype(m, C_l <\varphi>) = T \to T_R \]
\[ \emptyset, \emptyset, G \vdash e_0 \]
\[ \emptyset, \emptyset, G \vdash \varphi \]
\[ \{ \varphi : T, \text{this} : C_l<\alpha, \varphi>, \Sigma, \emptyset \vdash e_R : T_R \quad T_R <: T \}
\[ S[\ell] = C_l <d, \varphi>(\varphi) \]
\[ mbody(m, C_l <d, \varphi>) = (\varphi, e_R) \]
\[ \Sigma[\ell] = C_l <d, \varphi> = T_0 \]
\[ e_0 : C_l <d, \varphi> \]
\[ mtype(m, C_l <d, \varphi>) = T \to T_R \]
\[ \varphi : T_a \]
\[ T_a <: \varphi/\varphi, \ell/\text{this}\]

there are some \( D <\varphi > \) and \( T_R \) so that:
\[ T_R <: T_R \text{ and } C_l d, \varphi > <: D <\varphi > \]
so that \( \{ \varphi : T, \text{this} : D <\varphi >, \Sigma, \emptyset \vdash e_R : T'_R \)
there exists \( T_S, T_S <: T'_R \text{ s.t. } [\varphi/\varphi, \ell/\text{this} e_R : T_S \]
\[ T_S <: T_R \text{ and } T'_R <: T_R \]
\[ T_S <: T_R \]

Take \( T = T' = T_R \) in FDJ Preservation

By sub-derivation of IR-INVk
By sub-derivation of IR-INVk
By sub-derivation of IR-INVk
By sub-derivation of IR-INVk
By sub-derivation of IR-INVk
By sub-derivation of IR-INVk
By sub-derivation of IR-INVk
By sub-derivation of IR-INVk
By FDJ Type Preservation

By assumption
By assumption
By assumption
By assumption
By FDJ Type Preservation
By sub-derivation of DF-INVk
By sub-derivation of DF-INVk
By sub-derivation of DF-INVk
By sub-derivation of DF-INVk
By FDJ MethOK:
By sub-derivation of IR-INVk
By sub-derivation of IR-INVk
Since \( e_0 = \ell \), by T-Store
Since \( e_0 = \ell \), by T-Store
Since \( e_0 = \ell \), by T-Store
By inversion
For some \( T_a \) and \( T \)
By Method Lemma
By Method Lemma
By Method Lemma
Since term substitution preserves typing
By above
By transitivity of <:
\[
\{ \mathtt{T}, \text{this} : C_{t<d, \overline{d}^o}, \emptyset, G \vdash_{O_C} e_R \} \\
O_C = H[\ell] \\
\emptyset, \emptyset, G \vdash_{O} \ell \\
\emptyset, \emptyset, G \vdash_{O_C} [v/\ell, \ell/\text{this}] e_R \\
\emptyset, \emptyset, G \vdash_{O} \ell [v/\ell, \ell/\text{this}] e_R
\]

This proves (2).

\[
G \vdash_{CT,H} \Sigma \\
S' = S, H' = H \\
G \vdash_{CT,H'} \Sigma'
\]

This proves (3).

**Case Ir-Context:** \( e = \ell \triangleright v \) and \( e' = v \)

We have:

\[
O_C = H[\ell] \\
\emptyset, \emptyset, G \vdash_{O_C} e \\
\emptyset, \emptyset, G \vdash_{O} \ell \triangleright e \\
\emptyset, \emptyset, G \vdash_{O} v \triangleright S \\
\emptyset, \emptyset, G \vdash_{O} H; K; L_i; L_E \triangleright G \\
G = \langle DO, DD, DE \rangle \\
\forall \ell \in \text{dom}(S), \Sigma[\ell] = C\langle \overline{\ell} \rangle \\
H[\ell] = O = \langle C\langle \overline{D} \rangle \rangle \in DO \\
\forall m. \text{mbody}(m, C\langle \overline{\ell} \rangle) = (\overline{\ell}; e_R) \\
\{ \overline{\ell}; \text{this} : C\langle \overline{\ell} \rangle \}, \emptyset, G \vdash_{O} e_R
\]

This proves (1).

\[
\emptyset, \emptyset, G \vdash_{O} e' \\
\emptyset, \emptyset, G \vdash_{O} \ell \triangleright e \\
G \vdash_{CT,H} \Sigma
\]

This proves (2).

\[
(S, H, K, L_i, L_E) \sim (DO, DD, DE) \\
S' = S, H' = H, K' = K, L'_i = L_i, L'_E = L_E
\]

This proves (3).

**Case Irc-New:** \( e = \text{new} C\langle \overline{\ell} \rangle(v_{1,i+1..n}), \) and \( e' = \text{new} C\langle \overline{\ell} \rangle(v_{1,i+1..n}) \)

We have:

\[
G = \langle DO, DD, DE \rangle \\
O = C_{\text{this}}\langle \overline{D}_C \rangle \\
O_C = \langle C\langle \overline{D} \rangle \rangle \\
\forall i \in 1..|\overline{\ell}| \\
G \vdash_{O_D} D_i \in \text{findD}(C_{\text{this}}; p_i) \\
O_C \in \langle C\langle \overline{D} \rangle \rangle \\
\{O_C, \text{qual}(p_i)) \rightarrow D_i \} \subseteq DD \\
G \vdash_{O_D} \text{dparams}(C, O_C) \\
G \vdash_{O_D} \text{ddomains}(C, O_C) \\
\forall m \in \text{md mbody}(m, C\langle \overline{\ell} \rangle) = (\overline{\ell}; e_R) \\
C\langle \overline{D} \rangle \notin \Upsilon \Rightarrow \{ \overline{\ell}; \text{this} : C\langle \overline{\ell} \rangle \}, \Upsilon \cup \{C\langle \overline{D} \rangle \}, G \vdash_{O_C} e_R
\]

This proves (3).
\[
\begin{align*}
\theta \vdash e_i; S; H; K; L_i; L_E \rightsquigarrow G \quad e_i; S'; H'; K'; L_i'; L_E' & \quad \text{[Irc-New]} \\
\theta \vdash \text{new } C < \mathfrak{P} > (v_{1..i-1}, e_i, e_{i+1..n}); S; H; K; L_i; L_E \rightsquigarrow G \\
\text{new } C < \mathfrak{P} > (v_{1..i-1}, e_i, e_{i+1..n}); S'; H'; K'; L_i'; L_E' & \\
G = (DO, DD, DE) \quad \forall \ell \in \text{dom}(S), \Sigma[\ell] = C < \mathfrak{P} > \quad H[\ell] = O = \langle C < \mathfrak{P} > \rangle \in DO \\
\forall m. \ mbody(m, C < \mathfrak{P} >) = (\forall \mathfrak{T}, e_R) \quad (\forall \mathfrak{T}, \text{this } C < \mathfrak{P} >), \theta, G \vdash_O e_R & \quad \text{[Df-Sigma]}
\end{align*}
\]

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

This proves (1).

\[
\begin{align*}
\theta \vdash e_i; S; H; K; L_i; L_E \rightsquigarrow G \quad e_i; S'; H'; K'; L_i'; L_E' & \quad \text{By sub-derivation of Irc-New} \\
\emptyset, \emptyset, G \vdash_O e' & \quad \text{By induction hypothesis}
\end{align*}
\]

This proves (2).

\[
\begin{align*}
\theta \vdash e_i; S; H; K; L_i; L_E \rightsquigarrow G \quad e_i; S'; H'; K'; L_i'; L_E' & \quad \text{By sub-derivation of Irc-New} \\
G \vdash_{CT,H} \Sigma' & \quad \text{By induction hypothesis, take } \Sigma' = \Sigma
\end{align*}
\]

This proves (3).

**Case Irc-Read:** \(e = e_0.f_k\), and \(e' = e'_0.f_k\).

We have:
\[
\begin{align*}
e_0 : C < \mathfrak{P} > \quad (T_k f_k) \in \text{fieldDecls}(C) & \\
G \vdash_O \text{import}(C < \mathfrak{P} >, T_k) & \\
\Gamma, \mathfrak{T}, G \vdash_O e_0 & \quad \text{[Df-Read]} \\
\theta \vdash e_0; S; H; K; L_i; L_E \rightsquigarrow G \quad e_0; S'; H'; K'; L_i'; L_E' & \quad \text{[Irc-Read]} \\
\theta \vdash e_0.f_i; S; H; K; L_i; L_E \rightsquigarrow G & \\
\end{align*}
\]

\[
\begin{align*}
G = (DO, DD, DE) \quad \forall \ell \in \text{dom}(S), \Sigma[\ell] = C < \mathfrak{P} > \quad H[\ell] = O = \langle C < \mathfrak{P} > \rangle \in DO \\
\forall m. \ mbody(m, C < \mathfrak{P} >) = (\forall \mathfrak{T}, e_R) \quad (\forall \mathfrak{T}, \text{this } C < \mathfrak{P} >), \theta, G \vdash_O e_R & \quad \text{[Df-Sigma]}
\end{align*}
\]

To Show:
(1) \((S', H', K', L_i', L_E') \sim (DO, DD, DE)\)
(2) \(\emptyset, \emptyset, G \vdash_O e'\)
(3) \(G \vdash_{CT,H} \Sigma'\)
\[ \theta \vdash e_0; S; H; K; L_i; L_E \leadsto_G e'_0; S'; H'; K'; L'_i; L'_E \]

\[ (S', H', K', L'_i, L'_E) \sim (G) \]

By sub-derivation of IRC-Read

This proves (1).

\[ \theta \vdash e_0; S; H; K; L_i; L_E \leadsto_G e'_0; S'; H'; K'; L'_i; L'_E \]

\[ \emptyset, \emptyset, G \vdash e'_0 \]

\[ \emptyset, \emptyset, G \vdash e'_0, f_k \]

This proves (2).

\[ \theta \vdash e_0; S; H; K; L_i; L_E \leadsto_G e'_0; S'; H'; K'; L'_i; L'_E \]

By sub-derivation of IRC-Read

\[ G \vdash_{CT, H'} \Sigma' \]

By induction hypothesis, take \( \Sigma' = \Sigma \)

This proves (3).

\[ \text{Case IRC-Write-Rcv: } e = (e_0, f_k = e_1), \text{ and } e' = (e'_0, f_k = e_1). \]

We have:

\[ e_0 : C <T_k > \]

\[ e_1 : C_1 \langle \overline{T_k} \rangle \]

\[ G \vdash export(C \langle T_k \rangle, C_1 \langle \overline{T_k} \rangle) \]

\[ \Gamma, \overline{T_k}, G \vdash e_0 \]

\[ \Gamma, \overline{T_k}, G \vdash e_1 \]

[DF-Write]

\[ \theta \vdash e_0; S; H; K; L_i; L_E \leadsto_G e'_0; S'; H'; K'; L'_i; L'_E \]

\[ \theta \vdash e'_0, f_i = e_1; S; H; K; L_i; L_E \leadsto_G \]

\[ e'_0, f_i = e_1; S'; H'; K'; L'_i; L'_E \]

[IRC-WriteRcv]

\[ G = \langle DO, DD, DE \rangle \quad \forall \ell \in \text{dom}[S], \Sigma[\ell] = C <T_k > \quad H[\ell] = O = \langle C <\overline{T_k} \rangle \in DO \]

\[ \forall m, \mu_{body}(m, C <T_k >) = (T, \overline{T}, e_R) \]

\[ \{T, \overline{T}, this : C <\overline{T_k} >, \theta, G \vdash e_R \} \]

[DF-Sigma]

To Show:

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

(2) \( \emptyset, \emptyset, G \vdash e' \)

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

\[ \theta \vdash e_0; S; H; K; L_i; L_E \leadsto_G e'_0; S'; H'; K'; L'_i; L'_E \]

By sub-derivation of IRC-Write-Rcv

This proves (1).

\[ \theta \vdash e_0; S; H; K; L_i; L_E \leadsto_G e'_0; S'; H'; K'; L'_i; L'_E \]

By sub-derivation of IRC-Write-Rcv

\[ \emptyset, \emptyset, G \vdash e'_0 \]

\[ \emptyset, \emptyset, G \vdash e_1 \]

\[ \emptyset, \emptyset, G \vdash e'_0, f_k = e_1 \]

By sub-derivation of IRC-Write-Rcv

By induction hypothesis

By DF-Write

This proves (2).

\[ \theta \vdash e_0; S; H; K; L_i; L_E \leadsto_G e'_0; S'; H'; K'; L'_i; L'_E \]

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:

\[
\begin{align*}
e_0 &: C < \overline{p} > \\
e_1 &: C_1 < \overline{p'} >
\end{align*}
\]

\( G \vdash_O \text{import}(C < \overline{p} >, C_1 < \overline{p'} >) \)

\[
\begin{array}{c}
\Gamma, \Upsilon, G \vdash_O e_0 \\
\Gamma, \Upsilon, G \vdash_O e_1
\end{array}
\]

\[ [\text{DF-WRITE}] \]

\[
\theta \vdash e_1; S; H; K; L_I; L_E \leadsto_G e'_1; S'; H'; K'; L'_I; L'_E \]

\[ [\text{IRC-WRITE-ARG}] \]

\[ G = \langle DO, DD, DE \rangle \quad \forall \ell \in \text{dom}(S), \Sigma[\ell] = C < \overline{p} > \\
H[\ell] = O = \langle C < \overline{D} > \rangle \in DO \]

\[ \forall m. \text{mbody}(m, C < \overline{p} >) = (\Upsilon: \overline{T}, e_R) \]

\[ \{ \Upsilon: \overline{T}, \text{this}: C < \overline{p} > \}, \emptyset, G \vdash_O e_R \]

\[ [\text{DF-SIGMA}] \]

To Show:

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

This proves (1).

\[
\theta \vdash e_1; S; H; K; L_I; L_E \leadsto_G e'_1; S'; H'; K'; L'_I; L'_E
\]

By sub-derivation of IRC-WRITE-ARG

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

By induction hypothesis

This proves (2).

\[
\emptyset, \emptyset, G \vdash_O e'_1
\]

By induction hypothesis

This proves (3).

\[ G \vdash_{CT, H} \Sigma' \]

By induction hypothesis, take \( \Sigma' = \Sigma \)

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

We have:

\[
\begin{align*}
e_0 &: C < \overline{p} > \\
mtype(m, C < \overline{p} >) &= \overline{T}_R
\end{align*}
\]

\[ mtypeDecl(m, C) = \overline{T}_R \]

\[ G \vdash_O \text{import}(C < \overline{p} >, T_R) \]

\[ \forall k \in 1..|\overline{v}| \quad e_k : T_a \\
T_a : T'_k \\
G \vdash_O \text{import}(C < \overline{p} >, T_a) \]

\[
\begin{array}{c}
\Gamma, \Upsilon, G \vdash_O e_0 \\
\Gamma, \Upsilon, G \vdash_O \overline{v}
\end{array}
\]

\[ [\text{DF-INVK}] \]

\[
\emptyset, \emptyset, G \vdash O e_0.m(\overline{v})
\]

\[ [\text{IRC-RECVINVK}] \]

\[ G = \langle DO, DD, DE \rangle \quad \forall \ell \in \text{dom}(S), \Sigma[\ell] = C < \overline{p} > \\
H[\ell] = O = \langle C < \overline{D} > \rangle \in DO \]

\[ \forall m. \text{mbody}(m, C < \overline{p} >) = (\Upsilon: \overline{T}, e_R) \]

\[ \{ \Upsilon: \overline{T}, \text{this}: C < \overline{p} > \}, \emptyset, G \vdash_O e_R \]

\[ [\text{DF-SIGMA}] \]

To Show:

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

$\theta \vdash e_0; S; H; K; L_I; L_E \vdash_G e'_0; S'; H'; K'; L'_I; L'_E$

By sub-derivation of IRC-RECVINVK

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

By induction hypothesis

This proves (1).

$\theta \vdash e_0; S; H; K; L_I; L_E \vdash_G e'_0; S'; H'; K'; L'_I; L'_E$

By sub-derivation of IRC-RECVINVK

$\emptyset, \emptyset, G \vdash_0 e'_0$

By induction hypothesis

$\emptyset, \emptyset, G \vdash_0 \mathbf{\overline{e}}$

By DF-INVK

$\emptyset, \emptyset, G \vdash_0 e'_0.m(\mathbf{\overline{e}})$

By DF-INVK

This proves (2).

$\theta \vdash e_0; S; H; K; L_I; L_E \vdash_G e'_0; S'; H'; K'; L'_I; L'_E$

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:

$e_0 : C<\mathbf{\overline{e}}> \quad \text{mtype}(m, C<\mathbf{\overline{e}}>) = \mathbf{\overline{T}} \rightarrow T^\theta_0 \quad \text{mtypeDecl}(m, C) = \mathbf{\overline{R}} \rightarrow T^\theta_R$

$\forall k \in 1..|\mathbf{\overline{e}}| \; e_k : T^\theta_0 \quad T^\theta_a < T^\theta_k \quad G \vdash_0 \text{export}(C<\mathbf{\overline{e}}>, T_a)$

$\Gamma, \mathbf{\overline{T}}, G \vdash_0 e_0 \quad \Gamma, \mathbf{\overline{T}}, G \vdash_0 \mathbf{\overline{e}}$

[DF-INVK]

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

[IRC-ARGINVK]

$\overline{e} \vdash v.m(v_1..i-1, e_i, e_{i+1..n}); S; H; K; L_I; L_E \vdash_G v.m(v_1..i-1, e_i', e_{i+1..n}); S'; H'; K'; L'_I; L'_E$

$G = (DO, DD, DE) \quad \forall \ell \in \text{dom}(S), \Sigma[\ell] = C<\mathbf{\overline{e}}> \quad H[\ell] = O = (C<\mathbf{\overline{D}}>, E) \in DO$

$\forall m. \text{mbody}(m, C<\mathbf{\overline{e}}>) = (\mathbf{\overline{T}}, I, \mathbf{\overline{e}}) \quad (\mathbf{\overline{T}}, \mathbf{\overline{T}}, \text{this}: C<\mathbf{\overline{e}}>), \emptyset, G \vdash_0 \mathbf{\overline{e}}$

[DF-SIGMA]

To Show:

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

(2) $\emptyset, \emptyset, G \vdash_0 e'$

(3) $G \vdash_{CT,H'} \Sigma'$
\[ \theta \vdash \epsilon; S; H; K; L_i; L_E \leadsto_G \epsilon'; S'; H'; K'; L'_i; L'_E \]

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

This proves (1).

\[ \theta \vdash \epsilon; S; H; K; L_i; L_E \leadsto_G \epsilon'; S'; H'; K'; L'_i; L'_E \]

\(\emptyset, \emptyset, G \vdash_O \epsilon'_i\)

This proves (2).

\[ \theta \vdash \epsilon; S; H; K; L_i; L_E \leadsto_G \epsilon'; S'; H'; K'; L'_i; L'_E \]

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

This proves (3).

**Case Irc-Context:** \( \epsilon = \ell \triangleright \epsilon_0 \), and \( \epsilon' = \ell \triangleright \epsilon'_0 \).

We have:

\[ \Gamma, \bar{\epsilon}, G \vdash_{O_G} \epsilon \]

\[ \begin{array}{l}
\vdash \epsilon; S; H; K; L_i; L_E \leadsto_G \epsilon'; S'; H'; K'; L'_i; L'_E \\
\hline
\emptyset \vdash \ell \triangleright \epsilon; S; H; K; L_i; L_E \leadsto_G \ell \triangleright \epsilon'; S'; H'; K'; L'_i; L'_E
\end{array} \]

\[ G = (DO, DD, DE) \]

\[ \forall \eta \in \text{dom}(S), \Sigma[\eta] = C<\bar{\eta}> \]

\[ H[\eta] = O = (C<\bar{D}>) \in DO \]

\[ \exists m. \text{mbody}(m, C<\bar{\eta}>) = (\bar{\tau}, \bar{\epsilon}_R) \]

\[ \begin{array}{l}
\vdash \ell \triangleright \epsilon; S; H; K; L_i; L_E \leadsto_G \ell \triangleright \epsilon'; S'; H'; K'; L'_i; L'_E \\
\hline
G \vdash_{CT, H} \Sigma
\end{array} \]

By sub-derivation of **Irc-ArgInvk**

By induction hypothesis

By sub-derivation of **Irc-ArgInvk**

By induction hypothesis

By induction hypothesis, take \( \Sigma' = \Sigma \)

By sub-derivation of **Irc-ArgInvk**

By induction hypothesis

By induction hypothesis

By sub-derivation of **Irc-ArgInvk**

By induction hypothesis

By induction hypothesis, take \( \Sigma' = \Sigma \)

\[ \Box \]

34
3.5 Theorem: Dataflow Progress

If

\[ \emptyset, \Sigma, \theta \vdash e : T \]

\[ \Sigma \vdash S \]

\[ G \vdash_{CT,H} \Sigma \]

\[ \emptyset, \emptyset, G \vdash \sigma \]

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

then

either \( e \) is a value

or else \( \theta \vdash e \);

\[ S \xrightarrow{\quad} H; K; L_I; L_E \]

Proof: We prove progress by derivation of \( \emptyset, \emptyset, G \vdash \sigma \), 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 : \( e = \text{new } C < \overline{\tau} > (s) \).

Subcase \( \tau = \overline{\tau} \) that is \( e = \text{new } C < \overline{\tau} > (s) \). Take \( e' = t \), then Ir-New can apply.

\[ \ell \notin dom(S) \quad S' = S[\ell \mapsto C < \overline{\tau} > (s)] \]

\[ G = (DO, DD, DE) \]

\[ \tau = \overline{\tau}; d \]

\[ \forall i \in 1..|\overline{\ell}.d| \quad D_i = K[\ell_i, d_i] \]

\[ \ell_i \in \text{dom}(H) \text{ s.t. } H[\ell_i] = O_i \quad D_i = DD[O_i, \text{qual}(\ell_i, d_i)] \]

\[ \ell_i \in \text{dom}(H) \text{ s.t. } H[\ell_i] = O_i \quad D_i = DD[O_i, \text{qual}(\ell_i, d_i)] \]

\[ O_C = (C < \overline{\nu} >) \quad O_C \in DO \quad H' = H[\ell \mapsto O_C] \]

\[ \forall (\text{domain } d_j) \in \text{domains}(C < \overline{\nu} >) \quad D_j = DD[(O_C, C::d_j)] \quad K' = K[\ell, d_j \mapsto D_j] \]

\[ \theta \vdash \text{new } C < \overline{\nu} > (s); H; K; L_I; L_E \sim G \quad e'; S'; H'; K'; L_I; L_E \]

To show:

1. \( \forall i \in 1..|\overline{\ell}.d| \quad D_i = K[\ell_i, d_i] \)
2. \( O_C = (C < \overline{\nu} >) \quad O_C \in DO \)
3. \( \forall i \in 1..|\overline{\ell}.d| \quad H[\ell_i] = O_i \quad D_i = DD[(O_i, \text{qual}(\ell_i, d_i))] \)
4. \( \forall d_j \in \text{domains}(C < \overline{\nu}.d>) \quad D_j = DD[(O_C, C::d_j)] \)
\[(S, H, K, L_1, L_E) \sim (D_O, D_D, D_E)\]

By assumption

\[\forall \ell \in \text{dom}(S), \Sigma[\ell] = C_{\ell.d}\]

\[\forall \ell \in \text{dom}(S), \Sigma[\ell] = C_{\ell.d}\]

\[\implies H[\ell] = O_C = (C_{\ell.d}) \in D_O\]

and \[\forall \ell', d_j \in \ell.d \ K[\ell', d_j] = D_j = (D_{d_j}, \text{qual}(\ell', d_j)) \in \text{rng}(D_D)\]

and \[\forall d_i \in \text{domains}(C_{\ell.d}) \quad K[\ell.d_i] = D_i = (D_{d_i}, C_{\ell.d_i}) \{ (O_C, C_{\ell.d_i}) \mapsto D_i \} \in D_D\]

and \[\forall \ell_{\text{src}} \in \text{dom}(H), \quad \text{fields}(\Sigma[\ell_{\text{src}}]) = T_{\text{src}}\]

∀m. \text{mtype}(m, \Sigma[\ell_{\text{src}}]) = T \implies T_R

By sub-derivation of \text{Df-New}

\[\emptyset, \emptyset, G \vdash_O \text{new} C_{\ell.d}(\forall)\]

By sub-derivation of \text{Df-New}

\[\forall i \in 1..|\ell.d| \quad G \vdash_O D_i \in \text{findD}(C_{\text{this} :: \ell_i.d_i})\]

By sub-derivation of \text{Df-New}

\[O_C = (C_{\ell.d}) \quad \{ O_C \} \subseteq D_O\]

By sub-derivation of \text{Df-New}

\[G \vdash_O \text{dparams}(C, O_C)\]

By sub-derivation of \text{AUX-ALPHA}

\[\{ (O_C, C_{::a_i}) \mapsto D_i \} \subseteq D_D \quad a_i \in \text{params}(C)\]

By sub-derivation of \text{Df-New}

\[G \vdash_O \text{ddomains}(C, O_C)\]

By sub-derivation of \text{Df-New}

This proves (4).

Sub-subcase \(\ell_i' \neq \theta, \ell_i', d_i\) is a domain of some object in the context of \(\theta\)

That is, \(\exists n \in \text{dom}(H)\)

such that:

\[D_D[(H[n], C_n :: d_i)] = D_i\]

\[G \vdash_O H[n] \in \text{lookup}(\Sigma[n])\]

by subderivation of \text{AUX-FIND-PUBLIC}

This proves (3).

Take \(\ell_i = n\)
Sub-case $\ell'_i \neq \theta$, $\ell'_i, d_i$ substitutes the $j^{th}$ formal domain parameters of $C_{\text{this}}$, where $\Sigma[\theta] = C_{\text{this}}<\ell_{\text{this}}, d>$

That is:

\[
\begin{align*}
O &= \langle C_{\text{this}}<D'\rangle \\
DD[(O, C_{\text{this}}; \alpha_i)] &= D'_j \\
DD[(O, \text{qual}(\ell'_i, d_i))] &= D'_j
\end{align*}
\]

by subderivation of $\text{Df-New}$

\[
\forall i \in 1..|\ell'_i| \quad G \vdash O \quad D_i \in \text{findD}(C_{\text{this}}; \ell'_i, d_i)
\]

By sub-derivation of $\text{Df-New}$

$D_i = DD[(O, \text{qual}(\ell'_i, d_i))] = DD[(O, C_{\text{this}}; \alpha_j)] = D'_j$

By sub-derivation of $\text{AUX-FINDDD}$

Take $\ell_i = \theta$

Sub-case $\ell'_i = \theta$, $\ell'_i, d_i$ is a domain of $\theta$

$G \vdash O \quad D_i \in \text{findD}(C_{\text{this}}; \ell'_i, d_i)$

By above

$D_i = DD[(O, C; d_i)]$

By sub-derivation of $\text{AUX-FINDThis}$

$d_i \in \text{domains}(\Sigma[\ell_i])$

By inversion of $\text{QUAL-Var}$

$\text{qual}(\ell'_i, d_i) = C; d_i$

By above

$D_i = DD[(O, \text{qual}(\ell'_i, d_i))]$

This proves (3).

Take $\ell_i = \theta$

Sub-case $d_i = \text{shared}$

$G \vdash O \quad D_i \in \text{findD}(C_{\text{this}}; \ell'_i, d_i)$

By above

$D_i = D_{\text{shared}} = DD[(O, \text{world}_d; \text{shared})]$

By sub-derivation of $\text{AUX-FINDSHARED}$

This proves (3).

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

\[
\begin{align*}
\theta &\vdash e_i; S; H; K; L_i; L_E \rightsquigarrow G \quad e'_i; S'; H'; K'; L'_i; L'_E \\
\theta &\vdash \text{new } C<\overline{p}>(v_{1..i-1}, e_i, e_{i+1..n}); S; H; K; L_i; L_E \rightsquigarrow G \\
\text{new } C<\overline{p}>(v_{1..i-1}, e_i, e_{i+1..n}) &\rightsquigarrow G
\end{align*}
\]

By sub-derivation of $\text{Dr-New}$

By induction hypothesis

By inversion of $\text{IRC-New}$

This proves the case.

Case DF-VAR $\colon e = x$.

Not applicable since variable is not a closed term.

Case DF-LOC $\colon e = \ell$.

$e$ 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:

\[
\begin{align*}
\text{Case DF-READ: } & e = e_0.f_i. \text{ There are two subcases to consider depending on whether the receiver } e_0 \text{ is a value.} \\
\text{Subcase } e_0 = \ell. \text{ Then } e = \ell.f_i. \\
\text{From Ir-READ:} & \\
& S[\ell] = C<\mathcal{F}>(\mathcal{F}) \quad \text{fields}(C<\mathcal{F}>) = T \quad \mathcal{T} \\
& O = H[\theta] \quad O_\ell = H[\ell] \quad O_v = H[v_i] \quad T_i \in \mathcal{T} \\
& E = (O_\ell, O_v, \text{Imp}) \in \text{DE} \quad H; K; L_I; L_E \vdash O_v \in \text{irLookup}(T_i) \quad L'_I = L_I[(\ell, \theta) \mapsto \{ E \}] \\
& \theta \vdash \ell.f_i; S \quad H; K; L_I; L_E \vdash S.H: H; K; L_I; L_E
\end{align*}
\]

To show:

1. \( O = H[\theta] \)
2. \( O_\ell = H[\ell] \)
3. \( O_v = H[v_i] \quad T_i \in \mathcal{T} \quad H, K, L_I, L_E \vdash O_v \in \text{irLookup}(T_i) \quad E = (O_\ell, O_v, \text{Imp}) \in \text{DE} \)

\[ G \vdash_{ct,H} \Sigma \]

By assumption

\[ \forall \ell_1 \in \text{dom}(S), \Sigma[\ell_1'] = C<\mathcal{F}'> \]

By sub-derivation of Df-Sigma

\[ H[\ell_1'] = O' = \langle C<\mathcal{D}'> \rangle \in \text{DO} \]

By sub-derivation of Df-Sigma

\[ H[\theta] = O = \langle C<\mathcal{D}'> \rangle \in \text{DO} \]

Since \( \theta \in \text{dom}(S) \)

\[ H[\ell] = O_\ell = \langle C_i<\mathcal{D}_i'> \rangle \in \text{DO} \]

Since \( \ell \in \text{dom}(S) \)

\[ H[v_i] = O_v = \langle C_v<\mathcal{D}_v'> \rangle \in \text{DO} \]

Since \( v \in \text{dom}(S) \)

This proves (1), and (2).

\[ (S, H, K, L_I, L_E) \sim (\text{DO}, DD, \text{DE}) \]

By assumption

\[ \forall \ell \in \text{dom}(S), \Sigma[\ell] = C<\mathcal{F}'> \]

Since \( \Sigma \vdash S \)

\[ H[\theta] = O_C = \langle C<\mathcal{D}'> \rangle \in \text{DO} \]

\[ H[\ell] = O_C = \langle C<\mathcal{D}'> \rangle \in \text{DO} \]

and \( \forall \theta', d_j \in \mathcal{D}, K[\theta', d_j] = D_i = \langle D_i, \text{qual}(\theta', d_j) \rangle \in \text{rng}(DD) \)

and \( \forall d_i \in \text{domains}(C<\mathcal{D}>) \)

By Df-Approx

\[ K[\theta, d_i] = D_i = \langle D_i, C::d_i \rangle \quad \langle (O_C, C::d_i) \rightarrow D_i \rangle \in \text{DD} \]

\[ \forall \ell_\text{src} \in \text{dom}(H), \quad \text{fields}(\Sigma[\ell_\text{src}]) = T_{\text{src}} \quad \mathcal{T} \]

\[ \forall m. \text{mtype}(m, \Sigma[\ell_\text{src}]) = T \rightarrow T_R \]

By Df-Approx

\[ \forall T_h \in \{ T_{\text{src}} \} \cup \{ T_h \} \]

\[ H; K; L_I; L_E \vdash O_\text{src} \in \text{irLookup}(T_h) \]

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

By Df-Approx

\[ \forall \ell_\text{dst} \in \text{dom}(H), \quad \text{fields}(\Sigma[\ell_\text{dst}]) = T_{\text{dst}} \quad \mathcal{T} \]

\[ \forall m. \text{mtype}(m, \Sigma[\ell_\text{dst}]) = T \rightarrow T_R \]

By Df-Approx

\[ \forall T_k \in \{ T_{\text{dst}} \} \cup \{ T \} \]

\[ H; K; L_I; L_E \vdash O_\text{dst} \in \text{irLookup}(T_k) \]

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

By Df-Approx

38
\(\emptyset, \emptyset, G \vdash_O \ell.f_i\)

**Case domain parameter:** \(p_k = \alpha_j\)

\[\exists \ell_j', d_j \in \mathcal{D} \text{ s.t. } K[v_k', d_k] = K[\ell_j', d_j] = D_j\]

\(D_{vk} = D_j = DD([H[\ell], C_\ell; \alpha_j]) = DD([H[\ell], qual(\ell_j', d_j)]) = K[\ell_j', d_j]\)

**By Params Lemma**

**Case local domain:** \(p_k = \ell.d\)

\[\exists \ell_j', d_j \in \mathcal{D} \text{ s.t. } K[v_k', d_k] = K[\ell_j', d_j] = D_j\]

\(D_{vk} = D_j = DD([H[\ell], C_\ell; d_j]) = K[\ell, d_j]\)

**By sub-derivation of Df-New**

**Case local domain:** \(p_k = \text{shared}\)

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

\(G \vdash_H[\ell] \quad D_{vk} \in \text{findD}(C_\ell; \text{qual}(v_k', d_k))\)

**By inversion of AuxFindD**

\(G \vdash_H[\ell] \quad H[v_i] \in \text{lookup}(T_i')\)

**By inversion of Aux-Lookup. Take \(O_i = H[\ell]\)**

\(\langle H[\ell], H[\theta], H[v], \text{Imp} \rangle \subseteq DE\)

**By above. Take \(O = H[\theta] \quad O_i = H[\ell] \quad O_j = H[v_i]\)**

39
\( H[v] = O_{C_v} = \langle C_v < D > \rangle \in DO \)
and \( \forall v'_j, d_j \in \mathbb{V} \cdot d \) \( K[v'_j, d_j] = D_j = \langle D_{id_j}, \text{qual}(v'_j, d_j) \rangle \in \text{rng}(DD) \)
\( \text{and } \forall d_i \in \text{domains}(C_v < v' \cdot d >) \)
\( K[v, d_i] = D_i = \langle D_{id_i}, C_v :: d_i \rangle \{ (O_{C_v}, C_v :: d_i) \mapsto D_i \} \in DD \)  
By \text{De-APPROX}
\( H; K; L_E \vdash H[v] \in \text{irLookup}(\Sigma[v]) \)
By inversion of \text{IR-Lookup}

\( \text{Take } T_k = T'_i \in T, \text{this proves (3)}. \)

**Subcase** \( e_0 = e'_0.f_i. \) That is, \( e_0 \) is not a value

From \text{IRC-READ}:

\[
\frac{\theta \vdash e_0; S; H; K; L_i; L_E \rightsquigarrow_G e'_0; S'; H'; K'; L'_i; L'_E}{\theta \vdash e'_0.f_i; S; H; K; L_i; L_E \rightsquigarrow_G e'_0; f_i; S'; H'; K'; L'_i; L'_E}
\]

\( \theta \vdash e'_0; f_i; S; H; K; L_i; L_E \rightsquigarrow_G e''_0; f_i; S'; H'; K'; L'_i; L'_E \)

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 \text{IRC-READ}

\( \text{Take } e' = e''_0.f_i. \)
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

$$S[\ell] = C<\triangledown>(\triangledown) \quad \text{fields}(C<\triangledown>) = \triangledown \triangledown$$

$$S' = S[\ell \mapsto C<\triangledown>[[v/v_0]]]$$

$$O = H[\theta] \quad O_\ell = H[\ell] \quad O_v = H[v] \quad H; K; L_I; L_E \vdash O_v \in \text{irLookup}(T_i) \quad T_i \in \overline{T}$$

$$E = \langle O, O_\ell, O_v, \text{Exp} \rangle \in DE \quad L'_E = L_E[[\theta, \ell] \mapsto \{E\}]$$

$$\theta \vdash \ell.f_i = v; S \quad H; K; L_I; L_E \models C[v/v_i] \quad H; K; L_I; L'_E$$

To show:

1. $O = H[\theta]$
2. $O_\ell = H[\ell]$
3. $O_v = H[v]; H; K; L_I; L_E \vdash O_v \in \text{irLookup}(T_i)$

$$E = \langle O, O_\ell, O_v, \text{Exp} \rangle \in DE$$

By assumption

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

By sub-derivation of DF-Sigma

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

By sub-derivation of DF-Sigma

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

Since $\theta \in \text{dom}(S)$

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

Since $\ell \in \text{dom}(S)$

$$H[v] = O_v = \langle C_v\overline{<\overline{D}>} \rangle \in DO$$

Since $v \in \text{dom}(S)$

this proves (1), and (2).

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

By assumption

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

Since $\Sigma \vdash S$

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

$$H[\theta.d_i] = D_i = \langle D_{id_i}, \text{qual}(\theta', d_i) \rangle \in \text{rng}(DD)$$

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

$$K[\theta.d_i] = D_i = \langle D_{id_i}, C::d_i \rangle \cup \{O_C, C::d_i \rightarrow D_i\} \in DD$$

41
\[\emptyset, \emptyset, G \vdash_0 \ell.f_i = v\]

Since \(e_0 = \ell \in \text{dom}(H)\) and \(e_1 = v\):

\[\ell : \Sigma[\ell] = C_t \leftarrow \exists \bar{y} \quad (T_i^\prime, f_i) \in \text{fields}(C_t \leftarrow \exists \bar{y}) = \overrightarrow{T} \overrightarrow{T} \quad T'_i = C_t \leftarrow \exists \bar{y}\]

\[v : \Sigma[v] = C_v \leftarrow \exists \bar{y} \quad \Sigma[v] < : T'_i\]

\[G \vdash_o \text{export}(\Sigma[\ell], \Sigma[v])\]

\[G \vdash_o \text{lookup}(\Sigma[\ell])\]

\[G \vdash_o \text{lookup}(\Sigma[v])\]

\[\langle O, O_v, O_j, \text{Exp} \rangle \in \text{DF}\]

\[\Sigma[\ell] = C_t \leftarrow \exists \bar{y} . d \quad H[\ell] = O \ell = (C_t \leftarrow \exists \bar{y}) \in \text{DO}\]

\[\emptyset, \emptyset, G \vdash_0 \ell\]

\[\emptyset; \Sigma; \emptyset \vdash \ell : \Sigma[\ell]\]

\[G \vdash_{H[\ell]} H[\ell] \in \text{lookup}(\Sigma[\ell])\]

\[\emptyset, \emptyset, G \vdash_o v\]

\[\emptyset; \Sigma; \emptyset \vdash v : \Sigma[v]\]

\[\Sigma[v] = C_v \leftarrow \exists \bar{y} . d \quad H[v] = O_v = (C_v \leftarrow \exists \bar{y}) \in \text{DO}\]

\[G \vdash_{H[\ell]} H[v] \in \text{lookup}(\Sigma[v])\]

By sub-derivation of \text{DF-Write}

By Hypothesis

By Lookup Lemma

By sub-derivation of \text{DF-Write}

By Hypothesis

By Lookup Lemma

\[\Sigma[v] = C_v \leftarrow \exists \bar{y} . d \quad H[v] = O_v = (C_v \leftarrow \exists \bar{y}) \in \text{DO}\]

\[\forall v'_j, d_j \in \exists \bar{y} . d \quad K[v'_j.d_j] = D_j \in \text{rng}(DD)\]

\[\Sigma[v] < : T'_i\]

\[H; K; L_t; L_E \vdash H[v] \in \text{irLookup}(T'_i)\]

By inversion of IR-Lookup

Take \(T_k = T'_i \in \overrightarrow{T}\), this proves (3).

**Subcase** \(e_0 = e'_0\). Then \(e = (e'_0, f_i = e_1)\)

From IRC-WRITE-RCV:

\[\begin{align*}
\emptyset \vdash e_0; S; H; K; L_t; L_E \rightsquigarrow_G e'_0; S'; H'; K'; L'_t; L'_E \\
\emptyset \vdash e'_0, f_i = e_1; S; H; K; L_t; L_E \rightsquigarrow_G e'_0, f_i = e_1; S'; H'; K'; L'_t; L'_E
\end{align*}\]

By induction hypothesis

By IRC-WRITE-RCV

**Subcase** \(e_0 = v, \text{ and } e_1 = e'_1\). Then \(e = (v, f_i = e'_1)\)

From IRC-WRITE-ARG:

\[\begin{align*}
\Gamma; \Upsilon; G \vdash_o e_1 \\
\emptyset \vdash e_1; S; H; K; L_t; L_E \rightsquigarrow_G e'_1; S'; H'; K'; L'_t; L'_E
\end{align*}\]

By sub-derivation of \text{DF-Write}

By induction hypothesis

By IRC-WRITE-ARG

\[\emptyset \vdash e_1; S; H; K; L_t; L_E \rightsquigarrow_G e'_1; S'; H'; K'; L'_t; L'_E\]

\[\emptyset \vdash v, f_i = e_1; S; H; K; L_t; L_E \rightsquigarrow_G v, f_i = e'_1; S'; H'; K'; L'_t; L'_E\]

Take \(e' = (v, f_i = e'_1)\).
Case DF-INVK: \( e = e_0.m(\overline{\tau}) \). There are three subcases to consider, depending on whether the receiver \( e_0 \), or the arguments \( \overline{\tau} \) are values.

**Subcase** \( e_0 = \ell \), and \( \overline{\tau} = \emptyset \) that is \( e = \ell.m(\emptyset) \)

From Ir-INVK:

\[
\begin{align*}
S[\ell] &= C<\overline{\tau}><(\tau) \quad \text{mbody}(m, C<\overline{\tau}>) = (\tau, e_R) \\
O &= H[\theta] \quad O_\ell = H[\ell] \\
\text{mtype}(m, C<\overline{\tau}>) &= T \rightarrow T_R \\
H; K; L_1; L_E \vdash O_\ell \in \text{irLookup}(T_R) \quad E' = \langle O, O, O_\ell, \text{Exp} \rangle \in \text{DE} \\
\forall k \in 1..|\overline{\tau}| \quad O_k = H[\overline{\tau}_k] \\
H; K; L_1; L_E \vdash O_k \in \text{irLookup}(T_k) \\
E_k = \langle O, O_k, O_\ell, \text{Exp} \rangle \in \text{DE} \\
E_k \in L_E[(\theta, \ell) \mapsto \{ E' \}]
\end{align*}
\]

\( \theta \vdash \ell.m(\overline{\tau}); S \quad H; K; L_1; L_E \rightsquigarrow_G \ell \triangleright [\overline{\tau}/\ell \text{this}]; S \quad H; K; L'_1; L'_E \)

To show:

1. \( O = H[\theta] \)
2. \( O_\ell = H[\ell] \)
3. \( \text{mtype}(m, C<\overline{\tau}>) = T \rightarrow T_R \quad \forall \ell_1, H; K; L_1; L_E \vdash O_\ell \in \text{irLookup}(T_R) \quad E' = \langle O, O, \ell_1, \text{Exp} \rangle \in \text{DE} \\
4. \forall k \in 1..|\overline{\tau}| \quad O_k = H[\overline{\tau}_k] \\
\quad H; K; L_1; L_E \vdash O_k \in \text{irLookup}(T_k) \\
E_k = \langle O, O_\ell, O_k, \text{Exp} \rangle \in \text{DE} \\
\)

By sub-derivation of DF-SIGMA

By sub-derivation of DF-SIGMA

Since \( \theta \in \text{dom}(S) \)

Since \( \ell \in \text{dom}(S) \)

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

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

and \( \forall d_i \in \text{dom}(C<\overline{\theta}_j.d_j>) \)

By DF-APPROX

By DF-APPROX

By DF-APPROX

By DF-APPROX
\[ G \vdash \emptyset, \emptyset, \ell.m(\overline{v}) \]
\[ \ell : [\Sigma[\emptyset] = C_{\ell} < \overline{v}.d > \]
\[ mtype(m, C_{\ell} < \overline{v}.d > ) = T' \rightarrow T'_R \]
\[ mtypeDecl(m, C_{\ell} ) = T'_R \rightarrow T_R \]
\[ G \vdash O \text{ import}(\Sigma[\ell], T_R) \]
\[ G \vdash O, O_i \in \text{lookup}(\Sigma[\ell]) \]
\[ G \vdash, O, O_j \in \text{lookup}(T_R) \]
\[ E' = \langle O_i, O, O_j, E \rangle \in DE \]
\[ \emptyset, \emptyset, G \vdash \ell \]
\[ G \vdash \emptyset, H[\ell] \in \text{lookup}(\Sigma[\ell]) \]
\[ T_R = \Sigma[\emptyset] = C_R < \overline{w}.d > \]
\[ T_R = C_R < \overline{v}.d > \]
\[ C_R < \overline{v}.d > = [\ldots \ell_j'' . d_j / p_j \ldots ]C_R < \overline{v}.d > \]
\[ \forall H; K; L_i; L_E \vdash O_i \in \text{irLookup}(T_R) \]
\[ G \vdash H[\ell] O_i \in \text{lookup}(T_R) \]
\[ O_i \in \text{rng}(H) \]
\[ O_i = \langle C' < \overline{v}.d > \rangle \]
\[ C' < : C_R \]
\[ \forall \ell'' . d_j \in \overline{w}.d \]
\[ D''_j = K[\ell'' . d_j] \]
\[ D''_j = D'_j \]

To show
\[ \forall p_j \in \overline{v}.d \]
\[ G \vdash H[\ell] D''_j \in \text{findD}(C_{\ell} : p_j) \]
\[ D''_j = D'_j = K[\ell'' . d_j] \]

\[ p_j \text{ is a domain parameter, local domain of } \ell \text{ or shared. Therefore we split the proof in cases} \]

**Case domain parameter:** \( p_j = \alpha_i \)
\[ \exists \ell'_i . d_i \in \overline{v}.d \text{ s.t } K[\ell'' . d_j] = K[\ell'_i . d_i] = D_{\ell_i} \]
\[ D_{\ell_i} \in \overline{D_{\ell}} \]
\[ H[\ell] = O_{\ell_i} = \langle C_{\ell} < \overline{D_{\ell}} > \rangle \]
\[ D'_j = D_{\ell_i} = DD[\langle H[\ell], C_{\ell} : \alpha_i \rangle] = DD[\langle H[\ell], \text{qual}(\ell'_i . d_i) \rangle] = K[\ell'_i . d_i] \]

By Params Lemma

**Case local domain:** \( p_k = \ell . d_j \)
\[ D_j = DD[\langle H[\ell], \alpha_i : \ell . d_j \rangle] \]

By sub-derivation of Df-New

**Case local domain:** \( p_k = \text{shared} \)
\[ D_{\text{v.k}} = D_{\text{shared}} = DD[\langle H[\ell], \alpha_i : \text{shared} \rangle] \]

We showed
\[ \forall p_j \in \overline{v}.d \]
\[ G \vdash H[\ell] D''_j \in \text{findD}(C_{\ell} : p_j) \]
\[ G \vdash H[\ell] O_i \in \text{lookup}(T_R) \]
\[ \forall O_i, E' = \langle H[\ell], H[\theta], O_i, I \rangle \in DE \]

By in inversion of Aux-Lookup

By above. Take \( O_i = H[\ell], O = H[\theta] \)
\[ \forall i \in L.\{\mathbb{P}\} \; v_k : \Sigma[v_k] \; \Sigma[v_k] < : T'_k \; G \vdash_\mathcal{O} \text{export}(\Sigma[l], \Sigma[v_k]) \]

By sub-derivation of Df-Invk

\[ \emptyset, \emptyset, G \vdash_\mathcal{O} \mathcal{T} \]

By sub-derivation of Df-Invk

\[ \emptyset, \emptyset, G \vdash_\mathcal{O} e \]

By sub-derivation of Df-Invk

\[ G \vdash_\mathcal{O} H[l] \in \text{lookup}(\Sigma[l]) \]

By Lookup Lemma

\[ \forall v_k \in \mathcal{P} \; \Sigma[v_k] = C_v < v.d > \]

By sub-derivation of Df-Invk

\[ \emptyset, \emptyset, G \vdash_\mathcal{O} v_k \]

By sub-derivation of Df-Invk

\[ G \vdash_\mathcal{O} H[v_k] \in \text{lookup}(\Sigma[v_k]) \]

By subderivation of Aux-Import

\[ E' = \langle H[l], H[\emptyset], H[v_k], \text{Exp} \rangle \in \text{DE} \]

By Df-Approx

\[ \forall v_j, d_j \in v.d \; K[v_j, d_j] = D_j \in \text{rng}(DD) \]

By inversion of IR-Lookup

\[ \Sigma[v_k] < : T'_k \]

This proves (4).

**Subcase** \( e_0 = e'_0 \) that is \( e = e'_0, m(\mathcal{T}) \).

From IRC-RecvInvk:

\[ \theta \vdash e'_0; S; H; K; L_i; L_E \leadsto_G e'_0; S'; H'; K'; L'_i; L'_E \]

\[ \theta \vdash e_0, m(\mathcal{T}); S; H; K; L_i; L_E \leadsto_G e'_0, m(\mathcal{T}); S'; H'; K'; L'_i; L'_E \]

\[ \theta \vdash e'_0; S; H; K; L_i; L_E \leadsto_G e'_0, m(\mathcal{T}); S'; H'; K'; L'_i; L'_E \]

By induction hypothesis

\[ \theta \vdash e'_0, m(\mathcal{T}); S; H; K; L_i; L_E \leadsto_G e'_0, m(\mathcal{T}); S'; H'; K'; L'_i; L'_E \]

By IRC-RecvInvk

Take \( e' = e_0, m(\mathcal{T}) \).

**Subcase** \( e_0 = v \) that is \( e = v.m(v_1, \ldots, v_n) \).

From IRC-ArgInvk:

\[ \theta \vdash e_i; S; H; K; L_i; L_E \leadsto_G e_i; S'; H'; K'; L'_i; L'_E \]

\[ \theta \vdash v.m(v_1, \ldots, v_n); S; H; K; L_i; L_E \leadsto_G v.m(v_1, \ldots, v_n); S'; H'; K'; L'_i; L'_E \]

By sub-derivation of Df-Invk

\[ \theta \vdash v.m(v_1, \ldots, v_n); S; H; K; L_i; L_E \leadsto_G v.m(v_1, \ldots, v_n); S'; H'; K'; L'_i; L'_E \]

By induction hypothesis

\[ \theta \vdash v.m(v_1, \ldots, v_n); S; H; K; L_i; L_E \leadsto_G v.m(v_1, \ldots, v_n); S'; H'; K'; L'_i; L'_E \]

By IRC-ArgInvk

Take \( e' = v.m(v_1, \ldots, v_n) \).

**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:

\[ \theta \vdash \ell \triangleright v; S; H; K; L_i; L_E \leadsto_G v; S; H; K; L_i; L_E \]

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:

\[ \ell \vdash e; S; H; K; L; L_E \leadsto_G \ell' ; S'; H'; K'; L'_L; L'_E \]

\[ \theta \vdash \ell \triangleright e; S; H; K; L; L_E \leadsto_G \ell \triangleright e' ; S'; H'; K'; L'_L; L'_E \]

\[ \ell \vdash e_0; S; H; K; L; L_E \leadsto_G e'_0; S'; H'; K'; L'_L; L'_E \]

\[ \theta \vdash \ell \triangleright e_0; S; H; K; L; L_E \leadsto_G \ell \triangleright e'_0; S'; H'; K'; L'_L; L'_E \]

Take \( e' = \ell \triangleright e'_0 \).

By induction hypothesis

By Irc-Context

\[ \blacksquare \]
\[ \theta \vdash e; S; H; K; L_I; L_E \overset{\ast G}{\sim} e; S; H; K; L_I; L_E \quad [\text{DF-REFLEX}] \]

\[ \theta \vdash e; S; H; K; L_I; L_E \overset{\ast G}{\sim} e''; S''; H''; K''; L''_I; L''_E \]
\[ \theta \vdash e''; S''; H''; K''; L''_I; L''_E \overset{\ast G}{\sim} e'; S'; H'; K'; L'_I; L'_E \]
\[ \theta \vdash e; S; H; K; L_I; L_E \overset{\ast G}{\sim} e'; S'; H'; K'; L'_I; L'_E \quad [\text{DF-TRANS}] \]

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

### 3.6 Theorem: Object Graph Soundness

If

\( G = (DO, DD, DE) \)
\n\( DO, DD, DE \vdash (CT, e_{\text{root}}) \)
\n\( \forall e, \theta_0 \vdash e; \emptyset; \emptyset; \emptyset; \emptyset; \emptyset \overset{\ast G}{\sim} e; S; H; K; L_I; L_E \)
\n\( \Sigma \vdash S \)

then

\( DO, DD, DE \vdash_{CT, H} \Sigma \)
\n\( (S, H, K, L_I, L_E) \sim (DO, DD, DE) \)

where \( \overset{\ast G}{\sim} \) relation is the reflexive and transitive closure of \( \sim_G \) relation (Fig. 11). \( \theta_0 \) is the location of the first object instantiated by \( e_{\text{root}} \).

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 \( \overset{\ast G}{\sim} \) 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:

\( \theta_0 \vdash e; \emptyset; \emptyset; \emptyset; \emptyset; \emptyset \overset{\ast G}{\sim} e; S; H; K; L_I; L_E \)

Since \( S = \emptyset \):

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

By inversion of DF-TRANS:

\( \theta_0 \vdash e; \emptyset; \emptyset; \emptyset; \emptyset \overset{\ast G}{\sim} e'; S'; H'; K'; L'_I; L'_E \)

By induction hypothesis:

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

By inversion of DF-TRANS:

\( \theta_0 \vdash e'; S'; H'; K'; L'_I; L'_E \overset{\ast G}{\sim} e; S; H; K; L_I; L_E \)

By preservation:

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

By assumption:

¹The soundness proof follows similar steps to the one of points-to analysis [1].
\[ \theta_0 \vdash e; \emptyset; \emptyset; \emptyset; \emptyset \sim^* e; S; H; K; L_i; L_E \]

Since \( S = \emptyset \):
\( (\emptyset, \emptyset, \emptyset, \emptyset) \sim G \)

By inversion of \( \text{DF-TRANS} \):
\[ \theta_0 \vdash e; \emptyset; \emptyset; \emptyset; \emptyset \sim^* e'; S'; H'; K'; L_i'; L_E' \]

By induction hypothesis:
\[ DO, DD, DE \vdash_{CT,H} \Sigma' \]

By inversion of \( \text{DF-TRANS} \):
\[ \theta_0 \vdash e'; S'; H'; K'; L_i'; L_E' \sim G e; S; H; K; L_i; L_E \]

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
\[ \Gamma \cup \{ \tau : T \}, \Sigma, \theta \vdash e : T \]
\[ \Gamma \cup \{ \tau : T' \}, \Upsilon, G \vdash e \]
\[ \Gamma, \Sigma, \theta \vdash \tau : T_a \text{ where } T_a <: [\tau / \tau]T \]
then
\[ \Gamma, \Sigma, \theta \vdash [\tau / \tau]e : T' \text{ for some } T' <: [\tau / \tau]T \]
\[ \Gamma, \Upsilon, G \vdash_O [\tau / \tau]e \]

**Proof:** By induction on the \( \Gamma, \Upsilon, G \vdash_O e \) relation.

**Df-Weakening Lemma.**

If
\[ \Gamma, \Upsilon, G \vdash_O e \]
then
\[ \Gamma, \Upsilon \cup \{ C\langle D \rangle \}, G \vdash_O e \]

**Proof:** By induction on the \( \Gamma, \Upsilon, G \vdash_O e \) relation.

**Df-Strengthening Lemma.**

If
\[ \Gamma, \emptyset, G \vdash_O \text{new } C\langle D \rangle(v) \]
\[ \forall i \in 1.\lfloor \overline{P} \rfloor \quad D_i = DD[(O, p_i)] \]
\[ \Gamma, \Upsilon \cup \{ C\langle D \rangle \}, G \vdash_O e' \]
then
\[ \Gamma, \Upsilon, G \vdash_O e \]

**Proof:** By induction on the \( \Gamma, \Upsilon, G \vdash_O e \) relation.

**Df-Domains Lemma.**

If
\[ \emptyset, \Sigma, \theta \vdash e : T \]
\[ \Sigma \vdash S \]
\[ G \vdash_{CT,H} \Sigma \]
\[ \emptyset, \emptyset, G \vdash_O \text{new } C\langle P \rangle(v) \]
\[ (S, H, K, L_i, L_E) \sim (DO, DD, DE) \]
\[ G \vdash_O \text{domains}(C, O_C) \]
Proof: By generalized induction.

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
\( G \vdash_{H[\theta]} H[\ell] \in \text{lookup}(\Sigma[\ell]) \)

Proof:
\[ \Sigma[\ell] = C_{\ell} < D > \]
\[ H[\ell] = (C_{\ell} < D >) \]

To Show:
Take \( H[\theta] = C_{\theta} < D_{\theta} > \quad \theta \in \text{domain}(H) \)
\( \forall \ell', d_j \in l. d_i, G \vdash_{H[\theta]} D'_j \in \text{findD}(C_{\theta} :: \text{qual}(\ell', d_j)) \quad D' = D_j \quad D_j \in D \quad K[\ell'_j, d_j] = D_j \)

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 \( \ell \) in the presence of recursive types.

Case \( \ell'_j = \theta \). Local domains.
\( H[\theta] = \langle C_{\theta} < D_{\theta} > \rangle \in DO \)
\( \forall d_j \in \text{domains}(\Sigma[\theta]) \quad K[\theta, d_j] = D_j = \langle D_{d_j}, C_{\theta} :: d_j \rangle \quad \{(H[\theta], C_{\theta} :: d_j) \mapsto D_j\} \in DD \)
By Df-Approx
\( D'_j = DD[(H[\theta], C_{\theta} :: d_j)] = K[\theta, d_j] \)
By above
\( G \vdash_{H[\theta]} D'_j \in \text{findD}(C_{\theta} :: \text{this}.d_j) \)
By inversion of Aux-FindThis
\( G \vdash_{H[\theta]} D'_j \in \text{findD}(C_{\theta} :: \text{qual}(\ell'_j, d_j)) \)
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.
\( H[\ell] = O_C = \langle C_{\ell} < D > \rangle \in DO \)
and \( \forall \ell', d_j \in l. d_i \quad K[\ell'_j, d_j] = D_j = \langle D_{d_j}, \text{qual}(\ell'_j, d_j) \rangle \in \text{rng}(DD) \)
and \( \forall d_i \in \text{domains}(C_{\ell} < D >) \)
\( K[\ell, d_i] = D_i = \langle D_{d_i}, C :: d_i \rangle \quad \{(O_C, C :: d_i) \mapsto D_i\} \in DD \)
\( DD[(H[\ell], \ell, d_j)] = D_j = K[\ell, d_j] \)
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 \( \theta \).
Assume that \( \forall \ell' \in \text{domain}(H), G \vdash_{H[\theta]} H[\ell'] \in \text{lookup}(\Sigma[\ell']) \) but \( \ell \notin \text{dom}(H) \). Where \text{domain} means the set of all keys stored in \( H \).

49
To show:
\[ K[n.d_j] = D_j \quad G \vdash_{H[n]} D_j \in \text{findD}(C\theta::\text{qual}(n.d_j)) \]

\[ n : \Sigma[n] \]
\[ DD[(H[n], n.d_j)] = D_j = K[n.d_j] \quad \text{By Df-Approx, since } d_j \in \text{domains}(\Sigma[n]) \]
\[ G \vdash_{H[n]} H[n] \in \text{lookup}(\Sigma[n]) \quad \text{By induction hypothesis} \]
\[ G \vdash_{H[n]} D_j \in \text{findD}(C\theta::\text{qual}(n.d_j)) \quad \text{By inversion of } \text{Aux-Find-Public} \]

\[ G \vdash_{H[n]} H[\ell] \in \text{lookup}(\Sigma[\ell]) \quad \text{By induction} \]

\[ \hfill \blacklozenge \]

**Params Lemma.**

If \( \emptyset, \Sigma, \theta \vdash \ell : \Sigma[\ell] \)
\[ \Sigma \vdash S \]
\[ \emptyset, \emptyset, G \vdash_{O} \text{new } C\langle \ell.d \rangle(\forall) \]
\[ G \vdash_{CT,H} \Sigma \]
\[ (S, H, K, L_1, L_E) \sim (DO, DD, DE) \]
\[ \ell \in \text{domain}(S) \quad H[\ell] = C\langle D \rangle \quad \Sigma[\ell] = C\langle \ell.d \rangle \quad S[\ell] = C\langle \ell.d \rangle(\forall) \]
\[ CT(C) = \text{class } C\langle \forall \rangle \quad \{ F, J; \text{dom}; \ldots; \text{md}; \} \]

then \( \forall \ell', d_i \in \ell.d \quad DD[(H[\ell], \text{qual}(\ell',d_i))] = DD[(H[\ell], C::\alpha_i)] = D_i = K[\ell',d_i] \)

**Proof:**

By induction on the \( G \vdash_{O} dparams(C,O_C) \) relation using subderivation of Df-New IR-New.

\[ \hfill \blacklozenge \]
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. $\Gamma$ maps variables to their types a store $S$ maps locations $\ell$ to their contents; the set of variables includes the distinguished variable this of type $T_{\text{this}}$ used to refer to the receiver of a method; the result of the computation is a location $\ell$, which is sometimes referred to as a value $v$; $S[\ell]$ denotes the store entry of $\ell$; $S[\ell, i]$ denotes the value of $i^{\text{th}}$ field of $S[\ell]$; $S[\ell \mapsto C<\ell.d>(\overline{v})]$ denotes adding an entry for location $\ell$ to $S$; $\alpha$ and $\beta$ range over formal domain parameters; the expression form $\ell \triangleright e$ represents a method body $e$ executing with a receiver $\ell$; a program is a tuple $(CT, e_{\text{root}})$ 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 $\alpha$, 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 ($\text{OGraph}$) with nodes that represent abstract objects ($\text{OOObjects}$), groups of objects ($\text{ODomains}$), and $\text{OEdges}$ that represent dataflow communication between
CT ::= cdef table of class declarations
cdef ::= class C<α, β> extends C'<μ>'
        { dom; T x f; C(T_A f; T_A f) }  class decl.
dom ::= [public] domain d;
md ::= T_A ret m(T_B f) T_this { ret = e_R; return ret; }  method decl.
e ::= x = new C<A, p>(y)
    | ℓ | ℓ > e  expressions
    | x = y, f | x.f = y | x = y | x = r.m(μ)
n ::= x | v  values or variable names
p ::= α | n.d | SHARED  domain name
A ::= unique | p  domain may be unique
B ::= lent | A  domain may be lent or unique
T ::= C<μ>  precise type
T_A ::= C<A, μ>  owner domain may be unique
T_B ::= C<B, μ>  owner domain may be lent or unique
v, ℓ, θ ∈ locations
x, y, r, a ∈ variables
S ::= ℓ → C<ℓ.d>(μ)  location store
Σ ::= ℓ → T  store typing
Γ ::= x → T_B  type environment

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

G ∈ OGraph ::= { Objects = DO, DomainMap = DD, Edges = DE }
D ∈ ODomain ::= { Id = D.id, Domain = C::d }  Domain map
O ∈ OObject ::= { Type = C<μ> }  Dataflow Object
E ∈ OEdge ::= { From = Osrc, To = Odst, Label = O_label, Flag = Imp | Exp }  Dataflow Edge
DD ::= ∅ | DD ∪ { (O, C::d) → D }  Domain map
DO ::= ∅ | DO ∪ { O }  Dataflow Object
DE ::= ∅ | DE ∪ { E }  Dataflow Edge
Υ ::= ∅ | Υ ∪ { C<μ> }  Stack of visited OObjects
FG ::= ∅ | FG ∪ { (Osrc, x, Bsrc) annot (Odst, y, Bdst) }
annot ::= (i | ) | • | •  value flow annotations

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

abstract objects (Fig. 13). Each OEdge is a directed edge from a source Osrc to a destination Odst. 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 $(O, x, B)$ means a list of $j$ triplets $(O, x_1, B_1), (O, x_2, B_2), \ldots, (O, x_j, B_j)$ An edge in $FG$ has an annot label to track if the value flow is due to a method invocation ($\xrightarrow{\ell}$), a method return ($\xrightarrow{\ell}$). The label $\bullet$ denotes an empty annotation on a value flow edge due an assignment ($\xrightarrow{\neg}$). 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 $D_{shared}$, which constitutes the root of the OGraph. It uses the following initial values:

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

$$FG_0 = \emptyset, \ DO_0 = \{ O_{world} \}$$

$$DD_0 = \{(O_{world}, ::\text{shared}) \mapsto D_{shared} \}, DE_0 = \emptyset$$

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 $\Gamma$, a stack $\Upsilon$ 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 $OC$ 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 $\alpha_i$ maps to the same $D_i$ but in the
\[CT(C) = \text{class } C < \overline{\pi}, \overline{T}> \text{ extends } C' < \overline{\pi}> \{ \overline{T} T; \ \overline{dom}; \ldots; \ \overline{md}; \} \quad G = \{DO, DD, DE\} \]
\[O = C_{\text{this}} < \overline{D}> \quad \forall i \in 1..n \quad FG, G \vdash O D_i \in \text{findD}(C_{\text{this}}; p_i) \]
\[O_C = (C < \overline{D}>) \quad \{OC \} \subseteq DO \]
\[dparams(C, OC) \quad \{(OC, qual(p_i)) \mapsto D_i \} \subseteq DD \quad G \vdash_0 \text{ddomains}(C, OC) \]
\[\Gamma[\overline{\pi}] = T_o \quad \{(O, \overline{p_1}) \mapsto (OC, \text{this}_o, \alpha_0), (O, \overline{\pi}, \text{owner}(T_o)) \mapsto (OC, \text{this}_f, \text{owner}(T)) \} \subseteq FG \]

\[\forall m \in \text{md}. \ mbody(m, C < \overline{\pi}>) = (\overline{\tau}; T_e \ \epsilon_R) \]
\[C < \overline{D}> \notin \Upsilon \implies \{\overline{\tau}; T: \ \text{this}_C < \overline{\pi}>, \Upsilon \cup \{C < \overline{D}>, FG, G \vdash_0 \epsilon_R \}
\]

\[\text{[DF-New]} \]

\[\Gamma[\overline{\pi}] = T_o \quad \{(O, \overline{p_1}) \mapsto (OC, \text{this}_o, \alpha_0), (O, \overline{\pi}, \text{owner}(T_o)) \mapsto (OC, \text{this}_f, \text{owner}(T)) \} \subseteq FG \]

\[\forall m \in \text{md}. \ mbody(m, C < \overline{\pi}>) = (\overline{\tau}; T_e \ \epsilon_R) \]
\[C < \overline{D}> \notin \Upsilon \implies \{\overline{\tau}; T: \ \text{this}_C < \overline{\pi}>, \Upsilon \cup \{C < \overline{D}>, FG, G \vdash_0 \epsilon_R \}
\]

\[\text{[DF-New-Unique]} \]

\[\Gamma[r] = T_r = C < \overline{\pi}> \quad (T_k f_k) \in \text{fields}(T_r) \quad FG, G \vdash_0 \text{import}(T_r, T_k) \]
\[FG, G \vdash_0 \text{lookup}(T_r) \quad (T_k f_k) \in T_r \]

\[\{(O, \text{owner}(T_r)) \mapsto (O, \text{this}_o, \alpha_0), (O, \text{fk}, \text{owner}(T_k)) \mapsto (O, \text{fk}, \text{owner}(T_k)) \} \subseteq FG \]

\[\Gamma[r] = T_r = C < \overline{\pi}> \quad (T_k f_k) \in \text{fields}(T_r) \quad \Gamma[r] = T_r \quad T_k < T_r \quad FG, G \vdash_0 \text{export}(T_k, T_r) \]

\[FG, G \vdash_0 \text{lookup}(C < \overline{\pi}>) \quad (T_k f_k) \in T_r \quad \{(O, \text{owner}(T_r)) \mapsto (O, \text{fk}, \text{owner}(T_k)) \} \subseteq FG \]

\[\text{[DF-Read]} \]

\[\Gamma[r] = T_r = C < \overline{\pi}> \quad (T_k f_k) \in \text{fields}(T_r) \quad \Gamma[r] = T_r \quad T_k < T_r \quad FG, G \vdash_0 \text{export}(T_k, T_r) \]

\[FG, G \vdash_0 \text{lookup}(C < \overline{\pi}>) \quad (T_k f_k) \in T_r \quad \{(O, \text{owner}(T_r)) \mapsto (O, \text{fk}, \text{owner}(T_k)) \} \subseteq FG \]

\[\text{[DF-White]} \]

\[CT(C) = \text{class } C < \overline{\pi}, \overline{T}> \text{ extends } C' < \overline{\pi}> \{ \overline{T} T; \ \overline{dom}; \ldots; \ \overline{md}; \} \quad G = \{DO, DD, DE\} \]
\[\Gamma[\overline{\pi}] = T_o \quad \text{mtype}(m, C < \overline{\pi}>) = \overline{T} \rightarrow T_R \quad FG, G \vdash_0 \text{import}(C < \overline{\pi}>, T_R) \]
\[FG, G \vdash_0 \text{lookup}(C < \overline{\pi}>) \quad T_R \vdash m(T_R \ \overline{\pi}) \quad T_bis \{\text{ret} = \epsilon_R; \text{return}\ \text{ret}; \} \in \text{md} \quad i = \text{fresh}_i(O, x_0 = r_0.m(\overline{\pi})) \]
\[\Gamma[r] = T_o \quad \{(O, r.o, p_0) \mapsto (O, \text{this}_o, \alpha_0, (O, \overline{\pi}, \text{owner}(T_o))) \mapsto (O, \overline{\pi}, \text{owner}(T_o)) \} \subseteq FG \]
\[\Gamma[\text{ret}] = T_R \quad \{(O, \text{ret}, \text{owner}(T_R)) \mapsto (O, x_0, \text{owner}(T_R)) \} \subseteq FG \]

\[\text{[DF-InvK]} \]

\[\Gamma[r] = T_r \quad \Gamma[\overline{\pi}] = T_o \quad \{(O, r, \text{owner}(T_r)) \mapsto (O, x, \text{owner}(T_r)) \} \subseteq FG \]

\[\Gamma[r] = T_r \quad \Gamma[\overline{\pi}] = T_o \quad \{(O, r, \text{owner}(T_r)) \mapsto (O, x, \text{owner}(T_r)) \} \subseteq FG \]

\[\text{[DF-Assign]} \]

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

The context of \(O_C\) (Fig. 14). **DF-New** also ensures that the object hierarchy is created such that new **O**Domains 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 \( \pi \) to the corresponding fields \texttt{this.f} (Fig. 14). Next, \textit{Df-New-Unique} handles the case when the owner is \texttt{unique}. The rule is similar to \textit{Df-New}, except that it uses the auxiliary judgment \textit{solveUnique} to find the actual owner domain of \( O_C \). We describe \textit{solveUnique} later in Fig. 15.

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

Next, \textit{Df-Read}, \textit{Df-Write}, \textit{Df-Invk}, and \textit{Df-Assign} ensure value flow edges are created in \( FG \). For example, \textit{Df-Read}, ensures that the flow graph contains one edge from the receiver \( r \) in the context \( O \) to the context variable \texttt{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 \textit{Df-Invk} adds annotations to the value flow edges that correspond to the arguments of the invocation and to the return value. For \textit{Df-Read}, \textit{Df-Write}, and \textit{Df-Invk}, the context \texttt{OOObject} of the source is different from the context \texttt{OOObject} of the destination of a flow edge. On the other hand, for \textit{Df-Assign}, the context \texttt{OOObject} remains unchanged for the source and destination.

The precision of the analysis is provided by the auxiliary judgments \textit{Df-Lookup} (Fig. 15). For a given type \( C'<\pi> \) that includes a list of actual domain parameters, \textit{lookup} returns those \texttt{OOObjects} \( O_k \) 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 \((O,C_{\texttt{this}::p_i})\) in \( DD \). The second condition increases the precision of the analysis, because \textit{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 \textit{Df-Lookup-Lent}, and \textit{Df-Lookup-Unique} for \textit{lookup} where the owner domain is \texttt{lent} or \texttt{unique}. These rules use \textit{solveLent} and \textit{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 \( D'_i \) corresponding to \texttt{lent} or \texttt{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 \texttt{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 \texttt{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 \texttt{ODomain} for each flow object. Multiple dataflow edges can then refer to the same flow object.
G = \langle DO, DD, DE \rangle \quad O = C_{\text{this}}<\overrightarrow{T}> \quad O_k \in DO \quad O_k = \langle C<\overrightarrow{T}> \rangle \quad C <: C' \\
\forall i \in 1..[p] \quad FG, G \vdash_0 D'_i \in \text{findD}(C_{\text{this}};p_i) \quad D'_i = D_i \quad \text{[DF-LOOKUP]}

G = \langle DO, DD, DE \rangle \quad O = C_{\text{this}}<\overrightarrow{T}> \quad O_k \in DO \quad O_k = \langle C<D_1,\overrightarrow{T}> \rangle \quad C <: C' \\
\forall i \in 2..[p] \quad FG, G \vdash_0 D'_i \in \text{findD}(C_{\text{this}};p_i) \quad D'_i = D_i \quad \text{[DF-LOOKUP-LENT]}

A = \text{unique} \implies FG \vdash (O', x, \text{unique}) \in \text{findSrcUnique}(O', C') \land D'_i = DD[(O'', C',\text{unique})] \land D'_i = D_i \\
A = p' \implies FG, G \vdash_0 D'_i \in \text{findD}(C_{\text{this}};p') \land D'_i = D_i \quad \text{[DF-LOOKUP-UNIQUE]}

FG_P = \text{propagateAll}(FG) \\
(\alpha, s, \text{unique}) \sim (O', y, \text{unique}) \in FG_P \quad s : C \quad C <: C' \quad \exists (O'', x, \text{unique}) \sim (\alpha, s, \text{unique}) \in FG_P \quad \text{[Aux-Find-Unique]}

FG_P = \text{propagateAll}(FG) \\
(O', x, \text{unique}) \sim (O', y, A) \in FG_P \quad y : C \quad C <: C' \quad \text{[Aux-Resolve-Unique]}

FG_P = \text{propagateAll}(FG) \\
(O', x, y) \sim (O', y, A) \in FG_P \quad x : C \quad C <: C' \quad \text{[Aux-Resolve-Lent]}

G = \langle DO, DD, DE \rangle \quad FG \vdash (O', y, A) \in \text{solveUnique}(O, C) \quad \text{[Aux-UniqueDom]}

A = \text{unique} \implies (D = \langle D_{\text{this}}, C,:\text{unique} \rangle \land \{O, C,:\text{unique}\} \implies D) \subseteq DD \\
A = p' \implies (O' = \langle C_{\text{this}}<\overrightarrow{T}>, D) \land FG, G \vdash_0 D \in \text{findD}(C_{\text{this}};p') \\
FG, G \vdash_0 D \in \text{uniqueDomains}(C)

\text{Figure 15: Rules for resolving } \text{lent}, \text{ and } \text{unique. Auxiliary judgments import and export ensure dataflow edges are created.}

O = \langle C<\overrightarrow{T}> \rangle \quad n : C_n<\overrightarrow{T}> \quad FG, G \vdash_0 O_i \in \text{lookup}(C_n<\overrightarrow{T}>) \quad D_i = DD[(O_i, C_n,:d)] \quad \text{[DF-FindD-Public]}

\text{FG, G} \vdash_0 D_i \in \text{findD}(C_n,:d) \quad \text{[DF-FindD-This]}

O = \langle C<\overrightarrow{T}> \rangle \quad D_i = DD[(O, C,:d_i)] \quad \text{[DF-FindD-Shared]}

\text{FG, G} \vdash_0 D_i \in \text{findD}(C,:) \quad \text{[DF-FIndD]}
function summarize(FG)

\( FG* = FG \)

\( WL = \{ (O_1, x_1, B_1), (O_2, x_2, B_2) \in FG \text{ s.t. } a \text{ is } (i) \} \)

while \( WL \neq \emptyset \) do

remove \( e_1 : (O_1, x_1, B_1) \xrightarrow{sl} (O_2, x_2, B_2) \) from \( WL \)

if \( a_1 \) is \( (i) \) then

for \( e_2 : (O_2, x_2, B_2) \xrightarrow{sl} (O_3, x_3, B_3) \in FG* \) do

if \( e_3 = \text{concat}(e_1, e_2) \notin FG* \) then

add \( e_3 \) to \( FG* \) and \( WL \)

else

if \( a_1 \) is \( * \) or \( \ast \) then

for \( e'_2 : (O_0, x_0, B_0) \xrightarrow{sl} (O_1, x_1, B_1) \in FG* \) do

if \( e'_3 = \text{concat}(e'_2, e_1) \notin FG* \) then

add \( e'_3 \) to \( FG* \) and \( WL \)

return \( FG* \)

\( \text{concat}((O_1, x, B_1) \xrightarrow{sl} (O_0, y, B_2), (O_2, y, B_2) \xrightarrow{sl} (O_2, z, B_3)) = \)

\( = (O_1, x, B_1) \xrightarrow{sl} (O_2, z, B_3) \)

\( \text{concat}((O_1, x, B_1) \xrightarrow{sl} (O_2, y, B_2), (O_2, y, B_2) \xrightarrow{sl} (O_1, z, B_3)) = \)

\( = (O_1, x, B_1) \xrightarrow{sl} (O_1, z, B_3) \)

\( \text{concat}((O_1, x, B_1) \xrightarrow{sl} (O_2, y, B_2), (O_2, y, B_2) \xrightarrow{sl} (O_3, z, B_3)) = \)

\( = (O_1, x, B_1) \xrightarrow{sl} (O_3, z, B_3) \)

Figure 17: The algorithm \textit{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 \textit{solveLent} and \textit{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 \textit{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 \textit{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 \texttt{lent} and \texttt{unique}. For each example, Figure 18 also shows the extracted OGraph that has OObjects such as \( \langle \texttt{A<DATA,DOM1>}, \texttt{f,F} \rangle \) and \( \langle \texttt{A<DATA,DOM2>}, \texttt{f,F} \rangle \). The flow graph \( FG \) (not shown) has nodes such as \( \langle \texttt{A<DATA,DOM1>}, \texttt{f,F} \rangle \) and \( \langle \texttt{A<DATA,DOM2>}, \texttt{f,F} \rangle \) and the following flow edges:
The algorithm `summarize` (Fig. 17) computes \( F^* \) 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 \((i, \_i)\) 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.

\[
\langle \text{Main<SHARED>}, \text{n1,DOM1} \rangle \overset{11}{\rightarrow} \langle \text{A<DATA,DOM1>}, \text{f,F} \rangle \\
\langle \text{Main<SHARED>}, \text{n2,DOM2} \rangle \overset{14}{\rightarrow} \langle \text{A<DATA,DOM2>), \text{num,F} \rangle \\
\langle \text{A<DATA,DOM2>}, \text{num,F} \rangle \overset{\star}{\rightarrow} \langle \text{A<DATA,DOM2>}, \text{f,F} \rangle \\
\langle \text{A<DATA,DOM2>}, \text{f,F} \rangle \rightsquigarrow \langle \text{A<DATA,DOM2>}, \text{ret,F} \rangle \\
\langle \text{A<DATA,DOM2>}, \text{ret,F} \rangle \overset{15}{\rightarrow} \langle \text{Main<SHARED>}, \text{dest,lent} \rangle
\]

The flow graph \( F^* \) has a transitive flow:

\[
\langle \text{A<DATA,DOM2>}, \text{f,F} \rangle \rightsquigarrow \ldots \rightsquigarrow \langle \text{Main<SHARED>}, \text{dest,lent} \rangle
\]

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

\[
\langle \text{A<DATA,DOM1>}, \text{f,F} \rangle \rightsquigarrow \ldots \rightsquigarrow \langle \text{Main<SHARED>}, \text{dest,lent} \rangle
\]

According to \textit{Aux-Resolve-Lent}, the Flow Graph Analysis resolves \textit{lent} to \textit{F} in the context of \(\langle \text{A<DATA,DOM2>} \rangle\). Then, according to \textit{Aux-Lookup-Lent}, \textit{lookup} returns only \(\langle \text{Integer<DOM2>} \rangle\) and not \(\langle \text{Integer<DOM1>} \rangle\), which would introduce a false positive dataflow edge in the \textit{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(f))` 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:
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\langle FG, s \rangle \\
\text{FG}_p = \emptyset \\
\text{WL} = \emptyset \\
\text{for } e : \langle O_1, s, B_1 \rangle \sqsubseteq \langle O_2, x_2, B_2 \rangle \in \text{FG}_s \text{ do} \\
\text{if } \{ B_1, B_2 \} \cap \{ \text{lent}, \text{unique}, n, d \} \neq \emptyset \text{ then} \\
\text{if } a_1 \text{ is } ( \text{then} \\
\text{add } \langle O_1, s, B_1 \rangle \overset{\text{call}}{\rightarrow} \langle O_2, x_2, B_2 \rangle \text{ to } \text{FG}_p \text{ and } \text{WL} \\
\text{else} \\
\text{add } \langle O_1, s, B_1 \rangle \overset{\text{nCall}}{\rightarrow} \langle O_2, x_2, B_2 \rangle \text{ to } \text{FG}_p \text{ and } \text{WL} \\
\text{while } \text{WL} \neq \emptyset \text{ do} \\
\text{remove } e_1 : \langle O_1, x_1, B_1 \rangle \overset{\text{call}}{\rightarrow} \langle O_2, x_2, B_2 \rangle \text{ from } \text{WL} \\
\text{for } e_2 : \langle O_2, x_2, B_2 \rangle \overset{\text{call}}{\rightarrow} \langle O_3, x_3, B_3 \rangle \in \text{FG}_s \text{ do} \\
\text{if } e_3 = \text{concat}'(e_1, e_2) \notin \text{FG}_p \text{ then} \\
\text{add } e_3 \text{ to } \text{FG}_s \text{ and } \text{WL} \\
\text{return } \text{FG}_p \\
\text{function propagateAll}\langle \text{FG} \rangle \\
\text{FG}_s = \text{summarize}\langle \text{FG} \rangle; \\
\text{for } (O, x, B) \in \text{FG}_s \text{ do} \\
\text{FG}_p = \text{FG}_p \cup \text{propagate}\langle \text{FG}_s, x \rangle \\
\text{return } \text{FG}_p \\
\text{concat}'(\langle O_1, x, B_1 \rangle \overset{\text{call}}{\rightarrow} \langle O_2, y, B_2 \rangle, \langle O_2, y, B_2 \rangle \overset{\text{call}}{\rightarrow} \langle O_3, z, B_3 \rangle) = \\
\langle O_1, x, B_1 \rangle \overset{\text{call}}{\rightarrow} \langle O_2, z, B_3 \rangle \\
\text{concat}'(\langle O_1, x, B_1 \rangle \overset{\text{call}}{\rightarrow} \langle O_2, y, B_2 \rangle, \langle O_2, y, B_2 \rangle \overset{\text{call}}{\rightarrow} \langle O_3, z, B_3 \rangle) = \\
\langle O_1, x, B_1 \rangle \overset{\text{call}}{\rightarrow} \langle O_3, z, B_3 \rangle \\
\text{concat}'(\langle O_1, x, B_1 \rangle \overset{\text{call}}{\rightarrow} \langle O_2, y, B_2 \rangle, \langle O_2, y, B_2 \rangle \overset{\text{call}}{\rightarrow} \langle O_3, z, B_3 \rangle) = \\
\langle O_1, x, B_1 \rangle \overset{\text{call}}{\rightarrow} \langle O_3, z, B_3 \rangle \\
\text{Figure 19: } \text{The algorithm } \text{propagate} \text{ adds more edges for nodes with variables declared } \text{lent} \text{ or } \text{unique.} \\
\begin{align*} 
\langle \text{Main}<\text{SHARED}>, \text{n2}, \text{unique} \rangle \overset{\text{call}}{\rightarrow} & \langle \langle \text{A}<\text{DATA}, \text{DOM2}>, \text{num}, \text{F} \rangle \rangle \\
\langle \text{Main}<\text{SHARED}>, \text{n2}, \text{unique} \rangle \overset{\text{call}}{\rightarrow} & \langle \langle \text{A}<\text{DATA}, \text{DOM2}>, \text{f}, \text{F} \rangle \rangle \\
\langle \text{Main}<\text{SHARED}>, \text{n2}, \text{unique} \rangle \overset{\text{call}}{\rightarrow} & \langle \langle \text{A}<\text{DATA}, \text{DOM2}>, \text{ret}, \text{F} \rangle \rangle \\
\langle \text{Main}<\text{SHARED}>, \text{n2}, \text{unique} \rangle \overset{\text{call}}{\rightarrow} & \langle \langle \text{Main}<\text{SHARED}>, \text{dest}, \text{lent} \rangle \rangle 
\end{align*} \\
\text{Then, } \text{Df-New-Uniqwe} \text{ invokes } \text{findD}(<\text{Main}:\text{unique}) \text{ that in turn invokes } \text{solveUnique}(<\text{Main}<\text{SHARED}>, \text{Integer} >) \text{. Here, } \text{unique} \text{ is resolved to the domain parameter } \text{F} \text{ that is} \\
\text{bound to } \text{DOM2} \text{ in the context of } <\text{B}<\text{DATA}, \text{DOM2}>, \text{. Therefore, } \text{Df-New-Uniqwe} \text{ creates the } \text{OObject} \\
<\text{Integer}<\text{DOM2}>, \text{>.} 

63
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 \texttt{len}t 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


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.
\[ CT(C) = \text{class } C<\pi, \beta> \text{ extends } C'<\pi> \text{ assumes } \gamma \rightarrow \delta \{ \bar{D}; \; \bar{F}; \; \bar{K}; \; \bar{M}; \} \]

\[ \quad \text{(public domain } d) \in \bar{D} \quad \text{Aux-Public} \]

\[ \bar{D} = \text{public\_opt\_domain } d_C \quad \text{domains}(C' < \pi>) = \bar{d'} \quad \text{Aux-Domains} \]

\[ \quad \text{domains}(C<\pi, \beta>) = \text{this.d}_C, \bar{d'} \]

\[ \quad \text{domains}(\text{Object}<\alpha_0>) = \emptyset \quad \text{Aux-Domains-Obj} \]

\[ \quad \text{class } C<\pi> \quad \text{Aux-Params} \]

\[ \quad \text{params}(C) = \pi \]

\[ \bar{F} = \bar{T} \bar{f} \quad \text{fields}(C' < \pi>) = \bar{T'} \bar{f}' \quad \text{Aux-Fields} \]

\[ \quad \text{fields}(C<\pi, \beta>) = (\bar{T}/\pi, \bar{f}/\beta, \bar{T'}/\bar{f}', \bar{T'/f}) \quad \text{Aux-Fields-Obj} \]

\[ \quad \text{owner}(C<\pi>) = p_1 \quad \text{Aux-Owner} \]

\[ \quad \text{m is not defined in } \bar{M} \]

\[ \text{mtype}(m, C<\pi>) = [\pi/\pi] \bar{T} \rightarrow \bar{T}_R \quad \text{Aux-MType1} \]

\[ \text{m is not defined in } \bar{M} \]

\[ \text{mtype}(m, C<\pi, \beta>) = \text{mtype}(m, C' < \pi>) \quad \text{Aux-MType2} \]

\[ \quad (T_R \; m(\bar{T} \; \bar{x}) \{ \text{return } e; \}) \in \bar{M} \]

\[ \quad \text{mbody}(m, C<\pi>) = [\pi/\pi] (\bar{x}, \; e) \quad \text{Aux-MBody1} \]

\[ \quad \text{m is not defined in } \bar{M} \]

\[ \quad \text{mbody}(m, C<\pi, \beta>) = \text{mbody}(m, C' < \pi>) \quad \text{Aux-MBody2} \]

\[ \quad \text{mtype}(m, C<\pi>) = \bar{T'} \rightarrow \bar{T}' \quad \text{override}(m, C<\pi>, \bar{T} \rightarrow \bar{T}) \quad \text{Aux-Override} \]

**Figure 20:** Auxiliary Judgments. Source: [2].
CT(C) = \textbf{class} C<\alpha, \beta> \textbf{extends} C'<\beta> \textbf{assumes} \gamma \rightarrow \delta \{ D; L; F; K \}  

\[
\begin{align*}
\mathcal{F} &= T \downarrow = \text{fieldDecls}(C') = T' \downarrow' \\
\text{fieldDecls}(C) &= (T \downarrow), T' \downarrow'
\end{align*}
\]

Aux-Fields-Decl

\[
\begin{align*}
\text{fieldDecls(\textbf{Object})} &= \emptyset \\
\text{fieldDecls}(\tau) &= \emptyset
\end{align*}
\]

Aux-Fields-Decl-Obj

\[
\begin{align*}
(T_R m(\mathcal{T} \tau) \{ \text{return } e; \}) &\in M \\
\text{mtypeDecl}(m, C) &= T \rightarrow T_R
\end{align*}
\]

Aux-MTypeDecl1

\[
\begin{align*}
m &\text{ is not defined in } M \\
\text{mtypeDecl}(m, C) &= \text{mtypeDecl}(m, C')
\end{align*}
\]

Aux-MTypeDecl2

\textbf{Figure 21:} Auxiliary judgments without substitution of actual with formal domain parameters.