June 2005

Generalizing Parametricity Using Information Flow (Extended Version)

Geoffrey Washburn
University of Pennsylvania

Stephanie C. Weirich
University of Pennsylvania, sweirich@cis.upenn.edu

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-05-04.

Generalizing Parametricity Using Information Flow (Extended Version)

Abstract

Run-time type analysis allows programmers to easily and concisely define operations based upon type structure, such as serialization, iterators, and structural equality. However, when types can be inspected at run time, nothing is secret. A module writer cannot use type abstraction to hide implementation details from clients: clients can determine the structure of these supposedly "abstract" data types. Furthermore, access control mechanisms do not help isolate the implementation of abstract datatypes from their clients. Buggy or malicious authorized modules may leak type information to unauthorized clients, so module implementors cannot reliably tell which parts of a program rely on their type definitions.

Currently, module implementors rely on parametric polymorphism to provide integrity and confidentiality guarantees about their abstract datatypes. However, standard parametricity does not hold for languages with run-time type analysis; this paper shows how to generalize parametricity so that it does. The key is to augment the type system with annotations about information-flow. Implementors can then easily see which parts of a program depend on the chosen implementation by tracking the flow of dynamic type information.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-05-04.

Generalizing Parametricity Using Information Flow (Extended Version)*

Geoffrey Washburn Stephanie Weirich
{geoffw, sweirich}@cis.upenn.edu
University of Pennsylvania Technical Report MS-CIS-05-04
Department of Computer and Information Science
Levine Hall, University of Pennsylvania
3330 Walnut Street, Philadelphia, Pennsylvania
19104-6389

June 24, 2005

Abstract

Run-time type analysis allows programmers to easily and concisely define operations based upon type structure, such as serialization, iterators, and structural equality. However, when types can be inspected at run time, nothing is secret. A module writer cannot use type abstraction to hide implementation details from clients: clients can determine the structure of these supposedly "abstract" data types. Furthermore, access control mechanisms do not help isolate the implementation of abstract datatypes from their clients. Buggy or malicious authorized modules may leak type information to unauthorized clients, so module implementors cannot reliably tell which parts of a program rely on their type definitions.

Currently, module implementors rely on parametric polymorphism to provide integrity and confidentiality guarantees about their abstract datatypes. However, standard parametricity does not hold for languages with run-time type analysis; this paper shows how to generalize parametricity so that it does. The key is to augment the type system with annotations about information-flow. Implementors can then easily see which parts of a program depend on the chosen implementation by tracking the flow of dynamic type information.

*This is an extended version of the paper that appeared in the Twentieth Annual IEEE Symposium on Logic in Computer Science [WW05].

1 Introduction

Type analysis is an important programming idiom. Traditional applications for type analysis include serialization, structural equality, cloning and iteration. Many systems use type analysis for more sophisticated purposes such as generating user interfaces, testing code, implementing debuggers and XML support. For this reason, it is important to support type analysis in modern programming languages.

A canonical example of run-time type analysis is the generic structural equality function.

```
fun eq['a] =
  typecase 'a of
    bool    =>
      fn (x:bool, y:bool) =>
        if x then y else false
  | 'b * 'c =>
      fn (x:'b*'c, y:'b*'c) =>
        eq ['b] (fst x, fst y) &&
        eq ['c] (snd x, snd y)
  | ...
```

The `eq` function analyzes its type argument `'a` and returns an equality function for that type. More complex examples of type analysis include generic serialization and type-safe casts [Wei00]. Type-safe casts are especially important in systems with dynamic loading, as they are used to verify that reconstituted values have the expected type [HWC01].

Authors of abstract datatypes can use generic operations to quickly build implementations for their datatypes. For example, because equality for the following `Employee.t` datatype is structural, one may implement it via generic equality.

```
module Employee = struct
  (* name, SSN, address and salary *)
  type t = string * int * string * int
  (* An equality for this type. *)
  fun empEq (x : t) (y : t) =
    Generic.eq [t] (x,y)
end :> sig
  type t
  val empEq : t -> t -> bool
end
```

Although type analysis is very useful, it can also be dangerous. When types are analyzable, software developers cannot be sure that abstraction boundaries will be respected and that code will operate in a compositional fashion. As a consequence, type analysis may destroy properties of *integrity* and *confidentiality* that the author of the `Employee` module expects. Using a type-safe cast, anyone may create a value of type `Employee.t`. Although the type will be correct, other invariants not captured by the type system may be broken. For example, the following malicious code creates an employee with an invalid (negative) salary

```
val (forged : Employee.t) =
 case (Generic.cast
        [string * int * string * int]
        [Employee.t]) of
   SOME f =>
     f ("R U Kidding", 0, "none", -10)
 | NONE   => error "oops!"
```

Furthermore, even if the author of the `Employee` module tries to keep aspects of the employee data type hidden, another module can simply use generic operations to discover them. For example, if no accessor was provided to the salary component of an `Employee.t`, the following malicious code can extract it

```
val spy (x : Employee.t) : int =
 case (Generic.cast [Employee.t]
        [string * int * string * int])
 of SOME f => let (_, _, _, salary) = f x
              in salary end
  | NONE -> error "oops!"
```

One answer to these problems is to simply prohibit run-time type analysis. However, we believe the benefits of type analysis are too compelling to abandon altogether. Therefore, we propose a basis for a language that permits type analysis, yet allows module writers to define integrity and confidentiality policies for abstract datatypes. In particular, we want authors to know how changing their abstract datatype affects the rest of a program and how their code depends on other abstract types they use.

In languages without type analysis, these questions are easy to answer. Authors rely on *parametric polymorphism* to provide guarantees. The author knows the rest of the program must treat her abstract datatypes as black boxes that may only be "pushed around," not inspected, modified or created. Dually authors are restricted in the same fashion when using other abstract datatypes. In the presence of type analysis, the programmer cannot know what code may depend on the definition of an abstract

datatype. Any part of the program can dynamically discover the underlying type and introduce dependencies on its definition.

In the past it has been suggested that type analysis could be tamed by distinguishing between analyzable and unanalyzable types [HM95]. Unfortunately, just controlling which parts of the program may analyze a type does not allow programmers to answer our questions. Imagine an extension, not unlike "friends" in C++, where an author can specify which modules may analyze a type. In the following code, modules `A` and `B` may analyze the type `A.t`, and modules `B` and `C` may analyze the type `B.u`.

```
module A = struct
 type t = int
 val x = 3
end :> sig
 type t permit A, B
 val x : t
end

module B = struct
 type u = A.t
 val y = A.x
end :> sig
 type u permit B, C
 val y : u
end

module C = struct
 val z = case (cast [B.u] [int]) of
           SOME f => "It is an int"
         | NONE   => "It is not an int"
end :> sig
 val z : string
end
```

Module `C` is not parametric with respect to `A.t`, even though module `C` is not allowed to analyze `A.t`: If the implementation of `A.t` changes, so does the value of `C.z`. Despite restricting analysis of `A.t` to `A` and `B`, the implementation of the type has been leaked to a third-party. Furthermore, because the type `B.u` is abstract, the author of `A` cannot know of the dependency. Access control places undue trust in a client not to provide others with the capabilities and information it has been granted. Consequently, we must look beyond access-control for a method of answering the desired questions.

We propose that tracking the flow of type information through a program with *information-flow labels* allows a programmer to easily determine how their type definitions influence the rest of the program. Information-flow extends a standard type system with elements of a lattice that describes the information content for each computation. For example, we could use a simple lattice containing two points L (low-security) and H (high-security). A type $\texttt{bool}^{\mathsf{H}}$ then means the expression it describes could use "high-security" information to produce the resulting boolean, while an expression of type $\texttt{bool}^{\mathsf{L}}$ only requires "low-security" information to produce its result. The novelty

of our approach compared to previous information-flow type systems is that we also label kinds to track the information content of type constructors.

To regain parametric reasoning about abstract types in the presence of type analysis, we can label types with an information content that can be tracked. Consequently, computations depending on those types must also have that label.

```
module A = struct
 type t = int
 val x = 3
end :> sig
 type t^H
 val x : t^L
end

module B = struct
 type u = A.t
 val y = A.x
end :> sig
 type u^H
 val y : u^L
end

module C = struct
 val z = case (cast [B.u] [int]) of
           SOME f => "It is an int"
         | NONE   => "It is not an int"
end :> sig
 val z : string^H
end
```

In the revised example, module `A` is sealed with a signature that indicates that the type definition `t` depends upon high-security information and the value `x` only on low-security information. The type `B.u` and value `C.z` must both be labeled as high security because they depend upon the high-security information in `A.t`. The presence of a label H alerts the author of `A` to a dependency.

Furthermore, only the module `A` can create values of type `A.t` that are labeled with L. Using type analysis to create values of type `A.t` would taint the result with H. Therefore, if module `A` requires its inputs be of type $\texttt{A.t}^{\mathsf{L}}$, then it is impossible to use its functions with forged values. The author now has a guarantee that module invariants will be maintained and the integrity of her abstraction will not be violated.

Information flow avoids the problems of access control because labels are propagated even when no analysis occurs. For example, the identity function can be assigned both the type $\texttt{A.t}^{\mathsf{L}} \xrightarrow{\mathsf{L}} \texttt{A.t}^{\mathsf{L}}$ and the type $\texttt{A.t}^{\mathsf{H}} \xrightarrow{\mathsf{L}} \texttt{A.t}^{\mathsf{H}}$ witnessing that it propagates the information content of the argument. Here the function type constructor $\longrightarrow$ is itself labeled to indicate the information content of creating the function—creating the identity function does not require any information.

In the next section, we describe a core calculus for combining information-flow and run-time type analysis. We then follow with our key contribution: By tracking the

flow of type information, it is possible to generalize the standard parametricity theorem to handle languages with run-time type analysis. This generalized theorem can be used in the same manner as parametricity to establish integrity and confidentiality properties.

2 THE λ_{SECi} LANGUAGE

λ_{SECi} is a core calculus combining information flow and type analysis. We designed λ_{SECi} to be as simple as possible while still retaining the flavor of the problem. It is derived from the type-analyzing language λ_i^{ML} developed by Harper and Morrisett [HM95] and the information-flow security language λ_{SEC} of Zdancewic [Zda02]. We based λ_{SECi} on λ_i^{ML} because it provides a simple yet expressive model of run-time type analysis. The language λ_i^{ML} was developed as an intermediate language for efficiently compiling parametric polymorphism. Similarly, λ_{SEC} was developed to study information flow in the context of the simply-typed λ-calculus.

2.1 RUN-TIME TYPE ANALYSIS

The grammar for λ_{SECi} appears in Figure 1. It is a predicative, call-by-value polymorphic λ-calculus with booleans, functions and recursion. Fix-points are separate from functions to make nontermination aspect of proofs modular.

As in λ_i^{ML}, type constructors, τ, which can be analyzed at run-time, are separated from types, σ, which describe terms. We conjecture our results extend to languages with impredicative polymorphism, but for clarity and to emphasize the relationship with λ_i^{ML}, we do not examine the problem in this paper. Also for simplicity, we do not allow higher-order polymorphism, but conjecture that our results extend to that feature as well.

The language of type constructors consists of the simply-typed λ-calculus and three primitive constructors that correspond to types: bool, $\tau_1 \longrightarrow \tau_2$, and $\tau_1 \times \tau_2$.

The term form **typecase** can be used to define operations that depend on run-time type information. This term takes a constructor to scrutinize, τ, as well as three branches $(e_{\mathsf{bool}}, e_{\longrightarrow}, e_{\times})$ corresponding to the primitive constructors. During evaluation the constructor argument must be reduced to determine its head form so that a branch can be chosen.

$$\frac{\tau \rightsquigarrow^* \mathsf{bool}}{\textbf{typecase}\ [\gamma.\sigma]\ \tau\ e_{\mathsf{bool}}\ e_{\longrightarrow}\ e_{\times} \rightsquigarrow e_{\mathsf{int}}}\ \text{EV:TCASE-BOOL}$$

$$\frac{\tau \rightsquigarrow^* \tau_1 \longrightarrow \tau_2}{\textbf{typecase}\ [\gamma.\sigma]\ \tau\ e_{\mathsf{bool}}\ e_{\longrightarrow}\ e_{\times} \rightsquigarrow e_{\longrightarrow}[\tau_1][\tau_2]}\ \text{EV:TCASE-ARR}$$

kinds

$\kappa ::= \star^{\ell}$ *types*

$\mid \kappa_1 \xrightarrow{\ell} \kappa_2$ *operators*

type constructors

$\tau ::= \alpha \mid \lambda\alpha{:}\kappa.\tau \mid \tau_1\tau_2$ *λ-calculus*

$\mid \mathsf{bool}$ *booleans*

$\mid \tau_1 \longrightarrow \tau_2$ *functions*

$\mid \tau_1 \times \tau_2$ *products*

$\mid \mathsf{Typerec}\ \tau\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times}$ *analysis*

types

$\sigma ::= (\tau)^{\ell}$ *injection*

$\mid \sigma_1 \xrightarrow{\ell} \sigma_2$ *functions*

$\mid \sigma_1 \times^{\ell} \sigma_2$ *products*

$\mid \forall^{\ell_1}\alpha{:}\star^{\ell_2}.\sigma$ *con poly*

terms

$e ::= \mathbf{true} \mid \mathbf{false}$ *booleans*

$\mid x \mid \lambda x{:}\sigma.e \mid e_1e_2$ *λ-calculus*

$\mid \langle e_1, e_2\rangle \mid \mathbf{fst}\ e \mid \mathbf{snd}\ e$ *tuples*

$\mid \Lambda\alpha{:}\star^{\ell}.e \mid e[\tau]$ *con poly*

$\mid \mathbf{fix}\ x{:}\sigma.e$ *fix-point*

$\mid \mathbf{if}\ e_1\ \mathbf{then}\ e_2\ \mathbf{else}\ e_3$ *conditional*

$\mid \mathbf{typecase}[\gamma.\sigma]\ \tau\ e_{\mathsf{bool}}\ e_{\longrightarrow}\ e_{\times}$ *analysis*

values

$v ::= \mathbf{true} \mid \mathbf{false} \mid \lambda x{:}\sigma.e \mid \langle v_1, v_2\rangle \mid \Lambda\alpha{:}\star^{\ell}.e$

term substitutions	γ	$::=$	$\cdot \mid \gamma, [e/x]$
type substitutions	δ	$::=$	$\cdot \mid \delta, [\tau/\alpha]$
term variable contexts	Γ	$::=$	$\cdot \mid \Gamma, x{:}\sigma$
type variable contexts	Δ	$::=$	$\cdot \mid \Delta, \alpha{:}\kappa$

Figure 1: The λ_{SECi} language

$$\frac{\tau \rightsquigarrow^* \tau_1 \times \tau_2}{\textbf{typecase}\ [\gamma.\sigma]\ \tau\ e_{\mathsf{bool}}\ e_{\longrightarrow}\ e_{\times} \rightsquigarrow e_{\times}[\tau_1][\tau_2]}\ \text{EV:TCASE-PROD}$$

We write $e \rightsquigarrow e'$ to mean that term e reduces in a single step to e' and $\tau \rightsquigarrow \tau'$ to mean that constructor τ makes a weak-head reduction step to τ'.

λ_{SECi} also includes a constructor, Typerec, for analyzing type information. Without Typerec, it is impossible to assign a type to some useful terms that perform type analysis [HM95]. Typerec implements a *paramorphism* (a type of fold) over the structure of the argument constructor. When the head of the argument is one of the three primitive constructors, Typerec will apply the appropriate branch to the constituent types, as well as the recursive invocation of Typerec on them.

$$\frac{}{\mathsf{Typerec}\ (\mathsf{bool})\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times} \rightsquigarrow \tau_{\mathsf{bool}}}\ \text{WHR:TREC-BOOL}$$

$$\frac{}{\begin{array}{l}\mathsf{Typerec}\ (\tau_1 \longrightarrow \tau_2)\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times} \rightsquigarrow \\ \quad \tau_{\longrightarrow}\ \tau_1\ \tau_2\ (\mathsf{Typerec}\ \tau_1\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times}) \\ \qquad\qquad (\mathsf{Typerec}\ \tau_2\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times})\end{array}}\ \text{WHR:TREC-ARR}$$

$$\frac{}{\begin{array}{l}\mathsf{Typerec}\ (\tau_1 \times \tau_2)\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times} \rightsquigarrow \\ \quad \tau_{\times}\ \tau_1\ \tau_2\ (\mathsf{Typerec}\ \tau_1\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times}) \\ \qquad\qquad (\mathsf{Typerec}\ \tau_2\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times})\end{array}}\ \text{WHR:TREC-PROD}$$

2.2 The information content of constructors

Information-flow type systems track the flow of information by annotating types with labels that specify the information content of the terms they describe. Because our type constructors have computational content (and influence the evaluation of terms) in λ_{SECi}, we must also label kinds.

Labels, ℓ, are drawn from an unspecified join semi-lattice, with a least element ($\bot$), joins ($\sqcup$) for finite subsets of elements in the lattice, and a partial order ($\sqsubseteq$). The actual lattice used by the type system is determined by the desired confidentiality and integrity policies of the program. Intuitively, the higher a label is in the lattice, the more restricted the information content of a constructor or term should be. For most examples in this paper, we use a simple two point lattice ($\bot$ for low security, $\top$ for high security) that tracks the dynamic discovery of a single type definition. In practice, any lattice with the specified structure could be used. An example of a practical lattice with richer internal structure is the Decentralized Label Model (DLM) of Myers and Liskov [ML00].

$$\frac{\alpha{:}\kappa \in \Delta}{\Delta \vdash \alpha : \kappa}\ \textsc{wfc:var} \qquad \frac{}{\Delta \vdash \mathsf{bool} : \star^{\bot}}\ \textsc{wfc:bool}$$

$$\frac{\Delta \vdash \tau_1 : \star^{\ell_1} \qquad \Delta \vdash \tau_2 : \star^{\ell_2}}{\Delta \vdash \tau_1 \longrightarrow \tau_2 : \star^{\ell_1 \sqcup \ell_2}}\ \textsc{wfc:arr}$$

$$\frac{\Delta \vdash \tau_1 : \star^{\ell_1} \qquad \Delta \vdash \tau_2 : \star^{\ell_2}}{\Delta \vdash \tau_1 \times \tau_2 : \star^{\ell_1 \sqcup \ell_2}}\ \textsc{wfc:prod} \qquad \frac{\Delta, \alpha{:}\kappa_1 \vdash \tau : \kappa_2}{\Delta \vdash \lambda\alpha{:}\kappa_1.\tau : \kappa_1 \xrightarrow{\bot} \kappa_2}\ \textsc{wfc:abs}$$

$$\frac{\Delta \vdash \tau_1 : \kappa_1 \xrightarrow{\ell} \kappa_2 \qquad \Delta \vdash \tau_2 : \kappa_1}{\Delta \vdash \tau_1 \tau_2 : \kappa_2 \sqcup \ell}\ \textsc{wfc:app}$$

$$\frac{\begin{array}{c}\Delta \vdash \tau : \star^{\ell} \\ \ell \sqsubseteq \ell' \qquad \Delta \vdash \tau_{\mathsf{bool}} : \kappa \qquad \Delta \vdash \tau_{\longrightarrow} : \star^{\ell} \xrightarrow{\ell'} \star^{\ell} \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \\ \Delta \vdash \tau_{\times} : \star^{\ell} \xrightarrow{\ell'} \star^{\ell} \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \qquad \text{where } \ell' = \mathcal{L}(\kappa)\end{array}}{\Delta \vdash \mathsf{Typerec}\ \tau\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times} : \kappa}\ \textsc{wfc:trec}$$

$$\frac{\Delta \vdash \tau : \kappa_1 \qquad \kappa_1 \leq \kappa_2}{\Delta \vdash \tau : \kappa_2}\ \textsc{wfc:sub}$$

Figure 2: Constructor well-formedness

The labels on kinds describe the information content of type constructors. The kind of a constructor (and therefore its information content) is described using the judgement $\Delta \vdash \tau : \kappa$, read as "constructor τ is well-formed having kind κ with respect to the type variable context Δ." The operator $\mathcal{L}(\kappa)$, defined in Figure 3, extracts the label of a kind.

Our calculus is conservative: If the label of κ is ℓ, then the information content of a constructor of kind κ is *at most* ℓ. The information level of a constructor can be raised via subsumption. As kinds are labeled, the ordering $\sqsubseteq$ on labels induces a sub-kinding relation, $\kappa_1 \leq \kappa_2$. A kind $\star^{\ell_1}$ is a sub-kind of $\star^{\ell_2}$ if $\ell_1 \sqsubseteq \ell_2$. Sub-kinding for function kinds is standard. The relation is reflexive and transitive by definition.

The label of a constructor τ, of kind $\star^{\ell}$, also describes the information gained

$$\mathcal{L}(\star^\ell) \triangleq \ell \qquad \mathcal{L}(\kappa_1 \xrightarrow{\ell} \kappa_2) \triangleq \ell$$

$$\mathcal{L}((\tau)^\ell) \triangleq \ell \qquad \mathcal{L}(\sigma_1 \xrightarrow{\ell} \sigma_2) \triangleq \ell$$
$$\mathcal{L}(\sigma_1 \times^\ell \sigma_2) \triangleq \ell \qquad \mathcal{L}(\forall^{\ell_1} \alpha{:}\star^{\ell_2}.\sigma) \triangleq \ell_1$$

Figure 3: Kind and type label operators

when the constructor is analyzed. Type variables (such as `Employee.t`) may be given a high security level so that their information content may be traced throughout the program. For example, the kind of a Typerec constructor must be labeled at least as high as the analyzed constructor τ. This requirement accounts for information gained by inspecting τ.

$$\frac{\begin{array}{c}\Delta \vdash \tau : \star^\ell \\ \ell \sqsubseteq \ell' \qquad \Delta \vdash \tau_{\mathsf{bool}} : \kappa \qquad \Delta \vdash \tau_{\longrightarrow} : \star^\ell \xrightarrow{\ell'} \star^\ell \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \\ \Delta \vdash \tau_{\times} : \star^\ell \xrightarrow{\ell'} \star^\ell \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \qquad \text{where } \ell' = \mathcal{L}(\kappa)\end{array}}{\Delta \vdash \mathsf{Typerec}\ \tau\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times} : \kappa}\ \textsc{wfc:trec}$$

By default the label on the bool constructor is set to $\bot$. The label of the kind for function and product constructors must be at least as high as the join of its two constituent constructors. This is because the label must reflect the information content of the entire constructor.

To trace information flows through type applications, the kinds of type functions, $\kappa_1 \xrightarrow{\ell} \kappa_2$, have a label ℓ that represents the information propagated by invoking the function. The information, ℓ, is propagated into the result of application as $\kappa_2 \sqcup \ell$. This is shorthand for relabeling κ_2 with $\mathcal{L}(\kappa_2) \sqcup \ell$.

2.3 Tracking information flow in terms

The labels on types describe the information content of terms. We use the judgement $\Delta^\star \mid \Gamma \vdash e : \sigma$ to mean that "term e is well-formed with type σ with respect to the term context Γ and the type context $\Delta^\star$." We use the notation $\Delta^\star$ to denote type variable contexts restricted to variables of base kind $\star^\ell$ for any label ℓ. As we did for kinds, we define (in Figure 3) the operator $\mathcal{L}(\sigma)$ to extract the label of a type. Also, the judgement $\Delta^\star \vdash \sigma$ is used to indicate that "type σ is well-formed with respect to type context $\Delta^\star$." Like constructors, the information content specified by labels for terms is conservative. The lattice ordering induces a subtyping judgement $\Delta^\star \vdash \sigma_1 \leq \sigma_2$, and subsumption can be used to raise the information level of a term.

$$\frac{\Delta^\star \vdash \Gamma}{\Delta^\star \mid \Gamma \vdash \mathbf{true} : (\mathsf{bool})^\bot}\ \textsc{wft:true} \qquad \frac{\Delta^\star \vdash \Gamma}{\Delta^\star \mid \Gamma \vdash \mathbf{false} : (\mathsf{bool})^\bot}\ \textsc{wft:false}$$

$$\frac{\Delta^\star \vdash \Gamma \qquad x : \sigma \in \Gamma}{\Delta^\star \mid \Gamma \vdash x : \sigma}\ \textsc{wft:var} \qquad \frac{\Delta^\star \mid \Gamma, x{:}\sigma_1 \vdash e : \sigma_2 \qquad \Delta^\star \vdash \sigma_1}{\Delta^\star \mid \Gamma \vdash \lambda x{:}\sigma_1.e : \sigma_1 \xrightarrow{\bot} \sigma_2}\ \textsc{wft:abs}$$

$$\frac{\Delta^\star \mid \Gamma \vdash e_1 : \sigma_1 \xrightarrow{\ell} \sigma_2 \qquad \Delta^\star \mid \Gamma \vdash e_2 : \sigma_1}{\Delta^\star \mid \Gamma \vdash e_1 e_2 : \sigma_2 \sqcup \ell}\ \textsc{wft:app}$$

$$\frac{\Delta^\star, \alpha{:}\star^\ell \mid \Gamma \vdash e : \sigma}{\Delta^\star \mid \Gamma \vdash \Lambda\alpha{:}\star^\ell.e : \forall^\bot \alpha{:}\star^\ell.\sigma}\ \textsc{wft:tabs}$$

$$\frac{\Delta^\star \mid \Gamma \vdash e : \forall^\ell \alpha{:}\star^{\ell'}.\sigma \qquad \Delta^\star \vdash \tau : \star^{\ell'}}{\Delta^\star \mid \Gamma \vdash e[\tau] : \sigma[\tau/\alpha] \sqcup \ell}\ \textsc{wft:tapp}$$

$$\frac{\Delta^\star \mid \Gamma \vdash e_1 : \sigma_1 \qquad \Delta^\star \mid \Gamma \vdash e_2 : \sigma_2}{\Delta^\star \mid \Gamma \vdash \langle e_1, e_2 \rangle : \sigma_1 \times^\bot \sigma_2}\ \textsc{wft:pair} \qquad \frac{\Delta^\star \mid \Gamma \vdash e : \sigma_1 \times^\ell \sigma_2}{\Delta^\star \mid \Gamma \vdash \mathbf{fst}\, e : \sigma_1 \sqcup \ell}\ \textsc{wft:fst}$$

$$\frac{\Delta^\star \mid \Gamma \vdash e : \sigma_1 \times^\ell \sigma_2}{\Delta^\star \mid \Gamma \vdash \mathbf{snd}\, e : \sigma_2 \sqcup \ell}\ \textsc{wft:snd} \qquad \frac{\Delta^\star \mid \Gamma, x{:}\sigma \vdash e : \sigma \qquad \Delta^\star \vdash \sigma}{\Delta^\star \mid \Gamma \vdash \mathbf{fix}\, x{:}\sigma.e : \sigma}\ \textsc{wft:fix}$$

$$\frac{\Delta^\star \mid \Gamma \vdash e_1 : (\mathsf{bool})^\ell \qquad \Delta^\star \mid \Gamma \vdash e_2 : \sigma \qquad \Delta^\star \mid \Gamma \vdash e_3 : \sigma}{\Delta^\star \mid \Gamma \vdash \mathbf{if}\, e_1\, \mathbf{then}\, e_2\, \mathbf{else}\, e_3 : \sigma \sqcup \ell}\ \textsc{wft:if}$$

$$\frac{\begin{array}{c}\Delta^\star \vdash \tau : \star^\ell \qquad \Delta^\star, \gamma{:}\star^\ell \vdash \sigma \qquad \ell \sqsubseteq \ell' \qquad \Delta^\star \mid \Gamma \vdash e_{\mathsf{bool}} : \sigma[\mathsf{bool}/\gamma] \\ \Delta^\star \mid \Gamma \vdash e_{\longrightarrow} : \forall^{\ell'} \alpha{:}\star^\ell.\forall^{\ell'} \beta{:}\star^\ell.\sigma[\alpha \longrightarrow \beta/\gamma] \\ \Delta^\star \mid \Gamma \vdash e_\times : \forall^{\ell'} \alpha{:}\star^\ell.\forall^{\ell'} \beta{:}\star^\ell.\sigma[\alpha \times \beta/\gamma] \qquad \text{where } \ell' = \mathcal{L}(\sigma[\tau/\gamma])\end{array}}{\Delta^\star \mid \Gamma \vdash \mathbf{typecase}\, [\gamma.\sigma]\, \tau\, e_{\mathsf{bool}}\, e_{\longrightarrow}\, e_\times : \sigma[\tau/\gamma]}\ \textsc{wft:tcase}$$

$$\frac{\Delta^\star \mid \Gamma \vdash e : \sigma_1 \qquad \Delta^\star \vdash \sigma_1 \leq \sigma_2}{\Delta^\star \mid \Gamma \vdash e : \sigma_2}\ \textsc{wft:sub}$$

Figure 4: Term well-formedness

The types of λ_{SECi} include the standard ones for functions $\sigma_1 \xrightarrow{\ell} \sigma_2$, products $\sigma_1 \times^{\ell} \sigma_2$, and quantified types $\forall^{\ell_1} \alpha{:}\star^{\ell_2}.\sigma$, plus those that are computed by type constructors $(\tau)^{\ell}$. Note that in the well-formedness rule for types formed from type constructors, shown below

$$\frac{\Delta^\star \vdash \tau : \star^\ell}{\Delta^\star \vdash (\tau)^\perp}\ \textsc{wftp:con}$$

there is no need for a connection between the label ℓ on the kind and the label on the type. That is because ℓ describes the information content of τ, while the label ℓ' on $(\tau)^{\ell'}$ describes the information content of a term with type $(\tau)^{\ell'}$. It is sound to discard ℓ, because once a constructor has been coerced to a type it may only be used statically to describe terms and cannot be analyzed.

Information flow is tracked at the term level analogously to the type level. Term abstractions, $\sigma_1 \xrightarrow{\ell} \sigma_2$, like type functions propagate some information ℓ when the are applied. Similarly, type abstractions, $\forall^{\ell_1} \alpha{:}\star^{\ell_2}.\sigma$, propagate some information ℓ_1 when they are applied. The label ℓ_2 describes the information content of the constructor that may be used to instantiate the type abstraction. For products, $\sigma_1 \times^{\ell} \sigma_2$, the label ℓ indicates the information propagated when one of its components is projected.

Like Typerec, **typecase** examines the structure of the scrutinee and learns the information it carries, so the label ℓ' on the type of the term must be at least as high in the lattice as the label ℓ on the scrutinee.

$$\frac{\begin{array}{c}\Delta^\star \vdash \tau : \star^\ell \qquad \Delta^\star, \gamma{:}\star^\ell \vdash \sigma \qquad \ell \sqsubseteq \ell' \qquad \Delta^\star \mid \Gamma \vdash e_{\mathsf{bool}} : \sigma[\mathsf{bool}/\gamma] \\ \Delta^\star \mid \Gamma \vdash e_{\longrightarrow} : \forall^{\ell'} \alpha{:}\star^{\ell}.\forall^{\ell'} \beta{:}\star^{\ell}.\sigma[\alpha \longrightarrow \beta/\gamma] \\ \Delta^\star \mid \Gamma \vdash e_{\times} : \forall^{\ell'} \alpha{:}\star^{\ell}.\forall^{\ell'} \beta{:}\star^{\ell}.\sigma[\alpha \times \beta/\gamma] \qquad \text{where } \ell' = \mathcal{L}(\sigma[\tau/\gamma])\end{array}}{\Delta^\star \mid \Gamma \vdash \textbf{typecase}\ [\gamma.\sigma]\ \tau\ e_{\mathsf{bool}}\ e_{\longrightarrow}\ e_{\times} : \sigma[\tau/\gamma]}\ \textsc{wft:tcase}$$

Because the type of a **typecase** term can depend upon the scrutinized constructor τ, an annotation, $[\gamma.\sigma]$, is required for type checking.

2.4 Soundness

λ_{SECi} has the basic property expected from a typed language, that well-typed programs will not go wrong.

Theorem 2.1 (Type Safety)**.** *If* $\cdot \vdash e : \sigma$ *then* e *either evaluates to a value or diverges.*

Proof. The theorem is proven syntactically as a corollary of the standard progress and preservation lemmas [WF94]. For details of the proof, see Appendix B. □

$$\frac{\alpha \mapsto R \in \eta \qquad v_1 R v_2}{\eta \vdash v_1 \sim v_2 : \alpha}\ \textsc{lr:var} \qquad \frac{}{\eta \vdash v \sim v : \mathsf{bool}}\ \textsc{lr:bool}$$

$$\frac{\forall(\eta \vdash e_1 \approx e_2 : \sigma_1).\eta \vdash v_1 e_1 \approx v_2 e_2 : \sigma_2}{\eta \vdash v_1 \sim v_2 : \sigma_1 \to \sigma_2}\ \textsc{lr:arr}$$

$$\frac{\eta \vdash \mathbf{fst}\, v_1 \approx \mathbf{fst}\, v_2 : \sigma_1 \qquad \eta \vdash \mathbf{snd}\, v_1 \approx \mathbf{snd}\, v_2 : \sigma_2}{\eta \vdash v_1 \sim v_2 : \sigma_1 \times \sigma_2}\ \textsc{lr:prod}$$

$$\frac{\forall \tau_1, \tau_2.\forall(R \in \tau_1 \leftrightarrow \tau_2).\eta, \alpha \mapsto R \vdash v_1[\tau_1] \approx v_2[\tau_2] : \sigma}{\eta \vdash v_1 \sim v_2 : \forall \alpha{:}\star.\sigma}\ \textsc{lr:all}$$

$$\frac{e_1 \rightsquigarrow^* v_1 \qquad e_2 \rightsquigarrow^* v_2 \qquad \eta \vdash v_1 \sim v_2 : \sigma}{\eta \vdash e_1 \approx e_2 : \sigma}\ \textsc{lr:term} \qquad \frac{e_1 \uparrow \qquad e_2 \uparrow}{\eta \vdash e_1 \approx e_2 : \sigma}\ \textsc{lr:divr}$$

Figure 5: Logically related terms

3 Generalizing Parametricity

The parametricity theorem has long been used to reason about programs in languages with parametric polymorphism [Rey83]. For example, the theorem can be used to show that different implementations of an abstract datatype do not influence the behavior of the program or to show that external modules cannot forge values of abstract types. These are only a few of the corollaries of the parametricity theorem. This sections starts with an overview of the standard parametricity theorem, and then examine how it can be generalized for λ_{SECi}.

3.1 Parametricity

For pedagogical purposes, this section and and the following section only considers only the core of λ_{SECi} without type constructors, security labels, or type analysis. That is, a simple predicative polymorphic λ-calculus [Rey83]. None of the results presented in these sections are new. Informally, the parametricity theorem states that well-typed expressions, after applying related substitutions for their free type and term variables,

$$\frac{\forall \alpha{:}\star \in \Delta^\star.(\eta(\alpha) \in \delta_1(\alpha) \leftrightarrow \delta_2(\alpha))}{\eta \vdash \delta_1 \approx \delta_2 : \Delta^\star} \text{ TSLR:BASE}$$

$$\frac{\forall x{:}\sigma \in \Gamma.(\eta \vdash \gamma_1(x) \approx \gamma_2(x) : \sigma)}{\eta \vdash \gamma_1 \approx \gamma_2 : \Gamma} \text{ SLR:BASE}$$

Figure 6: Substitutions for parametricity

are related to themselves. The power of the theorem comes from the fact that terms typed by universally quantified type variables can be related by any relation. Section 3.2 considers some important corollaries of the parametricity theorem for reasoning about data abstraction in programs.

The logical relation used by the parametricity theorem is defined in Figure 5. Terms are related with the judgement $\eta \vdash e_1 \approx e_2 : \sigma$, read as "terms e_1 and e_2 are related at type σ with respect to the relations in η." Terms are related if they evaluate to related values, or both diverge. We write $e \uparrow$ to denote divergence.

The judgement $\eta \vdash v_1 \sim v_2 : \sigma$ means that "values v_1 and v_2 are related at type σ with respect to the relations in η". The relation between values is defined inductively over types σ, potentially containing free type variables. To account for these variables, the relations are parameterized by a map, η, between type variables and binary relations on values. This map is used when σ is a type variable (see rule LR:VAR). If σ is bool, the relation is identity. Typical for logical relations, values of function type are related only if when applied to related arguments, they produce related results. Likewise, values of product types are related if the projections of their components are related.

The most important rule defines the relationship between values of type $\forall \alpha{:}\star.\sigma$. Polymorphic values are related if their instantiations with *any* pair of types are related. Furthermore, we can use *any* relation R between values of those types as the relation on α. We use the notation $R \in \tau_1 \leftrightarrow \tau_2$ to mean that R is a binary relation on values of type τ_1 and of type τ_2. If quantification over types of higher kind were allowed, R must be a function on relations. This extension is orthogonal to our result, so we restrict ourselves to polymorphism over kind $\star$.

To state the parametricity theorem the notion of related substitutions for types and related terms must be defined. In Figure 6, the rule TSLR:BASE states that a relation mapping η is well-formed with respect to two type substitutions $\delta_1$and δ_2 for the variables in the type context $\Delta^\star$. There are no restrictions on the range of the type substitutions. On the other hand, SLR:BASE requires that a pair of term substitutions

for the variables in Γ must map to related terms. Even though λ_{SECi} has call-by-value semantics, term substitutions must map to terms, not values. Otherwise, it would it be impossible to prove the case for fix-points which requires a term substitution.

With these definitions it is possible to state the parametricity theorem for our restricted language:

Theorem 3.1 (Parametricity)**.** *If* $\Delta^\star \mid \Gamma \vdash e : \sigma$ *and*
$\eta \vdash \delta_1 \approx \delta_2 : \Delta^\star$ *and*
$\eta \vdash \gamma_1 \approx \gamma_2 : \Gamma$ *then*
$\eta \vdash \delta_1(\gamma_1(e)) \approx \delta_2(\gamma_2(e)) : \sigma$.

Proof. The proof is by induction on the typing judgment with appeals to supporting lemmas. □

The primary complication in this proof arises in the case for type application, where we would like to show that a term $v[\sigma']$ is related to itself (after appropriate substitutions) at type $\sigma[\sigma'/\alpha]$. By induction, we know that v is related to itself at type $\forall\alpha{:}\star.\sigma$, so by inversion of the rule LR:ALL we may conclude that $v[\sigma']$ is related to itself at type σ, where the type α is mapped to any relation R. However, what we need to show is that $v[\sigma']$ is related to itself at type $\sigma[\sigma'/\alpha]$. The trick is to instantiate R with the relation $\{(v_1, v_2) \mid \eta \vdash v_1 \approx v_2 : \tau\}$ and use the following type substitution lemma.

Lemma 3.2 (Type substitution for parametricity)**.**
If $\eta \vdash \delta_1 \approx \delta_2 : \Delta^\star$ *then*
$\eta \vdash e_1 \approx e_2 : \sigma[\tau/\alpha]$ *iff*
$\eta, \alpha \mapsto R \vdash e_1 \approx e_2 : \sigma$ *where*
R *is the relation* $\{(v_1, v_2) \mid \eta \vdash v_1 \approx v_2 : \tau\}$ *and* $\delta_i(\alpha) = \delta_i(\tau)$.

Proof. The proof in both directions of the biconditional is by induction on the structure of the term relation. □

Another significant complication in the proof is circularity in relating fix-points. To show that **fix** $x{:}\sigma.e$ is related to itself we must show that e is related to itself under an extended substitution term substitution where $\gamma_1(x) = \gamma_1(\textbf{fix}\ x{:}\sigma.e)$ and $\gamma_2(x) = \gamma_2(\textbf{fix}\ x{:}\sigma.e)$. However, for these substitutions to be related, we need to know that the fix-point is related to itself. Which is what were just trying to show! To escape this problem we apply a syntactic technique from Pitts [Pit00]. We define a restricted fix-point that can only be unfolded a finite number of times before diverging. The term $\textbf{fix}_{n+1}\ x{:}\sigma.e$ unwinds to $e[(\textbf{fix}_n\ x{:}\sigma.e)/x]$. By definition $\textbf{fix}_0\ x{:}\sigma.e$ always diverges. It is then straightforward to show that for any n, $\textbf{fix}_n\ x{:}\sigma.e$ is related to itself. Then the following continuity lemma can be used to prove that unbounded fix-points are related to themselves.

Lemma 3.3 (Continuity). *If* $\eta \vdash \delta_1 \approx \delta_2 : \Delta^\star$ *and*
for all n, $\eta \vdash \mathbf{fix}_n\ x{:}\sigma_1.e_1 \approx \mathbf{fix}_n\ x{:}\sigma_2.e_2 : \sigma$
where $\delta_1(\sigma) = \sigma_1$ *and* $\delta_2(\sigma) = \sigma_2$ *then*
$\eta \vdash \mathbf{fix}\ x{:}\sigma_1.e_1 \approx \mathbf{fix}\ x{:}\sigma_2.e_2 : \sigma$.

Proof. There are four cases.

- If both $\mathbf{fix}\ x{:}\sigma_1.e_1$ and $\mathbf{fix}\ x{:}\sigma_2.e_2$ diverge they are trivially related by LR:DIVR.
- If both $\mathbf{fix}\ x{:}\sigma_1.e_1$ and $\mathbf{fix}\ x{:}\sigma_2.e_2$ converge to a value, they must do so with some finite number of unwindings, n, and it is possible to instantiate the assumption, $\eta \vdash \mathbf{fix}_n\ x{:}\sigma_1.e_1 \approx \mathbf{fix}_n\ x{:}\sigma_2.e_2 : \sigma$, accordingly, to obtain the a derivation showing that they are related.
- In the last two cases either $\mathbf{fix}\ x{:}\sigma_1.e_1$ and $\mathbf{fix}\ x{:}\sigma_2.e_2$ diverge and the other converges to a value. However, if it does so it does so in a finite number of steps. Then instantiating $\eta \vdash \mathbf{fix}_n\ x{:}\sigma_1.e_1 \approx \mathbf{fix}_n\ x{:}\sigma_2.e_2 : \sigma$, we have a derivation that the other could have converged after a finite number of steps as well, leading to a contradiction.

□

3.2 Applications of the parametricity theorem

The parametricity theorem has been used for many purposes, most famously for deriving *free theorems* about functions in the polymorphic λ-calculus, just by looking at their types [Wad89]. Our purpose is more similar to that of Reynolds: reasoning about the properties of programs in the presence of type abstraction. While Reynolds saw the need to separate parametric polymorphism from ad-hoc polymorphism, we show how to generalize his work to both sorts of polymorphism.

Corollaries of Theorem 3.1 provide important results for reasoning about abstract types in programs. Many specific properties can be proven as a consequence of parametricity, but we believe the following two are representative of what a programmer desires.

Corollary 3.4 (Confidentiality). *If* $\cdot \vdash v_1 : \tau_1$ *and*
$\cdot \vdash v_2 : \tau_2$ *then*
$\alpha{:}\star \mid x{:}\alpha \vdash e : \mathsf{bool}$ *and*
$e[\tau_1/\alpha][v_1/x] \rightsquigarrow^* v$ *iff* $e[\tau_2/\alpha][v_2/x] \rightsquigarrow^* v$.

Proof. First construct a derivation that $\cdot \mid \cdot \vdash \Lambda\alpha{:}\star.\lambda x{:}\alpha.e : \forall\alpha{:}\star.\alpha \longrightarrow \mathsf{bool}$ using the appropriate typing rules and then appeal to Theorem 3.1 to obtain

$$\cdot \vdash \Lambda\alpha{:}\star.\lambda x{:}\alpha.e \sim \Lambda\alpha{:}\star.\lambda x{:}\alpha.e : \forall\alpha{:}\star.\alpha \longrightarrow \mathsf{bool}$$

Next, by inversion on LR:ALL and instantiation with the relation

$$R = \{(v_1, v_2) \mid (\cdot \mid \cdot \vdash v_1 : \tau_1), (\cdot \mid \cdot \vdash v_2 : \tau_2)\}$$

we can conclude that

$$\cdot, \alpha \mapsto R \vdash (\Lambda\alpha{:}\star.\lambda x{:}\alpha.e)[\tau_1] \approx (\Lambda\alpha{:}\star.\lambda x{:}\alpha.e)[\tau_2] : \alpha \longrightarrow \mathsf{bool}$$

By straightforward application of LR:VAR we have that

$$\cdot, \alpha \mapsto R \vdash v_1 \sim v_2 : \alpha$$

so by application of LR:TERM, inversion on LR:ARR, and instantiation we know

$$\cdot, \alpha \mapsto R \vdash (\Lambda\alpha{:}\star.\lambda x{:}\alpha.e)[\tau_1]v_1 \approx (\Lambda\alpha{:}\star.\lambda x{:}\alpha.e)[\tau_2]v_2 : \mathsf{bool}$$

Finally, because the relation is closed under reduction we have LR:ARR and instantiation we have

$$\cdot, \alpha \mapsto R \vdash e[\tau_1/\alpha][v_1/x] \approx e[\tau_2/\alpha][v_2/x] : \mathsf{bool}$$

from which the desired conclusion can be obtained by simple inversion. □

This first corollary says that a programmer is free to change the implementation of an abstract type without affecting the behavior of a program. It is the essence behind parametric polymorphism – type information is not allowed to influence program execution and values of abstract type must be treated parametrically.

Corollary 3.5 (Integrity)**.** *If* $\alpha{:}\star \mid \cdot \vdash e : \alpha$ *then* $e[\tau/\alpha]$ *for any* τ *must diverge.*

Proof. First construct a derivation that $\cdot \mid \cdot \vdash \Lambda\alpha{:}\star.e : \forall\alpha{:}\alpha$ using the appropriate typing rules, then appeal to Theorem 3.1 to obtain

$$\cdot \vdash \Lambda\alpha{:}\star.e \sim \Lambda\alpha{:}\star.e : \forall\alpha{:}\star.\alpha$$

Now assume an arbitrary τ. By inversion on LR:ALL and instantiation we can conclude

$$\cdot, \alpha \mapsto \varnothing \vdash (\Lambda\alpha{:}\star.e)[\tau] \approx (\Lambda\alpha{:}\star.e)[\tau] : \alpha$$

Because the relation is closed under reduction we have that

$$\cdot, \alpha \mapsto \varnothing \vdash e[\tau/\alpha] \approx e[\tau/\alpha] : \alpha$$

Furthermore, by inversion either $e[\tau/\alpha] \rightsquigarrow^* v$ or $e[\tau/\alpha] \uparrow$. However in the former case that would mean that

$$\cdot, \alpha \mapsto \varnothing \vdash v \sim v : \alpha$$

which by inversion on LR:VAR is impossible because there is no v such that $v \varnothing v$. Therefore $e[\tau/\alpha] \uparrow$. □

This second corollary states that there is no way for a program to invent values of an abstract type, violating the integrity of the abstraction.

3.3 Parametricity and type analysis

Consider the following λ_{SECi} term (eliding labels)

$$\textbf{typecase}\ [\gamma.\mathsf{bool}]\ \alpha\ \textbf{true}(\Lambda\alpha{:}\star.\Lambda\beta{:}\star.\textbf{false})(\Lambda\alpha{:}\star.\Lambda\beta{:}\star.\textbf{false})$$

This term violates Corollary 3.4, because we can substitute bool and $\mathsf{bool} \times \mathsf{bool}$ for α and it will produce different values: **true** versus **false**.

Still, we would like to state properties similar to Corollaries 3.4 and 3.5 for λ_{SECi}. It is not possible to directly extend the inductive proof for **typecase**. The proof would require that the two terms would produce related results, even when they may analyze different constructors. Furthermore, λ_{SECi} presents another complication: The weak-head normal forms of types include (for example) $\mathsf{Typerec}$ with its scrutinee a variable. Therefore, the logical relation must be extended to be inductively defined upon these sorts of types.

To solve the problem with **typecase**, we require that the constructors used to instantiate polymorphic types be related to each other, as defined in the next subsection. Labeling kinds is the key to making this change practical, because it means the relation need not be the identity relation when types are used parametrically. Now consider a labeled version of the earlier example

$$\textbf{typecase}\ [\gamma.(\mathsf{int})^{\top}]\ \alpha\ 1\ (\Lambda\alpha{:}\star^{\top}.\Lambda\beta{:}\star^{\top}.2)\ (\Lambda\alpha{:}\star^{\top}.\Lambda\beta{:}\star^{\top}.3)$$

If α has kind $\star^{\top}$ then the entire expression will have type $(\mathsf{int})^{\top}$ which means that to an observer at level $\bot$ the result will appear identical regardless of whether we substitute int or $\mathsf{int} \times \mathsf{int}$ for α.

We solve the problem of making the logical relation inductively defined upon weak-head normal types with unusual shapes by generalizing the trick of quantifying over all relations between values of given types, to quantifying over families of relations on values of the correct types.

3.4 Related constructors

The first step towards a generalized parametricity theorem is formalizing what it means for type constructors to be related. We write $\tau_1 \approx_\ell \tau_2 : \kappa$ to mean closed constructors τ_1 and τ_2 are related at kind κ with respect to an observer at level ℓ in the label lattice. Similarly, the judgement $\nu_1 \sim_\ell \nu_2 : \kappa$ is used to indicate that closed weak-head normal constructors ν_1 and ν_2 are related at kind κ with respect to an observer at level ℓ. The grammar of weak-head normal constructors and relations on constructors is defined in Figures 8 and 7, respectively.

$$\frac{\ell_1 \not\sqsubseteq \ell_0}{\nu_1 \sim_{\ell_0} \nu_2 : \star^{\ell_1}}\ \textsc{tslr:type-opaq} \qquad \frac{\ell_1 \sqsubseteq \ell_0}{\mathsf{bool} \sim_{\ell_0} \mathsf{bool} : \star^{\ell_1}}\ \textsc{tslr:type-bool}$$

$$\frac{\ell_1 \sqcup \ell_2 \sqsubseteq \ell_3 \qquad \ell_3 \sqsubseteq \ell_0 \qquad \tau_1 \approx_{\ell_0} \tau_3 : \star^{\ell_1} \qquad \tau_2 \approx_{\ell_0} \tau_4 : \star^{\ell_2}}{\tau_1 \longrightarrow \tau_2 \sim_{\ell_0} \tau_3 \longrightarrow \tau_4 : \star^{\ell_3}}\ \textsc{tslr:type-arr}$$

$$\frac{\ell_1 \sqcup \ell_2 \sqsubseteq \ell_3 \qquad \ell_3 \sqsubseteq \ell_0 \qquad \tau_1 \approx_{\ell_0} \tau_3 : \star^{\ell_1} \qquad \tau_2 \approx_{\ell_0} \tau_4 : \star^{\ell_2}}{\tau_1 \times \tau_2 \sim_{\ell_0} \tau_3 \times \tau_4 : \star^{\ell_3}}\ \textsc{tslr:type-prod}$$

$$\frac{\forall(\tau_1 \approx_{\ell_0} \tau_2 : \kappa_1).\nu_1\tau_1 \approx_{\ell_0} \nu_2\tau_2 : \kappa_2 \sqcup \ell_1}{\nu_1 \sim_{\ell_0} \nu_2 : \kappa_1 \xrightarrow{\ell_1} \kappa_2}\ \textsc{tslr:arr}$$

$$\frac{\tau_1 \rightsquigarrow^* \nu_1 \qquad \tau_2 \rightsquigarrow^* \nu_2 \qquad \nu_1 \sim_{\ell_0} \nu_2 : \kappa}{\tau_1 \approx_{\ell_0} \tau_2 : \kappa}\ \textsc{tsclr:base}$$

Figure 7: Logically related constructors

The rule for type functions, TSLR:ARR, is standard for logical relations. There are four rules for kind $\star$. The first, TSLR:TYPE-OPAQ, codifies that if the label of the constructors is higher than the observer they are indistinguishable. The remaining three state that if the label of a primitive constructor is less than the observer, their components must appear related to the observer. Constructors that are not in normal form are related by TSCLR:BASE if and only if their weak-head normal forms are related.

As suggested by TSLR:TYPE-OPAQ, if two constructors carry information more restrictive than the level of the observer, the observer shouldn't be able to tell them apart. For example, $\mathsf{bool} : \star^{\top}$ and $\mathsf{bool} \times \mathsf{bool} : \star^{\top}$ which carry "high-security" information $\top$, will be indistinguishable to an observer at a "low-security" level $\bot$. This is formalized in the following lemma.

Lemma 3.6 (Obliviousness for constructors)**.** *If* $\cdot \vdash \tau_1, \tau_2 : \kappa$ *and* $\mathcal{L}(\kappa) \not\sqsubseteq \ell_0$ *then* $\tau_1 \approx_{\ell_0} \tau_2 : \kappa$.

Proof. For the details of the proof, see Appendix D. □

Another important property of the relation is that is closed under subsumption.

constructor contexts
$\rho ::= \bullet \mid \mathsf{Typerec}\ \rho\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times} \mid \rho\,\tau$

weak-head normal-form constructors
$\nu ::= \mathsf{bool} \mid \tau_1 \longrightarrow \tau_2 \mid \tau_1 \times \tau_2 \mid \lambda\alpha{:}\kappa.\tau$

weak-head normal-form types
$\zeta ::= (\mathsf{bool})^{\ell} \mid (\rho\{\alpha\})^{\ell} \mid \sigma_1 \xrightarrow{\ell} \sigma_2 \mid \sigma_1 \times^{\ell} \sigma_2 \mid \forall^{\ell_1}\alpha{:}\star^{\ell_2}.\sigma$

Figure 8: Additional syntactic forms

Lemma 3.7 (Constructor relation consistent)**.**

If $\kappa_1 \leq \kappa_2$ *and* $\tau_1 \approx_{\ell_0} \tau_2 : \kappa_1$ *then* $\tau_1 \approx_{\ell_0} \tau_2 : \kappa_2$

Proof. For the details of the proof, see Appendix D. □

Finally, we can state a substitution theorem for constructors that is a simpler version of parametricity:

Lemma 3.8 (Substitution for constructors)**.** *If* $\Delta \vdash \tau : \kappa$ *and* $\delta_1 \approx_{\ell_0} \delta_2 : \Delta$ *then* $\delta_1(\tau) \approx_{\ell_0} \delta_2(\tau) : \kappa$.

Proof. For the details of the proof, see Appendix D. □

where related type substitutions map type variables to related constructors, as defined in the following rule

$$\frac{\forall \alpha{:}\kappa \in \Delta.(\delta_1(\alpha) \approx_{\ell_0} \delta_2(\alpha) : \kappa)}{\delta_1 \approx_{\ell_0} \delta_2 : \Delta}\ \textsc{tsslr:base}$$

3.5 Related terms

As with constructors, we parameterize the logical relation on terms by an observer ℓ. We write $\eta \vdash e_1 \approx_{\ell} e_2 : \sigma$ to indicate that terms e_1 and e_2 are related to an observer at level ℓ at type σ, with the relation mapping η. As with constructors we distinguish between related terms and normal forms, writing the judgement $\eta \vdash \nu_1 \sim_{\ell} \nu_2 : \zeta$ to indicate that values ν_1 and ν_2 are related to an observer at level ℓ at the weak-head normal type ζ, with the relation mapping η. These relations, as defined in Figure 10, are similar to the ones in Figure 5. One difference is that we only relate values at weak-head normal types ζ, defined in Figure 8.

$$\frac{\tau \rightsquigarrow \tau'}{(\tau)^\ell \rightsquigarrow (\tau')^\ell}\ \textsc{whr:inj-tc} \qquad \frac{}{(\tau_1 \longrightarrow \tau_2)^\ell \rightsquigarrow (\tau_1)^\ell \xrightarrow{\ell} (\tau_2)^\ell}\ \textsc{whr:inj-arr}$$

$$\frac{}{(\tau_1 \times \tau_2)^\ell \rightsquigarrow (\tau_1)^\ell \times^\ell (\tau_2)^\ell}\ \textsc{whr:inj-prod}$$

Figure 9: Type reduction

Restricting the value relation to weak-head normal types makes the logical relation much easier to state and understand. For example, the term $\langle \mathbf{true}, \mathbf{false} \rangle$ is well typed with the equivalent types $(\mathsf{bool} \times \mathsf{bool})^\ell$ and $(\mathsf{bool})^\ell \times^\ell (\mathsf{bool})^\ell$. However, restricting to weak-head normal types means that only the case for $(\mathsf{bool})^\ell \times^\ell (\mathsf{bool})^\ell$ be considered.

Like constructors, the relation over terms is defined so that terms typed at a level greater than the observer will be indistinguishable. This is enforced by the precondition $\ell_1 \sqsubseteq \ell_0$ found in SLR:CON and SLR:BOOL. The antecedent relations in SLR:ARR, SLR:PROD, and SLR:ALL have their types joined with ℓ_1; this accounts for information gained by destructing the value. The following lemma verifies our intuition about indistinguishability:

Lemma 3.9 (Obliviousness for terms). *If* $\Delta^\star \mid \cdot \vdash e_1, e_2 : \sigma$ *and*
$\delta_1 \approx_{\ell_0} \delta_2 : \Delta^\star$ *and*
$\mathcal{L}(\sigma) \not\sqsubseteq \ell_0$ *then*
$\eta \vdash \delta_1(e_1) \approx_{\ell_0} \delta_2(e_2) : \sigma$.

Proof. For the details of the proof, see Appendix D. □

There are two other significant differences between Figures 5 and 10: additional preconditions in SLR:ALL, and generalizing LR:VAR to SLR:CON. The rule SLR:CON solves the problem with Typerec appearing in the weak-head normal form of types. It generalizes LR:VAR to terms related at a constructor that cannot be normalized further because of an undetermined type variable. We characterize these constructors with constructor contexts, ρ, defined in Figure 8. Contexts are holes $\bullet$, Typerecs of a context, or a context applied to an arbitrary constructor. We write $\rho\{\tau\}$ for filling a context's hole with τ.

Previously, values were related at a type variable only if they were in the relation mapped to that variable by η. Here η maps to families of relations. We write R^ℓ_ρ for the application of R to a label ℓ and a context ρ, yielding a relation. Therefore, when we

$$\frac{\alpha \mapsto R \in \eta \qquad \ell_1 \sqsubseteq \ell_0 \implies v_1 R_\rho^{\ell_1} v_2}{\eta \vdash v_1 \sim_{\ell_0} v_2 : (\rho\{\alpha\})^{\ell_1}} \text{ SLR:CON}$$

$$\frac{\ell_1 \sqsubseteq \ell_0 \implies v_1 = v_2}{\eta \vdash v_1 \sim_{\ell_0} v_2 : (\mathsf{bool})^{\ell_1}} \text{ SLR:BOOL}$$

$$\frac{\forall(\eta \vdash e_1 \approx_{\ell_0} e_2 : \sigma_1).\eta \vdash v_1 e_1 \approx_{\ell_0} v_2 e_2 : \sigma_2 \sqcup \ell_1}{\eta \vdash v_1 \sim_{\ell_0} v_2 : \sigma_1 \xrightarrow{\ell_1} \sigma_2} \text{ SLR:ARR}$$

$$\frac{\eta \vdash \mathbf{fst}\, v_1 \approx_{\ell_0} \mathbf{fst}\, v_2 : \sigma_1 \sqcup \ell_1 \qquad \eta \vdash \mathbf{snd}\, v_1 \approx_{\ell_0} \mathbf{snd}\, v_2 : \sigma_2 \sqcup \ell_1}{\eta \vdash v_1 \sim_{\ell_0} v_2 : \sigma_1 \times^{\ell_1} \sigma_2} \text{ SLR:PROD}$$

$$\frac{\begin{array}{c}\forall(\tau_1 \approx_{\ell_0} \tau_2 : \star^{\ell_2}).\forall(R_\rho^{\ell_2} \in \delta_1((\rho\{\tau_1\})^{\ell_2}) \leftrightarrow \delta_2((\rho\{\tau_2\})^{\ell_2})). \\ \eta, \alpha \mapsto R \vdash v_1[\tau_1] \approx_{\ell_0} v_2[\tau_2] : \sigma \sqcup \ell_1 \qquad R \text{ consistent}\end{array}}{\eta \vdash v_1 \sim_{\ell_0} v_2 : \forall^{\ell_1} \alpha{:}\star^{\ell_2}.\sigma} \text{ SLR:ALL}$$

$$\frac{e_1 \rightsquigarrow^* v_1 \qquad e_2 \rightsquigarrow^* v_2 \qquad \sigma \rightsquigarrow^* \zeta \qquad \eta \vdash v_1 \sim_{\ell_0} v_2 : \zeta}{\eta \vdash e_1 \approx_{\ell_0} e_2 : \sigma} \text{ SCLR:TERM}$$

$$\frac{(e_1 \uparrow) \vee (e_2 \uparrow)}{\eta \vdash e_1 \approx_{\ell_0} e_2 : \sigma} \text{ SCLR:DIVR}$$

Figure 10: Logically related terms

write $R_\rho^\ell \in \delta_1((\rho\{\tau_1\})^\ell) \leftrightarrow \delta_2((\rho\{\tau_2\})^\ell)$ we mean that R is a dependent function of ℓ and ρ yielding a relation on values of type $\delta_1((\rho\{\tau_1\})^\ell)$ and $\delta_2((\rho\{\tau_2\})^\ell)$.

Quantification over R is required to be consistent. In this context, that means if $v_1 R_\rho^{\ell_1} v_2$ and $\ell_1 \sqsubseteq \ell_2$ then $v_1 R_\rho^{\ell_2} v_2$. This is adequate for call-by-value because quantification is over families of value relations. Therefore requiring that R yield relations that are strict or preserve least-upper bounds is unnecessary, as values are always terminating. It is important that the logical relation itself is consistent, that is, closed under subsumption.

Lemma 3.10 (Term relation consistent). *If* $\delta_1 \approx_{\ell_0} \delta_2 : \Delta^\star$ *and*
$\eta \vdash \Delta^\star$ *and*
$\Delta^\star \vdash \sigma_1 \leq \sigma_2$ *and*
$\eta \vdash e_1 \approx_{\ell_0} e_2 : \sigma_1$ *then*
$\eta \vdash e_1 \approx_{\ell_0} e_2 : \sigma_2$.

Proof. For the details of the proof, see Appendix D. □

We write $\eta \vdash \Delta^\star$ to mean that the mapping η is well-formed with respect to a pair of type substitutions, δ_1 and δ_2, as defined in the rule:

$$\frac{\forall \alpha{:}\star^{\ell_1} \in \Delta^\star.(\eta(\alpha)^{\ell_1}_{\rho} \in \delta_1((\rho\{\alpha\})^{\ell_1}) \leftrightarrow \delta_2((\rho\{\alpha\})^{\ell_1})) \quad \eta(\alpha) \text{ consistent}}{\eta \vdash \Delta^\star} \textsc{relm:reg}$$

The last significant difference in Figure 5 is in SLR:DIVR. Terms are related if either diverges, as opposed to our earlier definition where divergent terms were only related to other divergent terms. At first, this change might seem like a significant weakening of the relation. In particular, the logical relation is no longer transitive. However, this definition is standard for information-flow logical relations proofs with recursion [ABHR99, Zda02]. We will discuss in more detail in Section 3.6 how this requirement is merely an artifact of call-by-value information-flow.

3.6 Generalized parametricity

Before stating the generalized parametricity theorem, the notation of related term substitutions must be defined. Given related type substitutions, $\delta_1 \approx_{\ell_0} \delta_2 : \Delta^\star$, and a well-formed mapping, $\eta \vdash \Delta^\star$, term substitutions are related if they map variables to related terms.

$$\frac{\forall x{:}\sigma \in \Gamma.(\eta \vdash \gamma_1(x) \approx_{\ell_0} \gamma_2(x) : \sigma)}{\eta \vdash \gamma_1 \approx_{\ell_0} \gamma_2 : \Gamma} \textsc{sslr:base}$$

The only change from SLR:BASE is the additional of a label ℓ_0 for the observer.

Theorem 3.11 (Generalized parametricity). *If* $\Delta^\star \mid \Gamma \vdash e : \sigma$ *and*
$\delta_1 \approx_{\ell_0} \delta_2 : \Delta^\star$ *and*
$\eta \vdash \Delta^\star$ *and*
$\eta \vdash \gamma_1 \approx_{\ell_0} \gamma_2 : \Gamma$ *then*
$\eta \vdash \delta_1(\gamma_1(e)) \approx_{\ell_0} \delta_2(\gamma_2(e)) : \sigma$.

Proof. As with standard parametricity, the proof is by induction over $\Delta^\star \mid \Gamma \vdash e : \sigma$. In addition to the lemmas mentioned in Sections 3.4 and 3.5, Lemma 3.3 must be extended in the straightforward manner. For more details of the proof, see Appendix D. □

We call it generalized parametricity because Theorem 3.1 can be recovered by a series of restrictions:

- Restrict the label lattice to two elements, $\bot$ and $\top$ where $\bot \sqsubseteq \top$.
- For every kind κ in $\Delta^\star$, Γ, e, and σ require $\mathcal{L}(\kappa) = \top$.
- For every type σ' in Γ, e, and σ require $\mathcal{L}(\sigma') = \bot$.
- Require that the observer be $\bot$.

Even with these restrictions, because of the difference in SCLR:DIVR, Theorem 3.11 makes a weaker claim about the termination behavior of related terms than Theorem 3.1. This is merely an artifact of call-by-value information-flow, but it does impact our results. Consider the generalized version of Corollary 3.4.

Corollary 3.12 (Confidentiality)**.** *If* $\alpha{:}\star^\top \mid x{:}(\alpha)^\bot \vdash e : (\mathsf{bool})^\bot$ *then for any* $\cdot \vdash v_1 : \tau_1$ *and* $\vdash v_2 : \tau_2$ *if* $e[\tau_1/\alpha][v_2/x]$ *and* $e[\tau_2/\alpha][v_2/x]$ *both terminate, they will produce the same value.*

Proof. For the details of the proof, see Appendix D. □

This corollary states that what we substitute for α and x will not affect the value computed by e. However, it is possible that our choice of α and x could cause e to diverge. What is happening?

Unlike standard parametricity, Theorem 3.11 has an explicit observer. Standard parametricity has an implicit observer that can observe all computation. What makes information-flow techniques work is that some computations are opaque to the observer. Furthermore, the results of these computations are also inaccessible to the observer, making them effectively dead code. However, because the operational semantics is call-by-value, dead code must be executed even though the result is never used. Therefore, we conjecture that using a call-by-need operational semantics an exact correspondence could be recovered; the only part of the proof that would need to change is the proof of obliviousness for terms, Lemma 3.9.

3.7 Applications of generalized parametricity

A typical corollary of Theorem 3.11 is normally called noninterference; that it is possible to substitute values indistinguishable to the present observer and get indistinguishable results.

Corollary 3.13 (Noninterference). *If* $\cdot, x{:}\sigma_1 \vdash e : \sigma_2$ *where* $\mathcal{L}(\sigma_1) \not\sqsubseteq \mathcal{L}(\sigma_2)$ *then for any* $\vdash v_1 : \sigma_1$ *and* $\vdash v_2 : \sigma_1$ *it is the case that if both* $e[v_1/x]$ *and* $e[v_2/x]$ *terminate, they will both produce the same value*

Proof. For the details of the proof, see Appendix D. □

More importantly, it is also possible to restate the corollaries of standard parametricity. The previous subsection stated the revised corollary for confidentiality. The same can be done for integrity:

Corollary 3.14 (Integrity). *If* $\alpha{:}\star^{\top} \mid \cdot \vdash e : (\alpha)^{\bot}$ *then* $e[\tau/\alpha]$ *for any* τ *must diverge.*

Proof. For the details of the proof, see Appendix D. □

Furthermore, it is also possible to make much richer and refined claims because the label lattice expands upon the implicit two level lattice used by parametricity.

4 Related work

λ_{SECi} draws heavily upon previous work on type analysis, parametricity, and information flow.

Most information flow systems use a lattice model originating from work by Bell and LaPadula [BL75] and Denning [Den76]. Volpano et al. [VSI96] showed that Denning's work could be formulated as type system and proved its soundness with respect to noninterference. Heintze and Riecke's formalized information-flow and integrity in a typed λ-calculus with references, the SLam calculus [HR98], and proved a number of soundness and noninterference results. Pottier and Simonet have developed an extension to ML, called FlowCaml, and have shown noninterference using an alternative syntactic technique [PS02].

Prior to our research, FlowCaml was the only language with polymorphism and a noninterference proof. FlowCaml does not consider run-time type analysis and can rely on standard parametricity for types. The noninterference result for λ_{SECi} directly builds upon the methods of Zdancewic [Zda02] and Pitts [Pit00].

Other researchers have noticed the connection between parametricity and noninterference. The work of Tse and Zdancewic [TZ04] compliments our research by showing how parametricity can be used to prove noninterference. Tse and Zdancewic do so by encoding Abadi, et al.'s [ABHR99] dependency core calculus into System $\mathbb{F}$.

The fact that run-time type analysis (and other forms of ad-hoc polymorphism) breaks parametricity has been long understood, but little has been done to reconcile the two. Leifer et al. [LPSW03] design a system that preserves type abstraction in the presence of (un)marshalling. This is a weaker result because marshalling is merely

a single instance of an operation using run-time type analysis. Rossberg [Ros03] and Vytiniotis et al. [VWW05] use generative types to hide type information in the presence of run-time analysis, relying on colored-brackets [GMZ00] to provide easy access. However, none of this work has formalized the abstraction properties that their systems provide.

5 Conclusion

With λ_{SECi}, we address the conflict between run-time type analysis and enforceable data abstractions. By labeling their type abstractions, software developers can easily observe dependencies.

However, this refinement comes at with the penalty of having to write many annotations for a program to type check. We have not investigated how pervasive the necessary annotations will prove in practice. Existing large scale languages, such as Jif [MCN$^+$] and FlowCaml [PS02], implement some form of information-flow inference, but they can be difficult to use. Languages based on λ_{SECi} have the advantage that if the only goal is to secure type abstractions and no type analysis is performed, then no information-flow annotations are necessary. Regardless, it will be imperative to study the cost of maintaining the necessary annotations in practical languages based upon λ_{SECi}.

Acknowledgements

This paper benefitted greatly from conversations with Steve Zdancewic and Stephen Tse. We also appreciate the insightful comments by anonymous reviewers on earlier revisions of this work. This work was supported by NSF grant 0347289, Career: Type-Directed Programming in Object-Oriented Languages.

Colophon

This document prepared using the LaTeX typesetting system created by Leslie Lamport with the memoir class. The document was processed using pdfTeX's microtypography extensions implemented by Hàn Thế Thành. The body text is set at 11pt. The serif typefaces are from the Minion Pro Opticals family, designed by Robert Slimbach of Adobe Systems. The sans-serif typefaces are from the Cronos Pro Opticals family, also designed by Robert Slimbach. The monospace typeface is Adobe Letter Gothic designed by Roger Roberson. The serif mathematics typeface is AMS Euler, designed by Hermann Zapf.

Bibliography

[ABHR99] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon Riecke. A core calculus of dependency. In *Proc. 26th ACM Symp. on Principles of Programming Languages*, pages 147–160, San Antonio, TX, January 1999.

[BL75] D. E. Bell and L. J. LaPadula. Secure computer system: Unified exposition and Multics interpretation. Technical Report ESD-TR-75-306, MITRE Corp. MTR-2997, Bedford, MA, 1975.

[Den76] Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, May 1976.

[GMZ00] Dan Grossman, Greg Morrisett, and Steve Zdancewic. Syntactic type abstraction. *Transactions on Programming Languages and Systems*, 22(6):1037–1080, November 2000.

[HM95] Robert Harper and Greg Morrisett. Compiling polymorphism using intensional type analysis. In *22nd ACM Symp. on Principles of Programming Languages*, pages 130–141, San Francisco, January 1995.

[HR98] Nevin Heintze and Jon G. Riecke. The SLam calculus: Programming with secrecy and integrity. In *Proc. 25th ACM Symp. on Principles of Programming Languages*, San Diego, CA, 1998.

[HWC01] Michael Hicks, Stephanie Weirich, and Karl Crary. Safe and flexible dynamic linking of native code. In R. Harper, editor, *Types in Compilation: Third International Workshop, TIC 2000; Montreal, Canada, September 21, 2000; Revised Selected Papers*, volume 2071 of *Lecture Notes in Computer Science*, pages 147–176. Springer, 2001.

[LPSW03] James J. Leifer, Gilles Peskine, Peter Sewell, and Keith Wansbrough. Global abstraction-safe marshalling with hash types. In *Proc. 8th ICFP*, pages 87–98, Uppsala, Sweden, 2003.

[MCN+] Andrew C. Myers, Stephen Chong, Nathaniel Nystrom, Lantian Zheng, and Steve Zdancewic. Jif: Java information flow. Software release. Located at `http://www.cs.cornell.edu/jif/`.

[ML00] Andrew C. Myers and Barbara Liskov. Protecting privacy using the decentralized label model. *Transactions on Software Engineering and Methodology*, 9(4):410–442, 2000.

[Mor95] Greg Morrisett. *Compiling with Types*. PhD thesis, Carnegie Mellon University, School of Computer Science, Pittsburgh, Pennsylvania, December 1995.

[Pit00] Andrew M. Pitts. Parametric polymorphism and operational equivalence. *Mathematical Structures in Computer Science*, 10:1–39, 2000.

[PS02] François Pottier and Vincent Simonet. Information flow inference for ML. In *Proc. 29th ACM Symp. on Principles of Programming Languages*, Portland, OR, January 2002.

[Rey83] John C. Reynolds. Types, abstraction, and parametric polymorphism. In R. E. A. Mason, editor, *Information Processing 83*, pages 513–523, Amsterdam, 1983. Elsevier Science Publishers B. V.

[Ros03] Andreas Rossberg. Generativity and dynamic opacity for abstract types. In *Proc. of the 5th International ACM Conference on Principles and Practice of Declarative Programming*, Uppsala, Sweden, August 2003. Extended version available from `http://www.ps.uni-sb.de/Papers/generativity-extended.html`.

[TZ04] Stephen Tse and Steve Zdancewic. Translating dependency into parametricity. In *Proc. of the 9th ACM International Conference on Functional Programming*, Snowbird, Utah, September 2004.

[VSI96] Dennis Volpano, Geoffrey Smith, and Cynthia Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.

[VWW05] Dimitrios Vytiniotis, Geoffrey Washburn, and Stephanie Weirich. An open and shut typecase. In *Proc. of the 2nd ACM Workshop on Types in Language Design and Implementation*, Longbeach, California, January 2005.

[Wad89] Philip Wadler. Theorems for free! In *FPCA89: Conference on Functional Programming Languages and Computer Architecture*, London, September 1989.

[Wei00] Stephanie Weirich. Type-safe cast: Functional pearl. In *Proceedings of theFifth International Conference on Functional Programming (ICFP)*, pages 58–67, Montreal, September 2000.

[WF94] Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115:38–94, 1994.

[WW05] Geoffrey Washburn and Stephanie Weirich. Generalizing parametricity using information flow. In *The 20th Annual IEEE Symposium on Logic in Computer Science (LICS 2005)*, Chicago, IL, June 2005.

[Zda02] Stephan Zdancewic. *Programming Languages for Information Security*. PhD thesis, Cornell University, 2002.

APPENDIX A THE λ_{SECi} LANGUAGE

Definition A.1 (Type Grammar)**.**

kinds

$$\kappa ::= \star^{\ell} \qquad \textit{types}$$
$$\mid\ \kappa_1 \xrightarrow{\ell} \kappa_2 \qquad \textit{operators}$$

type constructors

$$\tau ::= \alpha \mid \lambda\alpha{:}\kappa.\tau \mid \tau_1\tau_2 \qquad \lambda\textit{-calculus}$$
$$\mid\ \mathsf{bool} \qquad \textit{booleans}$$
$$\mid\ \tau_1 \longrightarrow \tau_2 \qquad \textit{functions}$$
$$\mid\ \tau_1 \times \tau_2 \qquad \textit{products}$$
$$\mid\ \mathsf{Typerec}\ \tau\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times} \qquad \textit{analysis}$$

weak-head normal-form constructors

$$\nu ::= \mathsf{bool} \mid \tau_1 \xrightarrow{\ell} \tau_2 \mid \tau_1 \times^{\ell} \tau_2 \mid \lambda\alpha{:}\kappa.\tau$$

constructor contexts

$$\rho ::= \bullet \mid \mathsf{Typerec}\ \rho\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times} \mid \rho\,\tau$$

types

$$\sigma ::= (\tau)^{\ell} \qquad \textit{injection}$$
$$\mid\ \sigma_1 \xrightarrow{\ell} \sigma_2 \qquad \textit{functions}$$
$$\mid\ \sigma_1 \times^{\ell} \sigma_2 \qquad \textit{products}$$
$$\mid\ \forall^{\ell_1}\alpha{:}\star^{\ell_2}.\sigma \qquad \textit{con poly}$$

weak-head normal-form types

$$\zeta ::= (\mathsf{bool})^{\ell} \mid (\rho\{\alpha\})^{\ell} \mid \sigma_1 \xrightarrow{\ell} \sigma_2 \mid \sigma_1 \times^{\ell} \sigma_2 \mid \forall^{\ell_1}\alpha{:}\star^{\ell_2}.\sigma$$

type substitutions	δ	$::=$	$\cdot \mid \delta, [\tau/\alpha]$
type variable contexts	Δ	$::=$	$\cdot \mid \Delta, \alpha{:}\kappa$

Definition A.2 (Term Grammar)**.**

$$
\begin{array}{lll}
& \textit{terms} & \\
e ::= & \textbf{true} \mid \textbf{false} & \textit{booleans} \\
\mid & x \mid \lambda x{:}\sigma.e \mid e_1 e_2 & \lambda\textit{-calculus} \\
\mid & \langle e_1, e_2 \rangle \mid \textbf{fst}\, e \mid \textbf{snd}\, e & \textit{tuples} \\
\mid & \Lambda\alpha{:}\star^{\ell}.e \mid e[\tau] & \textit{con poly} \\
\mid & \textbf{fix}\, x{:}\sigma.e & \textit{fix-point} \\
\mid & \textbf{if}\, e_1\, \textbf{then}\, e_2\, \textbf{else}\, e_3 & \textit{conditional} \\
\mid & \textbf{typecase}[\gamma.\sigma]\ \tau\ e_{\mathsf{bool}}\ e_{\longrightarrow}\ e_{\times} & \textit{analysis}
\end{array}
$$

$$
\begin{array}{ll}
& \textit{values} \\
v ::= & \textbf{true} \mid \textbf{false} \mid \lambda x{:}\sigma.e \mid \langle v_1, v_2 \rangle \mid \Lambda\alpha{:}\star^{\ell}.e
\end{array}
$$

$$
\begin{array}{llll}
\textit{term substitutions} & \gamma & ::= & \cdot \mid \gamma, [e/x] \\
\textit{term variable contexts} & \Gamma & ::= & \cdot \mid \Gamma, x{:}\sigma
\end{array}
$$

A.1 Static semantics

Definition A.3 (Sub-kinding)**.**

$$\frac{}{\kappa \leq \kappa}\ \textsc{sbk:refl} \qquad \frac{\kappa_1 \leq \kappa_2 \qquad \kappa_2 \leq \kappa_3}{\kappa_1 \leq \kappa_3}\ \textsc{sbk:trans} \qquad \frac{\ell_1 \sqsubseteq \ell_2}{\star^{\ell_1} \leq \star^{\ell_2}}\ \textsc{sbk:type}$$

$$\frac{\kappa_3 \leq \kappa_1 \qquad \kappa_2 \leq \kappa_4 \qquad \ell_1 \sqsubseteq \ell_2}{\kappa_1 \xrightarrow{\ell_1} \kappa_2 \leq \kappa_3 \xrightarrow{\ell_2} \kappa_4}\ \textsc{sbk:arr}$$

Definition A.4 (Constructor well-formedness)**.**

$$\frac{\alpha{:}\kappa \in \Delta}{\Delta \vdash \alpha : \kappa}\ \textsc{wfc:var} \qquad \frac{}{\Delta \vdash \mathsf{bool} : \star^{\perp}}\ \textsc{wfc:bool}$$

$$\frac{\Delta \vdash \tau_1 : \star^{\ell_1} \qquad \Delta \vdash \tau_2 : \star^{\ell_2}}{\Delta \vdash \tau_1 \longrightarrow \tau_2 : \star^{\ell_1 \sqcup \ell_2}}\ \textsc{wfc:arr}$$

$$\frac{\Delta \vdash \tau_1 : \star^{\ell_1} \qquad \Delta \vdash \tau_2 : \star^{\ell_2}}{\Delta \vdash \tau_1 \times \tau_2 : \star^{\ell_1 \sqcup \ell_2}}\ \textsc{wfc:prod} \qquad \frac{\Delta, \alpha{:}\kappa_1 \vdash \tau : \kappa_2}{\Delta \vdash \lambda\alpha{:}\kappa_1.\tau : \kappa_1 \xrightarrow{\perp} \kappa_2}\ \textsc{wfc:abs}$$

$$\frac{\Delta \vdash \tau_1 : \kappa_1 \xrightarrow{\ell} \kappa_2 \qquad \Delta \vdash \tau_2 : \kappa_1}{\Delta \vdash \tau_1 \tau_2 : \kappa_2 \sqcup \ell} \text{ WFC:APP}$$

$$\frac{\begin{array}{c}\Delta \vdash \tau : \star^{\ell} \\ \ell \sqsubseteq \ell' \qquad \Delta \vdash \tau_{\mathsf{bool}} : \kappa \qquad \Delta \vdash \tau_{\longrightarrow} : \star^{\ell} \xrightarrow{\ell'} \star^{\ell} \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \\ \Delta \vdash \tau_{\times} : \star^{\ell} \xrightarrow{\ell'} \star^{\ell} \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \qquad \textit{where } \ell' = \mathcal{L}(\kappa)\end{array}}{\Delta \vdash \mathsf{Typerec}\ \tau\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times} : \kappa} \text{ WFC:TREC}$$

$$\frac{\Delta \vdash \tau : \kappa_1 \qquad \kappa_1 \leq \kappa_2}{\Delta \vdash \tau : \kappa_2} \text{ WFC:SUB}$$

Definition A.5 (Constructor equivalence)**.**

$$\frac{\Delta \vdash \tau : \kappa}{\Delta \vdash \tau = \tau : \kappa} \text{ EQC:REFL} \qquad \frac{\Delta \vdash \tau_1 = \tau_2 : \kappa \qquad \Delta \vdash \tau_2 = \tau_3 : \kappa}{\Delta \vdash \tau_1 = \tau_3 : \kappa} \text{ EQC:TRANS}$$

$$\frac{\Delta \vdash \tau_2 = \tau_1 : \kappa}{\Delta \vdash \tau_1 = \tau_2 : \kappa} \text{ EQC:SYM} \qquad \frac{\Delta \vdash \tau_3 = \tau_1 : \star^{\ell_1} \qquad \Delta \vdash \tau_2 = \tau_4 : \star^{\ell_2}}{\Delta \vdash \tau_1 \longrightarrow \tau_2 = \tau_3 \longrightarrow \tau_4 : \star^{\ell_1 \sqcup \ell_2}} \text{ EQC:ARR}$$

$$\frac{\Delta \vdash \tau_3 = \tau_1 : \star^{\ell_1} \qquad \Delta \vdash \tau_2 = \tau_4 : \star^{\ell_2}}{\Delta \vdash \tau_1 \times \tau_2 = \tau_3 \times \tau_4 : \star^{\ell_1 \sqcup \ell_2}} \text{ EQC:PROD}$$

$$\frac{\Delta, \alpha{:}\kappa_1 \vdash \tau_1 = \tau_2 : \kappa_2}{\Delta \vdash \lambda\alpha{:}\kappa_1.\tau_1 = \lambda\alpha{:}\kappa_1.\tau_2 : \kappa_1 \xrightarrow{\bot} \kappa_2} \text{ EQC:ABS-CON}$$

$$\frac{\Delta \vdash (\lambda\alpha{:}\kappa_1.\tau_1)\tau_2 : \kappa_2}{\Delta \vdash (\lambda\alpha{:}\kappa_1.\tau_1)\tau_2 = \tau_1[\tau_2/\alpha] : \kappa_2} \text{ EQC:ABS-BETA}$$

$$\frac{\Delta \vdash \tau_1 = \tau_3 : \kappa_1 \xrightarrow{\ell} \kappa_2 \qquad \Delta \vdash \tau_2 = \tau_3 : \kappa_1}{\Delta \vdash \tau_1 \tau_2 = \tau_3 \tau_4 : \kappa_2 \sqcup \ell} \text{ EQC:APP}$$

$$\frac{\begin{array}{c}\Delta \vdash \tau_1 = \tau_2 : \star^{\ell} \qquad \Delta \vdash \tau_{\mathsf{bool}} = \tau'_{\mathsf{bool}} : \kappa \\ \Delta \vdash \tau_{\longrightarrow} = \tau'_{\longrightarrow} : \star^{\ell} \xrightarrow{\ell'} \star^{\ell} \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \\ \Delta \vdash \tau_{\times} = \tau'_{\times} : \star^{\ell} \xrightarrow{\ell'} \star^{\ell} \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \\ \ell \sqsubseteq \ell' \qquad \textit{where } \ell' = \mathcal{L}(\kappa)\end{array}}{\Delta \vdash \begin{array}{ll}\mathsf{Typerec}\ \tau_1 & = \mathsf{Typerec}\ \tau_2 \\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times} & \tau'_{\mathsf{bool}}\ \tau'_{\longrightarrow}\ \tau'_{\times}\end{array} : \kappa} \text{ EQC:TREC-CON}$$

$$\frac{\Delta \vdash \mathsf{Typerec\ bool}\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times} : \kappa}{\Delta \vdash \mathsf{Typerec\ bool}\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times} = \tau_{\mathsf{bool}} : \kappa}\ \textsc{eqc:trec-bool}$$

$$\frac{\Delta \vdash \mathsf{Typerec}\ (\tau_1 \longrightarrow \tau_2)\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times} : \kappa}{\Delta \vdash \mathsf{Typerec}\ \quad = \tau_{\longrightarrow}\tau_1\tau_2 \quad : \kappa}\ \textsc{eqc:trec-arr}$$
$$(\tau_1 \longrightarrow \tau_2)\quad (\mathsf{Typerec}\ \tau_1$$
$$\tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times}\quad\ \ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times})$$
$$(\mathsf{Typerec}\ \tau_2$$
$$\tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times})$$

$$\frac{\Delta \vdash \mathsf{Typerec}\ (\tau_1 \times \tau_2)\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times} : \kappa}{\Delta \vdash \mathsf{Typerec}\ \quad = \tau_{\times}\tau_1\tau_2 \quad : \kappa}\ \textsc{eqc:trec-prod}$$
$$(\tau_1 \times \tau_2)\quad (\mathsf{Typerec}\ \tau_1$$
$$\tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times}\quad\ \ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times})$$
$$(\mathsf{Typerec}\ \tau_2$$
$$\tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times})$$

$$\frac{\Delta \vdash \tau_1 = \tau_2 : \kappa_1 \qquad \kappa_1 \leq \kappa_2}{\Delta \vdash \tau_1 = \tau_2 : \kappa_2}\ \textsc{eqc:sub}$$

Definition A.6 (Type variable context restriction)**.** *We will write $\Delta^\star$ for those type variable contexts Δ where $\forall\alpha{:}\kappa \in \Delta$, $\kappa = \star^\ell$ for some ℓ.*

Definition A.7 (Subtyping)**.**

$$\frac{\Delta^\star \vdash \sigma}{\Delta^\star \vdash \sigma \leq \sigma}\ \textsc{sbt:refl} \qquad \frac{\Delta^\star \vdash \sigma_1 \leq \sigma_2 \qquad \Delta^\star \vdash \sigma_2 \leq \sigma_3}{\Delta^\star \vdash \sigma_1 \leq \sigma_3}\ \textsc{sbt:trans}$$

$$\frac{\Delta^\star \vdash \tau_1 = \tau_2 : \star^{\ell_1}}{\Delta^\star \vdash (\tau_1)^{\ell_2} \leq (\tau_2)^{\ell_2}}\ \textsc{sbt:con}$$

$$\frac{\Delta^\star \vdash \tau_1 \longrightarrow \tau_2 : \star^{\ell_1}}{\Delta^\star \vdash (\tau_1 \longrightarrow \tau_2)^{\ell_2} \leq (\tau_1)^{\ell_2} \xrightarrow{\ell_2} (\tau_2)^{\ell_2}}\ \textsc{sbt:con-arr1}$$

$$\frac{\Delta^\star \vdash \tau_1 \longrightarrow \tau_2 : \star^{\ell_1}}{\Delta^\star \vdash (\tau_1)^{\ell_2} \xrightarrow{\ell_2} (\tau_2)^{\ell_2} \leq (\tau_1 \longrightarrow \tau_2)^{\ell_2}}\ \textsc{sbt:con-arr2}$$

$$\frac{\Delta^\star \vdash \tau_1 \times \tau_2 : \star^{\ell_1}}{\Delta^\star \vdash (\tau_1 \times \tau_2)^{\ell_2} \leq (\tau_1)^{\ell_2} \times^{\ell_2} (\tau_2)^{\ell_2}}\ \textsc{sbt:con-prod1}$$

$$\frac{\Delta^\star \vdash \tau_1 \times \tau_2 : \star^{\ell_1}}{\Delta^\star \vdash (\tau_1)^{\ell_2} \times^{\ell_2} (\tau_2)^{\ell_2} \leq (\tau_1 \times \tau_2)^{\ell_2}}\ \textsc{sbt:con-prod2}$$

$$\frac{\Delta^\star \vdash \sigma_3 \leq \sigma_1 \qquad \Delta^\star \vdash \sigma_2 \leq \sigma_4 \qquad \ell_1 \sqsubseteq \ell_2}{\Delta^\star \vdash \sigma_1 \xrightarrow{\ell_1} \sigma_2 \leq \sigma_3 \xrightarrow{\ell_2} \sigma_4}\ \textsc{sbt:arr}$$

$$\frac{\Delta^\star \vdash \sigma_1 \leq \sigma_3 \qquad \Delta^\star \vdash \sigma_2 \leq \sigma_4 \qquad \ell_1 \sqsubseteq \ell_2}{\Delta^\star \vdash \sigma_1 \times^{\ell_1} \sigma_2 \leq \sigma_3 \times^{\ell_2} \sigma_4}\ \textsc{sbt:prod}$$

$$\frac{\Delta^\star, \alpha{:}\star^{\ell_4} \vdash \sigma_1 \leq \sigma_2 \qquad \ell_4 \sqsubseteq \ell_2 \qquad \ell_1 \sqsubseteq \ell_3}{\Delta^\star \vdash \forall^{\ell_1} \alpha{:}\star^{\ell_2}.\sigma_1 \leq \forall^{\ell_3} \alpha{:}\star^{\ell_4}.\sigma_2}\ \textsc{sbt:all}$$

Definition A.8 (Type well-formedness)**.**

$$\frac{\Delta^\star \vdash \tau : \star^\ell}{\Delta^\star \vdash (\tau)^\perp}\ \textsc{wftp:con} \qquad \frac{\Delta^\star \vdash \sigma_1 \qquad \Delta^\star \vdash \sigma_2}{\Delta^\star \vdash \sigma_1 \xrightarrow{\perp} \sigma_2}\ \textsc{wftp:arr}$$

$$\frac{\Delta^\star \vdash \sigma_1 \qquad \Delta^\star \vdash \sigma_2}{\Delta^\star \vdash \sigma_1 \times^\perp \sigma_2}\ \textsc{wftp:prod} \qquad \frac{\Delta^\star, \alpha{:}\star^\ell \vdash \sigma}{\Delta^\star \vdash \forall^\perp \alpha{:}\star^\ell.\sigma}\ \textsc{wftp:all}$$

$$\frac{\Delta^\star \vdash \sigma_1 \qquad \Delta^\star \vdash \sigma_1 \leq \sigma_2}{\Delta^\star \vdash \sigma_2}\ \textsc{wftp:sub}$$

Definition A.9 (Type equivalence)**.** *We define* $\Delta^\star \vdash \sigma_1 = \sigma_2$ *to mean that* $\Delta^\star \vdash \sigma_1 \leq \sigma_2$ *and* $\Delta^\star \vdash \sigma_2 \leq \sigma_1$*.*

Definition A.10 (Term variable context well-formedness)**.**

$$\frac{}{\Delta^\star \vdash \cdot}\ \textsc{wftc:empty} \qquad \frac{\Delta^\star \vdash \Gamma \qquad \Delta^\star \vdash \sigma}{\Delta^\star \vdash \Gamma, x{:}\sigma}\ \textsc{wftc:cons}$$

Definition A.11 (Term well-formedness)**.**

$$\frac{\Delta^\star \vdash \Gamma}{\Delta^\star \mid \Gamma \vdash \textbf{true} : (\mathsf{bool})^\perp}\ \textsc{wft:true} \qquad \frac{\Delta^\star \vdash \Gamma}{\Delta^\star \mid \Gamma \vdash \textbf{false} : (\mathsf{bool})^\perp}\ \textsc{wft:false}$$

$$\frac{\Delta^\star \vdash \Gamma \qquad x : \sigma \in \Gamma}{\Delta^\star \mid \Gamma \vdash x : \sigma}\ \textsc{wft:var} \qquad \frac{\Delta^\star \mid \Gamma, x{:}\sigma_1 \vdash e : \sigma_2 \qquad \Delta^\star \vdash \sigma_1}{\Delta^\star \mid \Gamma \vdash \lambda x{:}\sigma_1.e : \sigma_1 \xrightarrow{\perp} \sigma_2}\ \textsc{wft:abs}$$

$$\frac{\Delta^\star \mid \Gamma \vdash e_1 : \sigma_1 \xrightarrow{\ell} \sigma_2 \qquad \Delta^\star \mid \Gamma \vdash e_2 : \sigma_1}{\Delta^\star \mid \Gamma \vdash e_1 e_2 : \sigma_2 \sqcup \ell}\ \textsc{wft:app}$$

$$\frac{\Delta^\star, \alpha{:}\star^\ell \mid \Gamma \vdash e : \sigma}{\Delta^\star \mid \Gamma \vdash \Lambda\alpha{:}\star^\ell.e : \forall^\perp \alpha{:}\star^\ell.\sigma}\ \textsc{wft:tabs}$$

$$\frac{\Delta^\star \mid \Gamma \vdash e : \forall^\ell \alpha{:}\star^{\ell'}.\sigma \qquad \Delta^\star \vdash \tau : \star^{\ell'}}{\Delta^\star \mid \Gamma \vdash e[\tau] : \sigma[\tau/\alpha] \sqcup \ell}\ \textsc{wft:tapp}$$

$$\frac{\Delta^\star \mid \Gamma \vdash e_1 : \sigma_1 \qquad \Delta^\star \mid \Gamma \vdash e_2 : \sigma_2}{\Delta^\star \mid \Gamma \vdash \langle e_1, e_2\rangle : \sigma_1 \times^\bot \sigma_2}\ \textsc{wft:pair} \qquad \frac{\Delta^\star \mid \Gamma \vdash e : \sigma_1 \times^\ell \sigma_2}{\Delta^\star \mid \Gamma \vdash \mathbf{fst}\, e : \sigma_1 \sqcup \ell}\ \textsc{wft:fst}$$

$$\frac{\Delta^\star \mid \Gamma \vdash e : \sigma_1 \times^\ell \sigma_2}{\Delta^\star \mid \Gamma \vdash \mathbf{snd}\, e : \sigma_2 \sqcup \ell}\ \textsc{wft:snd} \qquad \frac{\Delta^\star \mid \Gamma, x{:}\sigma \vdash e : \sigma \qquad \Delta^\star \vdash \sigma}{\Delta^\star \mid \Gamma \vdash \mathbf{fix}\, x{:}\sigma.e : \sigma}\ \textsc{wft:fix}$$

$$\frac{\Delta^\star \mid \Gamma \vdash e_1 : (\mathsf{bool})^\ell \qquad \Delta^\star \mid \Gamma \vdash e_2 : \sigma \qquad \Delta^\star \mid \Gamma \vdash e_3 : \sigma}{\Delta^\star \mid \Gamma \vdash \mathbf{if}\, e_1\, \mathbf{then}\, e_2\, \mathbf{else}\, e_3 : \sigma \sqcup \ell}\ \textsc{wft:if}$$

$$\frac{\begin{array}{c}\Delta^\star \vdash \tau : \star^\ell \qquad \Delta^\star, \gamma{:}\star^\ell \vdash \sigma \qquad \ell \sqsubseteq \ell' \qquad \Delta^\star \mid \Gamma \vdash e_{\mathsf{bool}} : \sigma[\mathsf{bool}/\gamma] \\ \Delta^\star \mid \Gamma \vdash e_{\longrightarrow} : \forall^{\ell'} \alpha{:}\star^\ell.\forall^{\ell'} \beta{:}\star^\ell.\sigma[\alpha \longrightarrow \beta/\gamma] \\ \Delta^\star \mid \Gamma \vdash e_\times : \forall^{\ell'} \alpha{:}\star^\ell.\forall^{\ell'} \beta{:}\star^\ell.\sigma[\alpha \times \beta/\gamma] \qquad \textit{where } \ell' = \mathcal{L}(\sigma[\tau/\gamma])\end{array}}{\Delta^\star \mid \Gamma \vdash \mathbf{typecase}\ [\gamma.\sigma]\ \tau\ e_{\mathsf{bool}}\ e_{\longrightarrow}\ e_\times : \sigma[\tau/\gamma]}\ \textsc{wft:tcase}$$

$$\frac{\Delta^\star \mid \Gamma \vdash e : \sigma_1 \qquad \Delta^\star \vdash \sigma_1 \leq \sigma_2}{\Delta^\star \mid \Gamma \vdash e : \sigma_2}\ \textsc{wft:sub}$$

A.2 Dynamic semantics

Definition A.12 (Constructor reduction)**.**

$$\frac{\tau_1 \rightsquigarrow \tau_1'}{\tau_1\tau_2 \rightsquigarrow \tau_1'\tau_2}\ \textsc{whr:app-con} \qquad \frac{}{(\lambda\alpha{:}\kappa.\tau_1)\tau_2 \rightsquigarrow \tau_1[\tau_2/\alpha]}\ \textsc{whr:app}$$

$$\frac{\tau \rightsquigarrow \tau'}{\mathsf{Typerec}\ \tau\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_\times \rightsquigarrow \mathsf{Typerec}\ \tau'\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_\times}\ \textsc{whr:trec-con}$$

$$\frac{}{\mathsf{Typerec}\ (\mathsf{bool})\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_\times \rightsquigarrow \tau_{\mathsf{bool}}}\ \textsc{whr:trec-bool}$$

$$\frac{}{\begin{array}{l}\mathsf{Typerec}\ (\tau_1 \longrightarrow \tau_2)\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_\times \rightsquigarrow \\ \quad \tau_{\longrightarrow}\ \tau_1\ \tau_2\ (\mathsf{Typerec}\ \tau_1\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_\times) \\ \qquad\qquad (\mathsf{Typerec}\ \tau_2\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_\times)\end{array}}\ \textsc{whr:trec-arr}$$

$$\frac{}{\begin{array}{l}\mathsf{Typerec}\ (\tau_1 \times \tau_2)\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_\times \rightsquigarrow \\ \quad \tau_\times\ \tau_1\ \tau_2\ (\mathsf{Typerec}\ \tau_1\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_\times) \\ \qquad\qquad (\mathsf{Typerec}\ \tau_2\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_\times)\end{array}}\ \textsc{whr:trec-prod}$$

Definition A.13 (Term computation rules)**.**

$$\frac{}{(\lambda x{:}\sigma.e)v \rightsquigarrow e[v/x]}\ \textsc{ev:app} \qquad \frac{}{(\Lambda\alpha{:}\kappa.e)[\tau] \rightsquigarrow e[\tau/\alpha]}\ \textsc{ev:tapp}$$

$$\frac{}{\mathbf{fst}\,\langle v_1, v_2\rangle \rightsquigarrow v_1}\ \textsc{ev:fst} \qquad \frac{}{\mathbf{snd}\,\langle v_1, v_2\rangle \rightsquigarrow v_2}\ \textsc{ev:snd}$$

$$\frac{}{\mathbf{fix}\ x{:}\sigma.e \rightsquigarrow e[\mathbf{fix}\ x{:}\sigma.e/x]}\ \textsc{ev:fix} \qquad \frac{}{\mathbf{if\ true\ then}\ e_1\ \mathbf{else}\ e_2 \rightsquigarrow e_1}\ \textsc{ev:if1}$$

$$\frac{}{\mathbf{if\ false\ then}\ e_1\ \mathbf{else}\ e_2 \rightsquigarrow e_2}\ \textsc{ev:if2}$$

$$\frac{\tau \rightsquigarrow^* \mathsf{bool}}{\mathbf{typecase}\ [\gamma.\sigma]\ \tau\ e_{\mathsf{bool}}\ e_{\longrightarrow}\ e_{\times} \rightsquigarrow e_{\mathsf{int}}}\ \textsc{ev:tcase-bool}$$

$$\frac{\tau \rightsquigarrow^* \tau_1 \longrightarrow \tau_2}{\mathbf{typecase}\ [\gamma.\sigma]\ \tau\ e_{\mathsf{bool}}\ e_{\longrightarrow}\ e_{\times} \rightsquigarrow e_{\longrightarrow}[\tau_1][\tau_2]}\ \textsc{ev:tcase-arr}$$

$$\frac{\tau \rightsquigarrow^* \tau_1 \times \tau_2}{\mathbf{typecase}\ [\gamma.\sigma]\ \tau\ e_{\mathsf{bool}}\ e_{\longrightarrow}\ e_{\times} \rightsquigarrow e_{\times}[\tau_1][\tau_2]}\ \textsc{ev:tcase-prod}$$

Definition A.14 (Term congruence rules)**.**

$$\frac{e_1 \rightsquigarrow e_1'}{e_1 e_2 \rightsquigarrow e_1' e_2}\ \textsc{ev:app1} \qquad \frac{e_2 \rightsquigarrow e_2'}{v_1 e_2 \rightsquigarrow v_1 e_2'}\ \textsc{ev:app2} \qquad \frac{e_1 \rightsquigarrow e_1'}{\langle e_1, e_2\rangle \rightsquigarrow \langle e_1', e_2\rangle}\ \textsc{ev:pair1}$$

$$\frac{e_2 \rightsquigarrow e_2'}{\langle v_1, e_2\rangle \rightsquigarrow \langle v_1, e_2'\rangle}\ \textsc{ev:pair2} \qquad \frac{e \rightsquigarrow e'}{\mathbf{fst}\,e \rightsquigarrow \mathbf{fst}\,e'}\ \textsc{ev:fst-con}$$

$$\frac{e \rightsquigarrow e'}{\mathbf{snd}\,e \rightsquigarrow \mathbf{snd}\,e'}\ \textsc{ev:snd-con}$$

$$\frac{e_1 \rightsquigarrow e_1'}{\mathbf{if}\ e_1\ \mathbf{then}\ e_2\ \mathbf{else}\ e_3 \rightsquigarrow \mathbf{if}\ e_1'\ \mathbf{then}\ e_2\ \mathbf{else}\ e_3}\ \textsc{ev:if-con}$$

$$\frac{e \rightsquigarrow e'}{e[\tau] \rightsquigarrow e'[\tau]}\ \textsc{ev:tapp-con}$$

Definition A.15 (Nontermination)**.** *If* $\cdot \mid \cdot \vdash e : \sigma$ *and there does not exist a derivation* $e \rightsquigarrow^* v$ *then* $e \uparrow$*.*

APPENDIX B λ_{SECi} SOUNDNESS

Lemma B.1 (Inversion on sub-kinding)**.**

1. *If* $\star^\ell \leq \kappa$ *then* $\kappa = \star^{\ell'}$ *where* $\ell \sqsubseteq \ell'$*.*
2. *If* $\kappa_1 \xrightarrow{\ell} \kappa_2 \leq \kappa$ *then* $\kappa = \kappa_3 \xrightarrow{\ell'} \kappa_4$ *where* $\kappa_3 \leq \kappa_1$ *and* $\kappa_2 \leq \kappa_4$ *and* $\ell \sqsubseteq \ell'$*.*

Proof. Straightforward induction over the structure of the sub-kinding derivation. □

Lemma B.2 (Inversion for constructor well-formedness)**.**

1. *If* $\Delta \vdash \tau_1 \longrightarrow \tau_2 : \star^\ell$ *then* $\Delta \vdash \tau_1 : \star^{\ell_1}$ *and* $\Delta \vdash \tau_2 : \star^{\ell_2}$ *and* $\ell_1 \sqcup \ell_2 \sqsubseteq \ell$*.*
2. *If* $\Delta \vdash \tau_1 \times \tau_2 : \star^\ell$ *then* $\Delta \vdash \tau_1 : \star^{\ell_1}$ *and* $\Delta \vdash \tau_2 : \star^{\ell_2}$ *and* $\ell_1 \sqcup \ell_2 \sqsubseteq \ell$*.*
3. *If* $\Delta \vdash \tau_1\tau_2 : \kappa$ *then* $\Delta \vdash \tau_1 : \kappa_1 \xrightarrow{\ell} \kappa_2$ *and* $\Delta \vdash \tau_2 : \kappa_1$ *and* $\kappa_2 \sqcup \ell \leq \kappa$*.*
4. *If* $\Delta \vdash \lambda\alpha{:}\kappa.\tau : \kappa_1 \xrightarrow{\ell} \kappa_2$ *then* $\Delta, \alpha{:}\kappa \vdash \tau : \kappa_3$ *and* $\kappa_1 \leq \kappa$ *and* $\kappa_3 \leq \kappa_2$*.*
5. *If* $\Delta \vdash \mathsf{Typerec}\ \tau\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times} : \kappa$ *then* $\Delta \vdash \tau : \star^\ell$ *and* $\Delta \vdash \tau_{\mathsf{bool}} : \kappa'$ *and* $\Delta \vdash \tau_{\longrightarrow} : \star^\ell \xrightarrow{\ell'} \star^\ell \xrightarrow{\ell'} \kappa' \xrightarrow{\ell'} \kappa' \xrightarrow{\ell'} \kappa'$ *and* $\Delta \vdash \tau_{\times} : \star^\ell \xrightarrow{\ell'} \star^\ell \xrightarrow{\ell'} \kappa' \xrightarrow{\ell'} \kappa' \xrightarrow{\ell'} \kappa'$ *where* $\ell' = \mathcal{L}(\kappa')$ *and* $\kappa' \leq \kappa$*.*

Proof. By induction over the structure of the well-formedness derivation, making use of Lemma B.1. □

Lemma B.3 (Weak-head reduction equivalence)**.**

1. *If* $\Delta \vdash \tau : \kappa$ *and* $\tau \rightsquigarrow \tau'$ *then* $\Delta \vdash \tau = \tau' : \kappa$*.*
2. *If* $\Delta \vdash \tau : \kappa$ *and* $\tau \rightsquigarrow^* \tau'$ *then* $\Delta \vdash \tau = \tau' : \kappa$*.*
3. *If* $\Delta^\star \vdash \sigma$ *and* $\sigma \rightsquigarrow \sigma'$ *then* $\Delta^\star \vdash \sigma = \sigma'$*.*
4. *If* $\Delta^\star \vdash \sigma$ *and* $\sigma \rightsquigarrow^* \sigma'$ *then* $\Delta^\star \vdash \sigma = \sigma'$*.*

Proof. Part 1 follows from straightforward induction over the structure of $\tau \rightsquigarrow \tau'$ and use of Lemma B.2. Part 2 follows from Part 1 and induction on the number of reduction steps. Part 3 follows from straightforward induction over the structure of $\sigma \rightsquigarrow \sigma'$ using Part 1. Finally, Part 4 follows from Part 3 and induction on the number of reduction steps. □

Lemma B.4 (Inversion for type well-formedness)**.**
If $\Delta^\star \vdash (\tau)^\ell$ *then* $\Delta^\star \vdash \tau : \star^{\ell'}$.

Proof. Proof by induction over the structure of $\Delta^\star \vdash (\tau)^\ell$. □

Lemma B.5 (Inversion for subtyping)**.**

1. *If* $\Delta^\star \vdash \sigma_1 \xrightarrow{\ell_1} \sigma_2 \leq \sigma$ *then* $\Delta^\star \vdash \sigma \leq \sigma_3 \xrightarrow{\ell_2} \sigma_4$ *and* $\Delta^\star \vdash \sigma_3 \leq \sigma_1$ *and* $\Delta^\star \vdash \sigma_2 \leq \sigma_4$ *and* $\ell_1 \sqsubseteq \ell_2$.
2. *If* $\Delta^\star \vdash \sigma_1 \times^{\ell_1} \sigma_2 \leq \sigma$ *then* $\Delta^\star \vdash \sigma \leq \sigma_3 \times^{\ell_2} \sigma_4$ *and* $\Delta^\star \vdash \sigma_1 \leq \sigma_3$ *and* $\Delta^\star \vdash \sigma_2 \leq \sigma_4$ *and* $\ell_1 \sqsubseteq \ell_2$.
3. *If* $\Delta^\star \vdash \forall^{\ell_1} \alpha{:}\star^{\ell_2}.\sigma_1 \leq \sigma_2$ *then* $\Delta^\star \vdash \sigma_2 \leq \forall^{\ell_3} \alpha{:}\star^{\ell_4}.\sigma_3$ *and* $\Delta^\star, \alpha{:}\star^{\ell_4} \vdash \sigma_3 \leq \sigma_1$ *and* $\ell_1 \sqsubseteq \ell_3$ *and* $\ell_4 \sqsubseteq \ell_2$.

Proof. By straightforward induction over the structure of the subtyping derivation. □

Lemma B.6 (Inversion for typing)**.**

1. *If* $\Delta^\star \mid \Gamma \vdash \lambda x{:}\sigma_1.e : \sigma$ *then* $\Delta^\star \vdash \sigma \leq \sigma_2 \xrightarrow{\ell} \sigma_3$ *and* $\Delta^\star \mid \Gamma, x{:}\sigma_1 \vdash e : \sigma_4$ *where* $\Delta^\star \vdash \sigma_2 \leq \sigma_1$ *and* $\Delta^\star \vdash \sigma_4 \leq \sigma_3$.
2. *If* $\Delta^\star \mid \Gamma \vdash \Lambda\alpha{:}\star^{\ell}.e : \sigma$ *then* $\Delta^\star \vdash \sigma \leq \forall^{\ell_1} \alpha{:}\star^{\ell_2}.\sigma_1$ *and* $\Delta^\star, \alpha{:}\star^{\ell} \mid \Gamma \vdash e : \sigma_2$ *where* $\Delta^\star, \alpha{:}\star^{\ell_2} \vdash \sigma_2 \leq \sigma_1$ *and* $\ell_2 \sqsubseteq \ell$.
3. *If* $\Delta^\star \mid \Gamma \vdash \mathbf{fix}\, x{:}\sigma_1.e : \sigma_2$ *then* $\Delta^\star \mid \Gamma, x{:}\sigma_1 \vdash e : \sigma_1$ *where* $\Delta^\star \vdash \sigma_1 \leq \sigma_2$.
4. *If* $\Delta^\star \mid \Gamma \vdash \langle e_1, e_2 \rangle : \sigma$ *then* $\Delta^\star \vdash \sigma \leq \sigma_1 \times^{\ell} \sigma_2$ *and* $\Delta^\star \mid \Gamma \vdash e_1 : \sigma_3$ *and* $\Delta^\star \mid \Gamma \vdash e_2 : \sigma_4$ *where* $\Delta^\star \vdash \sigma_3 \leq \sigma_1$ *and* $\Delta^\star \vdash \sigma_4 \leq \sigma_2$.
5. *If* $\Delta^\star \mid \Gamma \vdash \mathbf{fst}\, e : \sigma$ *then* $\Delta^\star \mid \Gamma \vdash e : \sigma_1 \times^{\ell} \sigma_2$ *where* $\Delta^\star \vdash \sigma_1 \sqcup \ell \leq \sigma$.
6. *If* $\Delta^\star \mid \Gamma \vdash \mathbf{snd}\, e : \sigma$ *then* $\Delta^\star \mid \Gamma \vdash e : \sigma_1 \times^{\ell} \sigma_2$ *where* $\Delta^\star \vdash \sigma_2 \sqcup \ell \leq \sigma$.
7. *If* $\Delta^\star \mid \Gamma \vdash e_1 e_2 : \sigma_1$ *then* $\Delta^\star \mid \Gamma \vdash e_1 : \sigma_2 \xrightarrow{\ell} \sigma_3$ *and* $\Delta^\star \mid \Gamma \vdash e_2 : \sigma_2$ *and* $\Delta^\star \vdash \sigma_3 \sqcup \ell \leq \sigma_1$.

8. *If* $\Delta^\star \mid \Gamma \vdash e[\tau] : \sigma$ *then* $\Delta^\star \mid \Gamma \vdash e : \forall^{\ell_1}\alpha{:}\star^{\ell_2}.\sigma'$ *and* $\Delta^\star \vdash \tau : \star^{\ell_2}$ *and* $\Delta^\star \vdash \sigma'[\tau/\alpha] \sqcup \ell_1 \leq \sigma$.

9. *If* $\Delta^\star \mid \Gamma \vdash \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 : \sigma$ *then* $\Delta^\star \mid \Gamma \vdash e_1 : (\mathsf{bool})^\ell$ *and* $\Delta^\star \mid \Gamma \vdash e_2 : \sigma'$ *and* $\Delta^\star \mid \Gamma \vdash e_3 : \sigma'$ *where* $\Delta^\star \vdash \sigma' \sqcup \ell \leq \sigma$.

10. *If* $\Delta^\star \mid \Gamma \vdash \textbf{typecase } [\gamma.\sigma]\ \tau\ e_{\mathsf{bool}}\ e_{\longrightarrow}\ e_{\times} : \sigma'$ *then*
$\Delta^\star \vdash \tau : \star^\ell$ *and*
$\Delta^\star, \gamma{:}\star^\ell \vdash \sigma$ *and*
$\Delta^\star \mid \Gamma \vdash e_{\mathsf{bool}} : \sigma[\mathsf{bool}/\gamma]$ *and*
$\Delta^\star \mid \Gamma \vdash e_{\longrightarrow} : \forall^{\ell'}\alpha{:}\star^\ell.\forall^{\ell'}\beta{:}\star^\ell.\sigma[\alpha \longrightarrow \beta/\gamma]$ *and*
$\Delta^\star \mid \Gamma \vdash e_{\times} : \forall^{\ell'}\alpha{:}\star^\ell.\forall^{\ell'}\beta{:}\star^\ell.\sigma[\alpha \times \beta/\gamma]$ *where*
$\ell' = \mathcal{L}(\sigma[\tau/\gamma])$ *and*
$\ell \sqsubseteq \ell'$ *and*
$\Delta^\star \vdash \sigma[\tau/\gamma] \leq \sigma'$.

Proof. By straightforward induction on the structure of the typing derivation with uses of Lemma B.5. □

Lemma B.7 (Substitution for constructors)**.** *If* $\Delta, \alpha{:}\kappa_1 \vdash \tau_1 : \kappa_2$ *and* $\Delta \vdash \tau_2 : \kappa_1$ *then* $\Delta \vdash \tau_1[\tau_2/\alpha] : \kappa_2$.

Proof. By straightforward induction over the structure of $\Delta, \alpha{:}\kappa_1 \vdash \tau_1 : \kappa_2$. □

Lemma B.8 (Substitution for equivalence)**.** *If* $\Delta, \alpha{:}\kappa_1 \vdash \tau_1 = \tau_2 : \kappa_2$ *and* $\Delta \vdash \tau : \kappa_1$ *then* $\Delta \vdash \tau_1[\tau/\alpha] = \tau_2[\tau/\alpha] : \kappa_2$.

Proof. By straightforward induction over the structure of $\Delta, \alpha{:}\kappa_1 \vdash \tau_1 = \tau_2 : \kappa_2$, making use of Lemma B.7. □

Lemma B.9 (Substitution for types)**.**

1. *If* $\Delta^\star, \alpha{:}\star^\ell \vdash \sigma_1 \leq \sigma_2$ *and* $\Delta^\star \vdash \tau : \star^\ell$ *then* $\Delta^\star \vdash \sigma_1[\tau/\alpha] \leq \sigma_2[\tau/\alpha]$.

2. *If* $\Delta^\star, \alpha{:}\star^\ell \vdash \sigma$ *and* $\Delta^\star \vdash \tau : \star^\ell$ *then* $\Delta^\star \vdash \sigma[\tau/\alpha]$.

Proof. By mutual induction over the structure of $\Delta, \alpha{:}\star^\ell \vdash \sigma_1 \leq \sigma_2$ and $\Delta, \alpha{:}\star^\ell \vdash \sigma$, using Lemmas B.7 and B.8. □

Lemma B.10 (Substitution commutes with equivalence)**.**

1. *If* $\Delta \vdash \tau_1 = \tau_2 : \kappa_1$ *and* $\Delta, \alpha{:}\kappa_1 \vdash \tau : \kappa_2$ *then* $\Delta \vdash \tau[\tau_1/\alpha] = \tau[\tau_2/\alpha] : \kappa_2$.

2. *If* $\Delta \vdash \tau_1 = \tau_2 : \star^\ell$ *and* $\Delta, \alpha{:}\star^\ell \vdash \sigma$ *then* $\Delta \vdash \sigma[\tau_1/\alpha] \leq \sigma[\tau_2/\alpha]$ *and* $\Delta \vdash \sigma[\tau_2/\alpha] \leq \sigma[\tau_1/\alpha]$.

Proof. Part 1 follows from induction over the structure of $\Delta, \alpha{:}\kappa_1 \vdash \tau : \kappa_2$. Part 2 follows from induction over the structure of $\Delta, \alpha{:}\star^\ell \vdash \sigma$ making use of Part 1. □

Lemma B.11 (Substitution for terms)**.**

1. *If* $\Delta^\star, \alpha{:}\star^\ell \mid \Gamma \vdash e : \sigma$ *and* $\Delta^\star \vdash \tau : \star^\ell$ *then* $\Delta^\star \mid \Gamma \vdash e[\tau/\alpha] : \sigma[\tau/\alpha]$.
2. *If* $\Delta^\star \mid \Gamma, x{:}\sigma_1 \vdash e : \sigma_2$ *and* $\Delta^\star \mid \Gamma \vdash e' : \sigma_1$ *then* $\Delta^\star \mid \Gamma \vdash e[e'/x] : \sigma_2$.

Proof. By straightforward induction over the typing derivations, using Lemmas B.7 and B.9. □

Lemma B.12 (Subject reduction)**.**

1. *If* $\Delta \vdash \tau : \kappa$ *and* $\tau \rightsquigarrow \tau'$ *then* $\Delta \vdash \tau' : \kappa$.
2. *If* $\Delta \vdash \tau : \kappa$ *and* $\tau \rightsquigarrow^* \tau'$ *then* $\Delta \vdash \tau' : \kappa$.
3. *If* $\Delta^\star \vdash \sigma$ *and* $\sigma \rightsquigarrow \sigma'$ *then* $\Delta^\star \vdash \sigma'$.
4. *If* $\Delta^\star \vdash \sigma$ *and* $\sigma \rightsquigarrow^* \sigma'$ *then* $\Delta^\star \vdash \sigma'$.
5. *If* $\Delta^\star \mid \Gamma \vdash e : \sigma$ *and* $e \rightsquigarrow e'$ *then* $\Delta^\star \mid \Gamma \vdash e' : \sigma$.

Proof. Part 1 follows by induction over the structure of $\tau \rightsquigarrow \tau'$ making use of Lemmas B.2 and B.7. Part 2 is a direct corollary of Part 1. Part 3 follows by induction over the structure of $\sigma \rightsquigarrow \sigma'$ making use of Lemma B.4 and Part 1. Part 4 is a direct corollary of Part 3. Part 5 follows by induction over the structure of $e \rightsquigarrow e'$ making use of Lemmas B.6, B.2, B.3, and B.10. □

Lemma B.13 (Weak head reduction terminates)**.**

1. *If* $\cdot \vdash \tau : \kappa$ *then* $\tau \rightsquigarrow^* \nu$.
2. *If* $\Delta^\star \vdash \sigma$ *then* $\sigma \rightsquigarrow^* \zeta$.

Proof. Follows from a standard logical relations proof that we omit here. See Morrisett's thesis [Mor95]. □

Lemma B.14 (Canonical forms for constructors)**.** *If* $\cdot \vdash \nu : \kappa$

1. $\kappa = \star^{\ell}$ *then* $\nu = \mathsf{bool}$ *or* $\nu = \tau_1 \longrightarrow \tau_2$ *or* $\nu = \tau_1 \times \tau_2$.
2. $\kappa = \kappa_1 \xrightarrow{\ell} \kappa_2$ *then* $\nu = \lambda\alpha{:}\kappa_3.\tau$ *where* $\kappa_1 \leq \kappa_3$.

Proof. By straightforward induction over the structure of $\Delta \vdash \nu : \kappa$. □

Lemma B.15 (Canonical forms for terms)**.** *If* $\cdot \mid \cdot \vdash \nu : \sigma$

1. $\sigma = \mathsf{bool}$ *then* $\nu =$ **true** *or* $\nu =$ **false**.
2. $\sigma = \sigma_1 \xrightarrow{\ell'} \sigma_2$ *then* $\nu = \lambda x{:}\sigma_3.e$ *where* $\Delta^\star \vdash \sigma_1 \leq \sigma_3$.
3. $\sigma = \forall^{\ell_1}\alpha{:}\star^{\ell_2}.\sigma'$ *then* $\nu = \Lambda\alpha{:}\star^{\ell_3}.e$ *where* $\ell_1 \sqsubseteq \ell_3$.
4. $\sigma = \sigma_1 \times^{\ell} \sigma_2$ *then* $\nu = \langle \nu_1, \nu_2 \rangle$.

Proof. By straightforward induction over the structure of $\cdot \mid \cdot \vdash \nu : \sigma$. □

Lemma B.16 (Progress)**.** *If* $\cdot \mid \cdot \vdash e : \sigma$ *then* e *is a value or there exists a derivation* $e \rightsquigarrow e'$.

Proof. By straightforward induction over the structure of $\cdot \mid \cdot \vdash e : \sigma$, using Lemmas B.15, B.13, and B.14. □

Theorem B.17 (Type safety)**.** *If* $\cdot \mid \cdot \vdash e : \sigma$ *then there exists a derivation that* $e \rightsquigarrow^* \nu$ *or* $e \uparrow$.

Proof. Proof by contradiction using Lemmas B.12 and B.16. □

APPENDIX C λ_{SECi} WITH FINITE UNWINDINGS

Definition C.1 (Extension for finite unwindings)**.**

terms
$e ::= \ldots$
$\quad \mid \mathbf{fix}_n\, x{:}\sigma.e$ *finite fix-point*

Definition C.2 (Term well-formedness)**.**

$$\frac{\Delta^\star \mid \Gamma, x{:}\sigma \vdash e : \sigma \qquad \Delta^\star \vdash \sigma}{\Delta^\star \mid \Gamma \vdash \mathbf{fix}_n\, x{:}\sigma.e : \sigma}\ \textsc{wft:fixn}$$

Definition C.3 (Computation rules)**.**

$$\frac{}{\mathbf{fix}_0\ x{:}\sigma.e \rightsquigarrow \mathbf{fix}_0\ x{:}\sigma.e}\ \textsc{ev:fix0} \qquad \frac{}{\mathbf{fix}_{n+1}\ x{:}\sigma.e \rightsquigarrow e[\mathbf{fix}_n\ x{:}\sigma.e/x]}\ \textsc{ev:fixn}$$

Lemma C.4 ($\mathbf{fix}_0$ always diverges)**.** $\mathbf{fix}_0\ x{:}\sigma.e \uparrow$.

Proof. Proof by contradiction, assuming there exists a derivation $\mathbf{fix}_0\ x{:}\sigma.e \rightsquigarrow^* v$. □

Lemma C.5 (Unwinding type equivalences)**.**

$$\Delta^\star \mid \Gamma \vdash \mathbf{fix}\ x{:}\sigma.e : \sigma \quad \textit{iff} \quad \Delta^\star \mid \Gamma \vdash \mathbf{fix}_n\ x{:}\sigma.e : \sigma$$

Proof. Trivial inversion upon the typing derivation in both directions. □

Lemma C.6 (Unwinding evaluation equivalence)**.**

$$\mathbf{fix}\ x{:}\sigma.e' \rightsquigarrow^* v \quad \textit{iff} \quad \textit{exists } n \textit{ such that } \mathbf{fix}_n\ x{:}\sigma.e' \rightsquigarrow^* v$$

Proof. Both directions follow by straightforward induction over number or reduction steps. □

Lemma C.7 (Bound can be increased)**.** *If* $\mathbf{fix}_n\ x{:}\sigma.e' \rightsquigarrow^* v$ *then* $\mathbf{fix}_m\ x{:}\sigma.e' \rightsquigarrow^* v$ *for* $m \geq n$.

Proof. Straightforward induction over the number of reduction steps in the derivation $\mathbf{fix}_n\ x{:}\sigma.e' \rightsquigarrow^* v$. □

APPENDIX D λ_{SECi} NONINTERFERENCE

Definition D.1 (Relations between values)**.** *We define* $\sigma_1 \leftrightarrow \sigma_2$ *to be the set of all binary relations between values of type* σ_1 *and values of type* σ_2.

Definition D.2 (Parameterized relation)**.** *A parameterized relation* R *is a function that when given a label* ℓ *and a type context* ρ *yields a binary relation between values of two types. For conciseness, we use the notation* R^ℓ_ρ *for the application of a label and a type context to a parameterized relation.*

We will sometimes abuse notation and write $R^\ell_\rho \in \delta_1(\rho\{\tau_1\}) \leftrightarrow \delta_2(\rho\{\tau_1\})$. *This can be roughly understood with dependent types as* $R : \Pi\ell.\Pi\rho.\delta_1(\rho\{\tau_1\}) \leftrightarrow \delta_2(\rho\{\tau_1\})$.

Definition D.3 (Parameterized relation consistency)**.** *We say that a parameterized relation* $R^\ell_\rho \in \sigma_1 \leftrightarrow \sigma_2$ *is consistent if* $v_1 R^{\ell_1}_\rho v_2$ *and* $\ell_1 \sqsubseteq \ell_2$ *then* $v_2 R^{\ell_2}_\rho v_2$.

Definition D.4 (Security logical relation for constructors)**.**

$$\frac{\ell_1 \not\sqsubseteq \ell_0}{v_1 \sim_{\ell_0} v_2 : \star^{\ell_1}}\ \textsc{tslr:type-opaq} \qquad \frac{\ell_1 \sqsubseteq \ell_0}{\mathsf{bool} \sim_{\ell_0} \mathsf{bool} : \star^{\ell_1}}\ \textsc{tslr:type-bool}$$

$$\frac{\ell_3 \sqsubseteq \ell_0 \qquad \begin{array}{c}\ell_1 \sqcup \ell_2 \sqsubseteq \ell_3 \\ \tau_1 \approx_{\ell_0} \tau_3 : \star^{\ell_1} \qquad \tau_2 \approx_{\ell_0} \tau_4 : \star^{\ell_2}\end{array}}{\tau_1 \longrightarrow \tau_2 \sim_{\ell_0} \tau_3 \longrightarrow \tau_4 : \star^{\ell_3}}\ \textsc{tslr:type-arr}$$

$$\frac{\ell_1 \sqcup \ell_2 \sqsubseteq \ell_3 \qquad \ell_3 \sqsubseteq \ell_0 \qquad \tau_1 \approx_{\ell_0} \tau_3 : \star^{\ell_1} \qquad \tau_2 \approx_{\ell_0} \tau_4 : \star^{\ell_2}}{\tau_1 \times \tau_2 \sim_{\ell_0} \tau_3 \times \tau_4 : \star^{\ell_3}}\ \textsc{tslr:type-prod}$$

$$\frac{\forall(\tau_1 \approx_{\ell_0} \tau_2 : \kappa_1).v_1\tau_1 \approx_{\ell_0} v_2\tau_2 : \kappa_2 \sqcup \ell_1}{v_1 \sim_{\ell_0} v_2 : \kappa_1 \xrightarrow{\ell_1} \kappa_2}\ \textsc{tslr:arr}$$

$$\frac{\tau_1 \rightsquigarrow^* v_1 \qquad \tau_2 \rightsquigarrow^* v_2 \qquad v_1 \sim_{\ell_0} v_2 : \kappa}{\tau_1 \approx_{\ell_0} \tau_2 : \kappa}\ \textsc{tsclr:base}$$

We implicitly require for $v_1 \sim_\ell v_2 : \kappa$ *and* $\tau_1 \approx_\ell \tau_2 : \kappa$ *that* $\cdot \vdash v_1, v_2 : \kappa$ *and* $\cdot \vdash \tau_1, \tau_2 : \kappa$ *respectively.*

Definition D.5 (Type reduction)**.**

$$\frac{\tau \rightsquigarrow \tau'}{(\tau)^\ell \rightsquigarrow (\tau')^\ell}\ \textsc{whr:inj-tc} \qquad \frac{}{(\tau_1 \longrightarrow \tau_2)^\ell \rightsquigarrow (\tau_1)^\ell \xrightarrow{\ell} (\tau_2)^\ell}\ \textsc{whr:inj-arr}$$

$$\frac{}{(\tau_1 \times \tau_2)^\ell \rightsquigarrow (\tau_1)^\ell \times^\ell (\tau_2)^\ell}\ \textsc{whr:inj-prod}$$

Definition D.6 (Security logical relation for terms)**.**

$$\frac{\alpha \mapsto \mathsf{R} \in \eta \qquad \ell_1 \sqsubseteq \ell_0 \implies v_1 \mathsf{R}_\rho^{\ell_1} v_2}{\eta \vdash v_1 \sim_{\ell_0} v_2 : (\rho\{\alpha\})^{\ell_1}}\ \textsc{slr:con}$$

$$\frac{\ell_1 \sqsubseteq \ell_0 \implies v_1 = v_2}{\eta \vdash v_1 \sim_{\ell_0} v_2 : (\mathsf{bool})^{\ell_1}}\ \textsc{slr:bool}$$

$$\frac{\forall(\eta \vdash e_1 \approx_{\ell_0} e_2 : \sigma_1).\eta \vdash v_1 e_1 \approx_{\ell_0} v_2 e_2 : \sigma_2 \sqcup \ell_1}{\eta \vdash v_1 \sim_{\ell_0} v_2 : \sigma_1 \xrightarrow{\ell_1} \sigma_2}\ \textsc{slr:arr}$$

$$\frac{\eta \vdash \mathbf{fst}\, v_1 \approx_{\ell_0} \mathbf{fst}\, v_2 : \sigma_1 \sqcup \ell_1 \qquad \eta \vdash \mathbf{snd}\, v_1 \approx_{\ell_0} \mathbf{snd}\, v_2 : \sigma_2 \sqcup \ell_1}{\eta \vdash v_1 \sim_{\ell_0} v_2 : \sigma_1 \times^{\ell_1} \sigma_2}\ \text{SLR:PROD}$$

$$\frac{\begin{array}{c}\forall(\tau_1 \approx_{\ell_0} \tau_2 : \star^{\ell_2}).\forall(R_\rho^{\ell_2} \in \delta_1((\rho\{\tau_1\})^{\ell_2}) \leftrightarrow \delta_2((\rho\{\tau_2\})^{\ell_2})). \\ \eta, \alpha \mapsto R \vdash v_1[\tau_1] \approx_{\ell_0} v_2[\tau_2] : \sigma \sqcup \ell_1 \qquad R \text{ consistent}\end{array}}{\eta \vdash v_1 \sim_{\ell_0} v_2 : \forall^{\ell_1} \alpha{:}\star^{\ell_2}.\sigma}\ \text{SLR:ALL}$$

$$\frac{e_1 \rightsquigarrow^* v_1 \qquad e_2 \rightsquigarrow^* v_2 \qquad \sigma \rightsquigarrow^* \zeta \qquad \eta \vdash v_1 \sim_{\ell_0} v_2 : \zeta}{\eta \vdash e_1 \approx_{\ell_0} e_2 : \sigma}\ \text{SCLR:TERM}$$

$$\frac{(e_1 \uparrow) \vee (e_2 \uparrow)}{\eta \vdash e_1 \approx_{\ell_0} e_2 : \sigma}\ \text{SCLR:DIVR}$$

We implicitly require for $\eta \vdash v_1 \sim_{\ell_0} v_2 : \zeta$ *and* $\eta \vdash e_1 \approx_{\ell_0} e_2 : \sigma$ *that* $\cdot \mid \cdot \vdash v_1 : \delta_1(\zeta)$, $\cdot \mid \cdot \vdash v_2 : \delta_2(\zeta)$ *and* $\cdot \mid \cdot \vdash e_1 : \delta_1(\sigma)$, $\cdot \mid \cdot \vdash e_2 : \delta_2(\sigma)$ *respectively where* $\delta_1 \approx_{\ell_0} \delta_2 : \Delta^\star$ *and* $\eta \vdash \Delta^\star$.

Definition D.7 (Related constructor substitutions)**.**

$$\frac{\forall \alpha{:}\kappa \in \Delta.(\delta_1(\alpha) \approx_{\ell_0} \delta_2(\alpha) : \kappa)}{\delta_1 \approx_{\ell_0} \delta_2 : \Delta}\ \text{TSSLR:BASE}$$

Definition D.8 (Relation mapping regularity)**.** *If* $\delta_1 \approx_{\ell_0} \delta_2 : \Delta^\star$ *then*

$$\frac{\begin{array}{c}\forall \alpha{:}\star^{\ell_1} \in \Delta^\star.(\eta(\alpha)_\rho^{\ell_1} \in \delta_1((\rho\{\alpha\})^{\ell_1}) \leftrightarrow \delta_2((\rho\{\alpha\})^{\ell_1})) \\ \eta(\alpha) \text{ consistent}\end{array}}{\eta \vdash \Delta^\star}\ \text{RELM:REG}$$

Definition D.9 (Related term substitutions)**.** *If* $\delta_1 \approx_{\ell_0} \delta_2 : \Delta^\star$ *and* $\eta \vdash \Delta^\star$ *then*

$$\frac{\forall x{:}\sigma \in \Gamma.(\eta \vdash \gamma_1(x) \approx_{\ell_0} \gamma_2(x) : \sigma)}{\eta \vdash \gamma_1 \approx_{\ell_0} \gamma_2 : \Gamma}\ \text{SSLR:BASE}$$

Lemma D.10 (Logical relations closed under reduction)**.**

1. $\tau_1 \approx_{\ell_0} \tau_2 : \kappa$ *iff* $\tau_1 \rightsquigarrow^* \tau_1'$ *and* $\tau_2 \rightsquigarrow^* \tau_2'$ *and* $\tau_1' \approx_{\ell_0} \tau_2' : \kappa$.
2. $\eta \vdash e_1 \approx_{\ell_0} e_2 : \sigma$ *iff* $e_1 \rightsquigarrow^* e_1'$ *and* $e_2 \rightsquigarrow^* e_2'$ *and* $\sigma \rightsquigarrow^* \sigma'$ *and* $\eta \vdash e_1' \approx_{\ell_0} e_2' : \sigma'$.

Proof. Follows from straightforward inversion upon the logical relations and from the properties of reduction. □

Lemma D.11 (Inversion for subtyping on normal types)**.**

1. *If* $\Delta^\star \vdash (\rho\{\alpha\})^{\ell_1} \leq \zeta$ *then* $\zeta = (\rho\{\alpha\})^{\ell_2}$ *where* $\ell_1 \sqsubseteq \ell_2$.
2. *If* $\Delta^\star \vdash (\mathsf{bool})^{\ell_1} \leq \zeta$ *then* $\zeta = (\mathsf{bool})^{\ell_2}$ *where* $\ell_1 \sqsubseteq \ell_2$.

Proof. By straightforward induction over the structure of the subtyping derivations. □

Lemma D.12 (Logical relations closed under subsumption)**.**

1. *If* $\kappa_1 \leq \kappa_2$ *and*
   - $v_1 \sim_{\ell_0} v_2 : \kappa_1$ *then* $\tau_1 \sim_{\ell_0} \tau_2 : \kappa_2$.
   - $\tau_1 \approx_{\ell_0} \tau_2 : \kappa_1$ *then* $\tau_1 \approx_{\ell_0} \tau_2 : \kappa_2$
2. *If* $\eta \vdash \Delta^\star$ *and*
   - $\Delta^\star \vdash \zeta_1 \leq \zeta_2$ *and* $\eta \vdash v_1 \sim_{\ell_0} v_2 : \zeta_1$ *then* $\eta \vdash v_1 \sim_{\ell_0} v_2 : \zeta_2$.
   - $\Delta^\star \vdash \sigma_1 \leq \sigma_2$ *and* $\eta \vdash e_1 \approx_{\ell_0} e_2 : \sigma_1$ *then* $\eta \vdash e_1 \approx_{\ell_0} e_2 : \sigma_2$

Proof. Part 1 follows from straightforward mutual induction over κ_1. Part 2 follows from straightforward mutual induction over σ_1 and ζ_1, with uses of Part 1, Definition D.3, and Lemmas B.5 and D.11. □

Corollary D.13 (Term and value relation is consistent)**.** *The relations* $\{(v_1, v_2) \mid \eta \vdash v_1 \sim_{\ell_0} v_2 : \zeta\}$ *and* $\{(e_1, e_2) \mid \eta \vdash e_1 \approx_{\ell_0} e_2 : \zeta\}$ *are consistent.*

Proof. A direct consequence of Definition D.3 and Lemma D.12 Part 2. □

Lemma D.14 (Obliviousness)**.**

1. *If* $\cdot \vdash \tau_1, \tau_2 : \kappa$ *and* $\mathcal{L}(\kappa) \not\sqsubseteq \ell_0$ *then* $\tau_1 \approx_{\ell_0} \tau_2 : \kappa$.
2. *If* $\eta \vdash \Delta^\star$ *and* $\delta_1 \approx_{\ell_0} \delta_2 : \Delta^\star$ *and* $\mathcal{L}(\zeta) \not\sqsubseteq \ell_0$ *and*
   - $\Delta^\star \mid \cdot \vdash v_1, v_2 : \zeta$ *then* $\eta \vdash \delta_1(v_1) \sim_{\ell_0} \delta_2(v_2) : \zeta$.
   - $\Delta^\star \mid \cdot \vdash e_1, e_2 : \sigma$ *then* $\eta \vdash \delta_1(e_1) \approx_{\ell_0} \delta_2(e_2) : \sigma$.

Proof. Part 1 follows from the use of Lemma B.13 and straightforward induction upon κ. Part 2 follows from Theorem B.17 and induction upon ζ. □

Lemma D.15 (Constructor substitution for term relations). *If* $\eta \vdash \Delta^\star$ *and* $\mathsf{R}^\ell_\rho = \{(\nu_1, \nu_2) \mid \eta \vdash \nu_1 \sim_{\ell_0} \nu_2 : \zeta_2\}$ *and* $\delta_i(\alpha) = \delta_i(\tau)$ *then*

1. $\eta, \alpha \mapsto \mathsf{R} \vdash \nu_1 \sim_{\ell_0} \nu_2 : \zeta_1$ *and* $(\rho\{\tau\})^\ell \rightsquigarrow^* \zeta_2$ *iff* $\eta \vdash \nu_1 \sim_{\ell_0} \nu_2 : \zeta_3$ *where* $\zeta[\tau/\alpha] \rightsquigarrow^* \zeta_3$.

2. $\eta, \alpha \mapsto \mathsf{R} \vdash e_1 \approx_{\ell_0} e_2 : \sigma$ *and* $(\rho\{\tau\})^\ell \rightsquigarrow^* \zeta$ *iff* $\eta \vdash e_1 \approx_{\ell_0} e_2 : \sigma[\tau/\alpha]$.

Proof. Follows from mutual induction over the logical relations, making use of Lemma D.14 Part 2 and Corollary D.13. □

Lemma D.16 (Constructor relation closed under Typerec). *If* $\tau \approx_{\ell_0} \tau' : \star^\ell$ *and*

- $\tau_{\mathsf{bool}} \approx_{\ell_0} \tau'_{\mathsf{bool}} : \kappa$ *and*
- $\tau_{\longrightarrow} \approx_{\ell_0} \tau'_{\longrightarrow} : \star^\ell \xrightarrow{\ell'} \star^\ell \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa$ *and*
- $\tau_\times \approx_{\ell_0} \tau'_\times : \star^\ell \xrightarrow{\ell'} \star^\ell \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa$.

where $\ell' = \mathcal{L}(\kappa)$ *then* $\mathsf{Typerec}\ \tau\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_\times \approx_{\ell_0} \mathsf{Typerec}\ \tau'\ \tau'_{\mathsf{bool}}\ \tau'_{\longrightarrow}\ \tau'_\times : \kappa$.

Proof. Straightforward induction over the structure of $\tau \approx_{\ell_1} \tau' : \star^\ell$ making use of Lemma D.14 Part 1. □

Lemma D.17 (Fixpoint continuity). *If for all* n, $\eta \vdash \mathbf{fix}_n\ x{:}\sigma_1.e_1 \approx_{\ell_0} \mathbf{fix}_n\ x{:}\sigma_2.e_2 : \sigma$ *then* $\eta \vdash \mathbf{fix}\ x{:}\sigma_1.e_1 \approx_{\ell_0} \mathbf{fix}\ x{:}\sigma_2.e_2 : \sigma$ *where* $\delta_i(\sigma) = \sigma_i$.

Proof. By substitution we know that $\cdot \mid \cdot \vdash \mathbf{fix}\ x{:}\sigma_i.e_i : \sigma_i$. Using Theorem B.17 we know that either $\mathbf{fix}\ x{:}\sigma_i.e_i \rightsquigarrow^* \nu_i$ or $\mathbf{fix}\ x{:}\sigma_i.e_i \uparrow$.

Case For both $i = 1$ and $i = 2$, $\mathbf{fix}\ x{:}\sigma_i.e_i \rightsquigarrow^* \nu_i$

- From Lemma C.6 we know that $\mathbf{fix}_n\ x{:}\sigma_1.e_1 \rightsquigarrow^* \nu_1$ and $\mathbf{fix}_m\ x{:}\sigma_2.e_2 \rightsquigarrow^* \nu_2$.
- There exists some $p > m$ and $p > n$. By Lemma C.7 we can conclude $\mathbf{fix}_p\ x{:}\sigma_i.e_i \rightsquigarrow^* \nu_i$.
- Instantiating for all n,$\eta \vdash \mathbf{fix}_n\ x{:}\sigma_1.e_1 \approx_{\ell_0} \mathbf{fix}_n\ x{:}\sigma_2.e_2 : \sigma$ with p we have that $\eta \vdash \mathbf{fix}_p\ x{:}\sigma_1.e_1 \approx_{\ell_0} \mathbf{fix}_p\ x{:}\sigma_2.e_2 : \sigma$.

- By inversion upon $\eta \vdash \mathbf{fix}_p\ x{:}\sigma_1.e_1 \approx_{\ell_0} \mathbf{fix}_p\ x{:}\sigma_2.e_2 : \sigma$. we know that either $\mathbf{fix}_p\ x{:}\sigma_i.e_i \rightsquigarrow^* v_i'$ or $\mathbf{fix}_p\ x{:}\sigma_i.e_i \uparrow v_i'$. However, we already have that $\mathbf{fix}_p\ x{:}\sigma_i.e_i \rightsquigarrow^* v_i$. Therefore, we also know by inversion that $\eta \vdash v_1 \sim_{\ell_0} v_2 : \zeta$ for $\sigma \rightsquigarrow^* \zeta$.
- Given that $\mathbf{fix}\ x{:}\sigma_i.e_i \rightsquigarrow^* v_i$ by SCLR:TERM we can conclude that $\eta \vdash \mathbf{fix}\ x{:}\sigma_1.e_1 \approx_{\ell_0} \mathbf{fix}\ x{:}\sigma_2.e_2 : \sigma$.

Case For $i = 1$ or $i = 2$, $\mathbf{fix}\ x{:}\sigma_i.e_i \uparrow$

- Follows directly from SCLR:DIVR.

□

Theorem D.18 (Substitution)**.**

1. *If* $\Delta \vdash \tau : \kappa$ *and* $\delta_1 \approx_{\ell_0} \delta_2 : \Delta$ *then* $\delta_1(\tau) \approx_{\ell_0} \delta_2(\tau) : \kappa$.
2. *If* $\Delta^\star \mid \Gamma \vdash e : \sigma$ *and* $\delta_1 \approx_{\ell_0} \delta_2 : \Delta^\star$ *and* $\eta \vdash \Delta^\star$ *and* $\eta \vdash \gamma_1 \approx_{\ell_0} \gamma_2 : \Gamma$ *then* $\eta \vdash \delta_1(\gamma_1(e)) \approx_{\ell_0} \delta_2(\gamma_2(e)) : \sigma$.

Proof. Part 1 follows by induction over the structure of $\Delta \vdash \tau : \kappa$.

Case

$$\frac{\alpha{:}\kappa \in \Delta}{\Delta \vdash \alpha : \kappa}\ \text{WFC:VAR}$$

- Immediate by inversion upon $\delta_1 \approx_{\ell_0} \delta_2 : \Delta$.

Case

$$\frac{}{\Delta \vdash \mathsf{bool} : \star^\bot}\ \text{WFC:BOOL}$$

- By the definition of substitution $\delta_i(\mathsf{bool}) = \mathsf{bool}$, and $\mathsf{bool} \rightsquigarrow^* \mathsf{bool}$ by TRC:REFL, therefore $\delta_i(\mathsf{bool}) \rightsquigarrow^* \delta_i(\mathsf{bool})$.
- $\bot \sqsubseteq \ell_0$ for any ℓ_0 so it follows trivially from TSLR:TYPE-BOOL that $\mathsf{bool} \sim_{\ell_0} \mathsf{bool} : \star^\bot$.
- By TSCLR:BASE on $\mathsf{bool} \sim_{\ell_0} \mathsf{bool} : \star^\bot$ and $\delta_i(\mathsf{bool}) \rightsquigarrow^* \delta_i(\mathsf{bool})$ we can conclude that $\mathsf{bool} \approx_{\ell_0} \mathsf{bool} : \star^\bot$.

Case

$$\frac{\Delta \vdash \tau_1 : \star^{\ell_1} \qquad \Delta \vdash \tau_2 : \star^{\ell_2}}{\Delta \vdash \tau_1 \longrightarrow \tau_2 : \star^{\ell_1 \sqcup \ell_2}}\ \text{WFC:ARR}$$

- By the definition of substitution $\delta_i(\tau_1 \longrightarrow \tau_2) = \delta_i(\tau_1) \longrightarrow \delta_i(\tau_2)$, and $\delta_i(\tau_1) \longrightarrow \delta_i(\tau_2) \rightsquigarrow^* \delta_i(\tau_1) \longrightarrow \delta_i(\tau_2)$ by TRC:REFL, therefore $\delta_i(\tau_1 \longrightarrow \tau_2) \rightsquigarrow^* \delta_i(\tau_1 \longrightarrow \tau_2)$.
- Lattice joins and order are decidable, so either $\ell_1 \sqcup \ell_2 \sqsubseteq \ell_0$ or $\ell_1 \sqcup \ell_2 \not\sqsubseteq \ell_0$.

  Sub-Case $\ell_1 \sqcup \ell_2 \sqsubseteq \ell_0$.
  - Appeal to the induction hypothesis on $\Delta \vdash \tau_1 : \star^{\ell_1}$ and $\Delta \vdash \tau_2 : \star^{\ell_2}$ with $\delta_1 \approx_{\ell_0} \delta_2 : \Delta$ yielding $\delta_1(\tau_1) \approx_{\ell_0} \delta_2(\tau_1) : \star^{\ell_1}$ and $\delta_1(\tau_2) \approx_{\ell_0} \delta_2(\tau_2) : \star^{\ell_2}$.
  - Using TSLR:TYPE-ARR on these along with $\ell_1 \sqcup \ell_2 \sqsubseteq \ell_1 \sqcup \ell_2$ (by reflexivity) and $\ell_1 \sqcup \ell_2 \sqsubseteq \ell_0$ yields

$$\delta_1(\tau_1) \longrightarrow \delta_1(\tau_2) \sim_{\ell_0} \delta_2(\tau_1) \longrightarrow \delta_2(\tau_2) : \star^{\ell_1 \sqcup \ell_2}$$

  Sub-Case $\ell_1 \sqcup \ell_2 \not\sqsubseteq \ell_0$
  - It follows trivially from TSLR:TYPE-OPAQ that

$$\delta_1(\tau_1) \longrightarrow \delta_1(\tau_2) \sim_{\ell_0} \delta_2(\tau_1) \longrightarrow \delta_2(\tau_2) : \star^{\ell_1 \sqcup \ell_2}$$

- Using TSCLR:BASE on $\delta_i(\tau_1) \longrightarrow \delta_i(\tau_2) \rightsquigarrow^* \delta_i(\tau_1) \longrightarrow \delta_i(\tau_2)$ and

$$\delta_1(\tau_1) \longrightarrow \delta_1(\tau_2) \sim_{\ell_0} \delta_2(\tau_1) \longrightarrow \delta_2(\tau_2) : \star^{\ell_1 \sqcup \ell_2}$$

  gives us

$$\delta_1(\tau_1) \longrightarrow \delta_1(\tau_2) \approx_{\ell_0} \delta_2(\tau_1) \longrightarrow \delta_2(\tau_2) : \star^{\ell_1 \sqcup \ell_2}$$

  which by the equality described above, is the same as

$$\delta_1(\tau_1 \longrightarrow \tau_2) \approx_{\ell_0} \delta_2(\tau_1 \longrightarrow \tau_2) : \star^{\ell_1 \sqcup \ell_2}$$

Case The case for WFC:PROD is symmetric to the case for WFC:ARR.

Case

$$\frac{\Delta, \alpha{:}\kappa_1 \vdash \tau : \kappa_2}{\Delta \vdash \lambda\alpha{:}\kappa_1.\tau : \kappa_1 \xrightarrow{\perp} \kappa_2} \text{ WFC:ABS}$$

- By the definition of substitution $\delta_i(\lambda\alpha{:}\kappa_1.\tau) = \lambda\alpha{:}\kappa_1.\delta_i(\tau)$ and by TRC:REFL we know $\lambda\alpha{:}\kappa_1.\delta_i(\tau) \rightsquigarrow^* \lambda\alpha{:}\kappa_1.\delta_i(\tau)$, therefore $\delta_i(\lambda\alpha{:}\kappa_1.\tau) \rightsquigarrow^* \delta_i(\lambda\alpha{:}\kappa_1.\tau)$.
- Assume $\tau_1 \approx_{\ell_0} \tau_2 : \kappa_1$. Therefore, $\delta_1, [\tau_1/\alpha] \approx_{\ell_0} \delta_2, [\tau_2/\alpha] : \Delta, \alpha{:}\kappa_1$ by Definition D.7 and inversion upon $\delta_1 \approx_{\ell_0} \delta_2 : \Delta$.

- Appealing to the induction hypothesis on $\Delta, \alpha{:}\kappa_1 \vdash \tau : \kappa_2$ with $\delta_1, [\tau_1/\alpha] \approx_{\ell_0} \delta_2, [\tau_2/\alpha] : \Delta, \alpha{:}\kappa_1$ we have that

$$(\delta_1, [\tau_1/\alpha])(\tau) \approx_{\ell_0} (\delta_2, [\tau_2/\alpha])(\tau) : \kappa_2$$

- By Lemma D.10 we know that this is the same as

$$(\lambda\alpha{:}\kappa_1.\delta_1(\tau))\tau_1 \approx_{\ell_0} (\lambda\alpha{:}\kappa_1.\delta_2(\tau))\tau_2 : \kappa_2$$

  Furthermore by Lemma D.12 Part 1 on $\kappa_2 \sqsubseteq \kappa_2 \sqcup \bot$ and

$$(\lambda\alpha{:}\kappa_1.\delta_1(\tau))\tau_1 \approx_{\ell_0} (\lambda\alpha{:}\kappa_1.\delta_2(\tau))\tau_2 : \kappa_2$$

  we know that

$$(\lambda\alpha{:}\kappa_1.\delta_1(\tau))\tau_1 \approx_{\ell_0} (\lambda\alpha{:}\kappa_1.\delta_2(\tau))\tau_2 : \kappa_2 \sqcup \bot$$

- Consequently, discharging our assumption we have that

$$\lambda\alpha{:}\kappa_1.\delta_1(\tau) \sim_{\ell_0} \lambda\alpha{:}\kappa_1.\delta_2(\tau) : \kappa_1 \xrightarrow{\bot} \kappa_2$$

  Use of TSCLR:BASE on this and $\lambda\alpha{:}\kappa_1.\delta_i(\tau) \rightsquigarrow^* \lambda\alpha{:}\kappa_1.\delta_i(\tau)$ yields

$$\lambda\alpha{:}\kappa_1.\delta_1(\tau) \approx_{\ell_0} \lambda\alpha{:}\kappa_1.\delta_2(\tau) : \kappa_1 \xrightarrow{\bot} \kappa_2$$

  By the above identity, this is the same as

$$\delta_1(\lambda\alpha{:}\kappa_1.\tau) \approx_{\ell_0} \delta_2(\lambda\alpha{:}\kappa_1.\tau) : \kappa_1 \xrightarrow{\bot} \kappa_2$$

Case

$$\frac{\Delta \vdash \tau_1 : \kappa_1 \xrightarrow{\ell} \kappa_2 \qquad \Delta \vdash \tau_2 : \kappa_1}{\Delta \vdash \tau_1\tau_2 : \kappa_2 \sqcup \ell}\ \text{WFC:APP}$$

- Appealing to the induction hypothesis on $\Delta \vdash \tau_1 : \kappa_1 \xrightarrow{\ell} \kappa_2$ and $\Delta \vdash \tau_2 : \kappa_1$ with $\delta_1 \approx_{\ell_0} \delta_2 : \Delta$ gives us $\delta_1(\tau_1) \approx_{\ell_0} \delta_2(\tau_1) : \kappa_1 \xrightarrow{\ell} \kappa_2$ and $\delta_1(\tau_2) \approx_{\ell_0} \delta_2(\tau_2) : \kappa_1$.
- By inversion upon $\delta_1(\tau_1) \approx_{\ell_0} \delta_2(\tau_1) : \kappa_1 \xrightarrow{\ell} \kappa_2$ we have that $\delta_i(\tau_1) \rightsquigarrow^* \nu_i$ and $\nu_1 \sim_{\ell_0} \nu_2 : \kappa_1 \xrightarrow{\ell} \kappa_2$. By further inversion upon $\nu_1 \sim_{\ell_0} \nu_2 : \kappa_1 \xrightarrow{\ell} \kappa_2$ we know that

$$\forall(\tau_1' \approx_{\ell_0} \tau_2' : \kappa_1).\nu_1\tau_1' \approx_{\ell_0} \nu_2\tau_2' : \kappa_2 \sqcup \ell$$

- Instantiating this with $\delta_1(\tau_2) \approx_{\ell_0} \delta_2(\tau_2) : \kappa_1$ gives us

  $$\nu_1(\delta_1(\tau_2)) \approx_{\ell_0} \nu_2(\delta_2(\tau_2)) : \kappa_2 \sqcup \ell$$

  By inversion on this we get that $\nu_i(\delta_i(\tau_2)) \rightsquigarrow^* \nu_i'$ and $\nu_1' \sim_{\ell_0} \nu_2' : \kappa_2 \sqcup \ell_2$.
- Given $\delta_i(\tau_1) \rightsquigarrow^* \nu_i$ and $\nu_i(\delta_i(\tau_2)) \rightsquigarrow^* \nu_i'$ we know that $\delta_i(\tau_1)\delta_i(\tau_2) \rightsquigarrow^* \nu_i'$. As $\delta_i(\tau_1)\delta_i(\tau_2) = \delta_i(\tau_1\tau_2)$, this is the same as $\delta_i(\tau_1\tau_2) \rightsquigarrow^* \nu_i'$.
- We have what we need and can conclude $\delta_1(\tau_1\tau_2) \approx_{\ell_0} \delta_2(\tau_1\tau_2) : \kappa_2 \sqcup \ell$ by TSCLR:BASE.

Case

$$\frac{\begin{array}{c}\Delta \vdash \tau : \star^\ell \\ \ell \sqsubseteq \ell' \qquad \Delta \vdash \tau_{\mathsf{bool}} : \kappa \qquad \Delta \vdash \tau_{\longrightarrow} : \star^\ell \xrightarrow{\ell'} \star^\ell \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \\ \Delta \vdash \tau_{\times} : \star^\ell \xrightarrow{\ell'} \star^\ell \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \qquad \text{where } \ell' = \mathcal{L}(\kappa)\end{array}}{\Delta \vdash \mathsf{Typerec}\ \tau\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times} : \kappa}\ \text{WFC:TREC}$$

- By appealing to the induction hypothesis on $\delta_1 \approx_{\ell_0} \delta_2 : \Delta$ and
  - $\Delta \vdash \tau : \star^\ell$ and
  - $\Delta \vdash \tau_{\mathsf{bool}} : \kappa$ and
  - $\Delta \vdash \tau_{\longrightarrow} : \star^\ell \xrightarrow{\ell'} \star^\ell \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa$ and
  - $\Delta \vdash \tau_{\times} : \star^\ell \xrightarrow{\ell'} \star^\ell \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa$

  yields
  - $\delta_1(\tau) \approx_{\ell_0} \delta_2(\tau) : \star^\ell$ and
  - $\delta_1(\tau_{\mathsf{bool}}) \approx_{\ell_0} \delta_2(\tau_{\mathsf{bool}}) : \kappa$ and
  - $\delta_1(\tau_{\longrightarrow}) \approx_{\ell_0} \delta_2(\tau_{\longrightarrow}) : \star^\ell \xrightarrow{\ell'} \star^\ell \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa$ and
  - $\delta_1(\tau_{\times}) \approx_{\ell_0} \delta_2(\tau_{\times}) : \star^\ell \xrightarrow{\ell'} \star^\ell \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa \xrightarrow{\ell'} \kappa$

- Using Lemma D.16 on these facts gives us that

  $$\mathsf{Typerec}\ \delta_1(\tau)\ \delta_1(\tau_{\mathsf{bool}})\ \delta_1(\tau_{\longrightarrow})\ \delta_1(\tau_{\times}) \approx_{\ell_0} \mathsf{Typerec}\ \delta_2(\tau)\ \delta_2(\tau_{\mathsf{bool}})\ \delta_2(\tau_{\longrightarrow})\ \delta_2(\tau_{\times}) : \kappa$$

  By the definition of substitution this is identical to

  $$\delta_1(\mathsf{Typerec}\ \tau\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times}) \approx_{\ell_0} \delta_2(\mathsf{Typerec}\ \tau\ \tau_{\mathsf{bool}}\ \tau_{\longrightarrow}\ \tau_{\times}) : \kappa$$

Case

$$\frac{\Delta \vdash \tau : \kappa_1 \qquad \kappa_1 \leq \kappa_2}{\Delta \vdash \tau : \kappa_2}\ \text{WFC:SUB}$$

- First, appeal to the induction hypothesis on $\Delta \vdash \tau : \kappa_1$ with $\delta_1 \approx_{\ell_0} \delta_2 : \Delta$ to conclude $\delta_1(\tau) \approx_{\ell_0} \delta_2(\tau) : \kappa_1$.
- Using Lemma D.12 Part 1. on this with $\kappa_1 \sqsubseteq \kappa_2$ we can conclude the desired result, $\delta_1(\tau) \approx_{\ell_0} \delta_2(\tau) : \kappa_2$.

Part 2 follows by induction over the structure/heights of typing derivations.

Cases The cases for WFT:TRUE and WFT:FALSE are analogous to that for WFC:BOOL.

Case

$$\frac{\Delta^\star \vdash \Gamma \qquad x : \sigma \in \Gamma}{\Delta^\star \mid \Gamma \vdash x : \sigma}\ \textsc{wft:var}$$

- Follows immediately by inversion upon $\eta \vdash \gamma_1 \approx_{\ell_0} \gamma_2 : \Gamma$.

Cases The cases for WFT:ABS and WFT:APP are analogous to those for WFC:ABS and WFC:APP.

Case

$$\frac{\Delta^\star, \alpha{:}\star^\ell \mid \Gamma \vdash e : \sigma}{\Delta^\star \mid \Gamma \vdash \Lambda\alpha{:}\star^\ell.e : \forall^\perp \alpha{:}\star^\ell.\sigma}\ \textsc{wft:tabs}$$

- By the definition of substitution, we know that $\delta_i(\gamma_i(\Lambda\alpha{:}\star^\ell.e)) = \Lambda\alpha{:}\star^\ell.\delta_i(\gamma_i(e))$. Furthermore, by TRC:REFL we know that $\Lambda\alpha{:}\star^\ell.\delta_i(\gamma_i(e)) \rightsquigarrow^* \Lambda\alpha{:}\star^\ell.\delta_i(\gamma_i(e))$. Therefore, we have that $(\delta_i(\gamma_i(\Lambda\alpha{:}\star^\ell.e)) \rightsquigarrow^* (\delta_i(\gamma_i(\Lambda\alpha{:}\star^\ell.e))$.
- Assume $\delta_1(\tau_1) \approx_{\ell_0} \delta_2(\tau_2) : \star^\ell$ and a consistent R such that

$$R^{\ell_2}_\rho \in \delta_1((\rho\{\tau_1\})^{\ell_2}) \leftrightarrow \delta_2((\rho\{\tau_2\})^{\ell_2})$$

- Therefore, by Definition D.7 and RELM:REG we know that $\eta, \alpha \mapsto R \vdash \Delta^\star, \alpha{:}\star^\ell$ and $\delta_1, [\delta_1(\tau_1)/\alpha] \approx_{\ell_0} \delta_2, [\delta_2(\tau_2)/\alpha] : \Delta^\star, \alpha{:}\star^\ell$.
- Appealing to the induction hypothesis on $\Delta^\star, \alpha{:}\star^\ell \mid \Gamma \vdash e : \sigma$ with the above gives us that

$$\eta, \alpha \mapsto R \vdash (\delta_1, [\delta_1(\tau_1)/\alpha])(\gamma_1(e)) \approx_{\ell_0} (\delta_2, [\delta_2(\tau_2)/\alpha])(\gamma_2(e)) : \sigma$$

- Using Lemma D.10 we can conclude that

$$\eta, \alpha \mapsto R \vdash \delta_1(\gamma_1((\Lambda\alpha{:}\star^\ell.e)[\tau_1])) \approx_{\ell_0} \delta_2(\gamma_2((\Lambda\alpha{:}\star^\ell.e)[\tau_2])) : \sigma$$

Furthermore, by Lemma D.12 and $\Delta^\star \vdash \sigma \leq \sigma \sqcup \perp$ we know that

$$\eta, \alpha \mapsto R \vdash \delta_1(\gamma_1((\Lambda\alpha{:}\star^\ell.e)[\tau_1])) \approx_{\ell_0} \delta_2(\gamma_2((\Lambda\alpha{:}\star^\ell.e)[\tau_2])) : \sigma \sqcup \perp$$

- Discharging our assumptions, we have that

$$\eta \vdash \delta_1(\gamma_1(\Lambda\alpha{:}\star^{\ell}.e)) \sim_{\ell_0} \delta_2(\gamma_2(\Lambda\alpha{:}\star^{\ell}.e)) : \forall^{\perp}\alpha{:}\star^{\ell}.\sigma$$

  Using this along with $(\delta_i(\gamma_i(\Lambda\alpha{:}\star^{\ell}.e)) \rightsquigarrow^* (\delta_i(\gamma_i(\Lambda\alpha{:}\star^{\ell}.e))$ and SCLR:TERM we can conclude that

$$\eta \vdash \delta_1(\gamma_1(\Lambda\alpha{:}\star^{\ell}.e)) \approx_{\ell_0} \delta_2(\gamma_2(\Lambda\alpha{:}\star^{\ell}.e)) : \forall^{\perp}\alpha{:}\star^{\ell}.\sigma$$

Case

$$\frac{\Delta^\star \mid \Gamma \vdash e : \forall^{\ell}\alpha{:}\star^{\ell'}.\sigma \qquad \Delta^\star \vdash \tau : \star^{\ell'}}{\Delta^\star \mid \Gamma \vdash e[\tau] : \sigma[\tau/\alpha] \sqcup \ell}\ \text{WFT:TAPP}$$

- Appealing to the induction hypothesis on $\Delta^\star \mid \Gamma \vdash e : \forall^{\ell}\alpha{:}\star^{\ell'}.\sigma$, we get that $\eta \vdash \delta_1(\gamma_1(e)) \approx_{\ell_1} \delta_2(\gamma_2(e)) : \forall^{\ell}\alpha{:}\star^{\ell'}.\sigma$.
- By inversion on $\eta \vdash \delta_1(\gamma_1(e)) \approx_{\ell_0} \delta_2(\gamma_2(e)) : \forall^{\ell}\alpha{:}\star^{\ell'}.\sigma$ we know that either $\delta_i(\gamma_i(e)) \rightsquigarrow^* \nu_i$ or $\delta_i(\gamma_i(e)) \uparrow$.

  Sub-Case $\delta_i(\gamma_i(e)) \rightsquigarrow^* \nu_i$.

  - Also inversion we know that, $\forall^{\ell}\alpha{:}\star^{\ell'}.\sigma' \rightsquigarrow^* \zeta$ and $\eta \vdash \nu_1 \sim_{\ell_0} \nu_2 : \zeta$. By inversion on the weak-head reduction we know that $\zeta = \forall^{\ell}\alpha{:}\star^{\ell'}.\sigma$. Inverting $\eta \vdash \nu_1 \sim_{\ell_1} \nu_2 : \forall^{\ell}\alpha{:}\star^{\ell'}.\sigma$ we know that

$$\begin{array}{l}\forall(\delta_1(\tau_1') \approx_{\ell_0} \delta_2(\tau_2') : \star^{\ell'}).\\ \quad \forall(R_\rho^{\ell'} \in \delta_1((\rho\{\tau_1'\})^{\ell'}) \leftrightarrow \delta_2((\rho\{\tau_2'\})^{\ell'}).\\ \quad\quad \eta, \alpha \mapsto R \vdash \nu_1[\tau_1] \approx_{\ell_1} \nu_2[\tau_2] : \sigma \sqcup \ell\end{array}$$

  - Using Part 1 on $\Delta^\star \vdash \tau : \star^{\ell'}$ we have that $\delta_1(\tau) \approx_{\ell_0} \delta_2(\tau) : \star^{\ell'}$.
  - Choose $R_\rho^{\ell'}$ to be $\{(\nu_1, \nu_2) \mid \eta \vdash \nu_1 \sim_{\ell_0} \nu_2 : \zeta, (\rho\{\tau\})^{\ell'} \rightsquigarrow^* \zeta\}$.
  - Applying $\delta_1(\tau) \approx_{\ell_0} \delta_2(\tau) : \star^{\ell'}$ and R gives us that

$$\eta, \alpha \mapsto R \vdash \nu_1[\delta_1(\tau)] \approx_{\ell_1} \nu_2[\delta_2(\tau)] : \sigma \sqcup \ell$$

    Using Lemma D.15 on this we can conclude

$$\eta \vdash \nu_1[\delta_1(\tau)] \approx_{\ell_1} \nu_2[\delta_2(\tau)] : \sigma[\tau/\alpha] \sqcup \ell$$

  - Given that $\delta_i(\gamma_i(e)) \rightsquigarrow^* \nu_i$ we know that $\delta_i(\gamma_i(e))[\delta_i(\tau)] \rightsquigarrow^* \nu_i[\delta_i(\tau)]$. Using Lemma D.10 we can conclude that

$$\eta \vdash \delta_1(\gamma_1(e))[\delta_1(\tau)] \approx_{\ell_1} \delta_1(\gamma_2(e))[\delta_2(\tau)] : \sigma[\tau/\alpha] \sqcup \ell$$

    which by the definition of substitution is identical to the desired result

$$\eta \vdash \delta_1(\gamma_1(e[\tau])) \approx_{\ell_1} \delta_1(\gamma_2(e[\tau])) : \sigma[\tau/\alpha] \sqcup \ell$$

Sub-Case $\delta_i(\gamma_i(e))\uparrow$.

– Then we know that $\delta_i(\gamma_i(e[\tau]))\uparrow$ as well. Using SCLR:DIVR we can conclude $\eta \vdash \delta_1(\gamma_1(e[\tau])) \approx_{\ell_0} \delta_2(\gamma_2(e[\tau])) : \sigma[\tau/\alpha] \sqcup \ell$.

Case

$$\frac{\Delta^\star \mid \Gamma \vdash e_1 : \sigma_1 \qquad \Delta^\star \mid \Gamma \vdash e_2 : \sigma_2}{\Delta^\star \mid \Gamma \vdash \langle e_1, e_2\rangle : \sigma_1 \times^\perp \sigma_2}\ \text{WFT:PAIR}\ .$$

- By appealing to the induction hypothesis on $\Delta^\star \mid \Gamma \vdash e_1 : \sigma_1$ and $\Delta^\star \mid \Gamma \vdash e_2 : \sigma_2$ with $\delta_1 \approx_{\ell_0} \delta_2 : \Delta^\star$ and $\eta \vdash \Delta^\star$ and $\eta \vdash \gamma_1 \approx_{\ell_0} \gamma_2 : \Gamma$ we have that
$$\eta \vdash \delta_1(\gamma_1(e_1)) \approx_{\ell_0} \delta_2(\gamma_2(e_1)) : \sigma_1$$
and
$$\eta \vdash \delta_1(\gamma_1(e_1)) \approx_{\ell_0} \delta_2(\gamma_2(e_1)) : \sigma_2$$

- By inversion on $\eta \vdash \delta_1(\gamma_1(e_1)) \approx_{\ell_0} \delta_2(\gamma_2(e_1)) : \sigma_1$ either $\delta_i(\gamma_i(e_1)) \rightsquigarrow^* v_{1i}$ or $\delta_i(\gamma_i(e_1))\uparrow$.

  Sub-Case $\delta_i(\gamma_i(e_1)) \rightsquigarrow^* v_{1i}$.

  – By inversion upon $\eta \vdash \delta_1(\gamma_1(e_2)) \approx_{\ell_0} \delta_2(\gamma_2(e_2)) : \sigma_2$ either $\delta_i(\gamma_i(e_2)) \rightsquigarrow^* v_{2i}$ or $\delta_i(\gamma_i(e_2))\uparrow$.

  Sub-Sub-Case $\delta_i(\gamma_i(e_2)) \rightsquigarrow^* v_{2i}$.

  * Because $\delta_i(\gamma_i(e_1)) \rightsquigarrow^* v_{1i}$ and $\delta_i(\gamma_i(e_2)) \rightsquigarrow^* v_{2i}$ we can conclude that $\langle \delta_i(\gamma_i(e_1)), \delta_i(\gamma_i(e_2))\rangle \rightsquigarrow^* \langle v_{1i}, v_{2i}\rangle$ which by the definition of substitution is identical to $\delta_i(\gamma_i(\langle e_1, e_2\rangle)) \rightsquigarrow^* \langle v_{1i}, v_{2i}\rangle$.
  * Therefore, **fst** $\delta_i(\gamma_i(\langle e_1, e_2\rangle)) \rightsquigarrow^* v_{1i}$ and **snd** $\delta_i(\gamma_i(\langle e_1, e_2\rangle)) \rightsquigarrow^* v_{2i}$ respectively. Also by the above inversions upon
  $$\eta \vdash \delta_1(\gamma_1(e_1)) \approx_{\ell_0} \delta_2(\gamma_2(e_1)) : \sigma_1$$
  and
  $$\eta \vdash \delta_1(\gamma_1(e_2)) \approx_{\ell_0} \delta_2(\gamma_2(e_2)) : \sigma_2$$
  we know that $\eta \vdash v_{11} \sim_{\ell_0} v_{12} : \zeta_1$ and $\eta \vdash v_{21} \sim_{\ell_0} v_{22} : \zeta_2$ for $\sigma_1 \rightsquigarrow^* \zeta_1$ and $\sigma_2 \rightsquigarrow^* \zeta_2$.
  * Using Lemma D.12 on these along with $\Delta^\star \vdash \zeta_i \leq \zeta_i \sqcup \perp$ and $\Delta^\star \vdash \sigma_i \leq \sigma_i \sqcup \perp$ we have that $\eta \vdash v_{11} \sim_{\ell_0} v_{12} : \zeta_1 \sqcup \perp$ and $\eta \vdash v_{21} \sim_{\ell_0} v_{22} : \zeta_2 \sqcup \perp$ for $\sigma_1 \sqcup \perp \rightsquigarrow^* \zeta_1 \sqcup \perp$ and $\sigma_2 \sqcup \perp \rightsquigarrow^* \zeta_2 \sqcup \perp$.

* Consequently, by SCLR:TERM we have that

  $$\eta \vdash \mathbf{fst}\, \delta_1(\gamma_1(\langle e_1, e_2\rangle)) \approx_{\ell_0} \mathbf{fst}\, \delta_2(\gamma_2(\langle e_1, e_2\rangle)) : \sigma_1 \sqcup \bot$$

  and

  $$\eta \vdash \mathbf{snd}\, \delta_1(\gamma_1(\langle e_1, e_2\rangle)) \approx_{\ell_0} \mathbf{snd}\, \delta_2(\gamma_2(\langle e_1, e_2\rangle)) : \sigma_2 \sqcup \bot$$

* Finally, by SLR:PROD we can conclude

  $$\eta \vdash \delta_1(\gamma_1(\langle e_1, e_2\rangle)) \sim_{\ell_0} \delta_2(\gamma_2(\langle e_1, e_2\rangle)) : \sigma_1 \times^{\bot} \sigma_2$$

  Using this along with $\langle \delta_i(\gamma_i(e_1)), \delta_i(\gamma_i(e_2))\rangle \rightsquigarrow^* \langle v_{1i}, v_{2i}\rangle$ gives us the desired result

  $$\eta \vdash \delta_1(\gamma_1(\langle e_1, e_2\rangle)) \approx_{\ell_0} \delta_2(\gamma_2(\langle e_1, e_2\rangle)) : \sigma_1 \times^{\bot} \sigma_2$$

Sub-Sub-Case $\delta_i(\gamma_i(e_2)) \uparrow$.

* Then we know that $\delta_i(\gamma_i(\langle e_1, e_2\rangle)) \uparrow$ and we can use SCLR:DIVR to conclude that

  $$\eta \vdash \delta_1(\gamma_1(\langle e_1, e_2\rangle)) \approx_{\ell_0} \delta_2(\gamma_2(\langle e_1, e_2\rangle)) : \sigma_1 \times^{\bot} \sigma_2$$

Sub-Case $\delta_i(\gamma_i(e_1)) \uparrow$.

– Then we know that $\delta_i(\gamma_i(\langle e_1, e_2\rangle)) \uparrow$ and we can use SCLR:DIVR to conclude that

$$\eta \vdash \delta_1(\gamma_1(\langle e_1, e_2\rangle)) \approx_{\ell_0} \delta_2(\gamma_2(\langle e_1, e_2\rangle)) : \sigma_1 \times^{\bot} \sigma_2$$

Case

$$\frac{\Delta^\star \mid \Gamma \vdash e : \sigma_1 \times^{\ell} \sigma_2}{\Delta^\star \mid \Gamma \vdash \mathbf{fst}\, e : \sigma_1 \sqcup \ell}\ \text{WFT:FST}$$

- Appealing to the induction hypothesis on $\Delta^\star \mid \Gamma \vdash e : \sigma_1 \times^{\ell} \sigma_2$ we know that $\eta \vdash \delta_1(\gamma_1(e)) \approx_{\ell_0} \delta_2(\gamma_2(e)) : \sigma_1 \times^{\ell} \sigma_2$.
- By inversion upon $\eta \vdash \delta_1(\gamma_1(e)) \approx_{\ell_0} \delta_2(\gamma_2(e)) : \sigma_1 \times^{\ell} \sigma_2$ we know that either $\delta_i(\gamma_i(e)) \rightsquigarrow^* v_i$ or $\delta_i(\gamma_i(e)) \uparrow$.

  Sub-Case $\delta_i(\gamma_i(e)) \rightsquigarrow^* v_i$,

  – Also by inversion upon

  $$\eta \vdash \delta_1(\gamma_1(e)) \approx_{\ell_0} \delta_2(\gamma_2(e)) : \sigma_1 \times^{\ell} \sigma_2$$

  we have that $\sigma_1 \times^{\ell} \sigma_2 \rightsquigarrow^* \sigma' \; \eta \vdash v_1 \sim_{\ell_0} v_2 : \sigma'$.

- By inversion upon $\sigma_1 \times^{\ell} \sigma_2 \rightsquigarrow^* \sigma'$ we know that $\sigma' = \sigma_1 \times^{\ell} \sigma_2$.
- By inversion upon $\eta \vdash v_1 \sim_{\ell_0} v_2 : \sigma_1 \times^{\ell} \sigma_2$ we know that $\eta \vdash \mathbf{fst}\, v_1 \approx_{\ell_0} \mathbf{fst}\, v_2 : \sigma_1 \sqcup \ell$ and $\eta \vdash \mathbf{snd}\, v_1 \approx_{\ell_0} \mathbf{snd}\, v_2 : \sigma_2 \sqcup \ell$.
- Given that $\delta_i(\gamma_i(e)) \rightsquigarrow^* v_i$ we know that $\mathbf{fst}\, \delta_i(\gamma_i(e)) \rightsquigarrow^* \mathbf{fst}\, v_i$ which by the definition of substitution is the same as $\delta_i(\gamma_i(\mathbf{fst}\, e)) \rightsquigarrow^* \mathbf{fst}\, v_i$. Therefore by Lemma D.10 we can conclude that

$$\eta \vdash \delta_1(\gamma_1(\mathbf{fst}\, e)) \approx_{\ell_0} \delta_2(\gamma_2(\mathbf{fst}\, e)) : \sigma_1 \sqcup \ell$$

Sub-Case $\delta_i(\gamma_i(e)) \uparrow$

- Therefore, we can conclude that $\mathbf{fst}\, \delta_i(\gamma_i(e)) \uparrow$, which by the definition of substitution is the same as $\delta_i(\gamma_i(\mathbf{fst}\, e)) \uparrow$. Therefore, regardless of whether $i = 1$ or $i = 2$ by SCLR:DIVR we have that $\eta \vdash \delta_1(\gamma_1(\mathbf{fst}\, e)) \approx_{\ell_0} \delta_2(\gamma_2(\mathbf{fst}\, e)) : \sigma_1 \sqcup \ell$.

Case The case for WFT:SND is symmetric to the case for WFT:FST.

Case

$$\frac{\Delta^\star \mid \Gamma \vdash e_1 : (\mathsf{bool})^{\ell} \qquad \Delta^\star \mid \Gamma \vdash e_2 : \sigma \qquad \Delta^\star \mid \Gamma \vdash e_3 : \sigma}{\Delta^\star \mid \Gamma \vdash \mathbf{if}\, e_1\, \mathbf{then}\, e_2\, \mathbf{else}\, e_3 : \sigma \sqcup \ell}\ \text{WFT:IF}$$

Sub-Case $\ell \not\sqsubseteq \ell_0$.

- Then by Lemma D.14 Part 2 we know that

$$\eta \vdash \delta_1(g_1(\mathbf{if}\, e_1\, \mathbf{then}\, e_2\, \mathbf{else}\, e_3)) \approx_{\ell_0} \delta_2(g_2(\mathbf{if}\, e_1\, \mathbf{then}\, e_2\, \mathbf{else}\, e_3)) : \sigma \sqcup \ell$$

Sub-Case $\ell \sqsubseteq \ell_0$.

- By appealing to the induction hypothesis on $\Delta^\star \mid \Gamma \vdash e_1 : (\mathsf{bool})^{\ell}$ we know that $\eta \vdash \delta_1(\gamma_1(e_1)) \approx_{\ell_0} \delta_2(\gamma_2(e_1)) : (\mathsf{bool})^{\ell}$. By inversion on this we know that either $\delta_i(\gamma_i(e_1)) \rightsquigarrow^* v_i$ or $\delta_i(\gamma_i(e_1)) \uparrow$.

  Sub-Sub-Case $\delta_i(\gamma_i(e_1)) \rightsquigarrow^* v_i$.

  - Also by inversion we know that $\eta \vdash v_1 \sim_{\ell_0} v_2 : \zeta$ where $(\mathsf{bool})^{\ell} \rightsquigarrow^* \zeta$. And by inversion on the weak-head reduction we know that $\zeta = (\mathsf{bool})^{\ell}$.
  - Therefore, by inversion upon $\eta \vdash v_1 \sim_{\ell_0} v_2 : (\mathsf{bool})^{\ell}$ we can conclude $\ell \sqsubseteq \ell_0 \Rightarrow v_1 = v_2$. We assumed that $\ell \sqsubseteq \ell_0$, so $v_1 = v_2$.
  - By Lemma B.15 we know that $v_i = \mathbf{true}$ or $v_i = \mathbf{false}$.

Sub-Sub-Sub-Case $v_i = \textbf{true}$. By appealing to the induction hypothesis on $\Delta^\star \mid \Gamma \vdash e_1 : (\mathsf{bool})^\ell$ we know that

$$\eta \vdash \delta_1(\gamma_1(e_2)) \approx_{\ell_0} \delta_2(\gamma_2(e_2)) : \sigma$$

By Lemma D.12 we can conclude

$$\eta \vdash \delta_1(\gamma_1(e_2)) \approx_{\ell_0} \delta_2(\gamma_2(e_2)) : \sigma \sqcup \ell$$

We know that $\delta_i(\gamma_i(\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3)) \rightsquigarrow^* \delta_i(\gamma_i(e_2))$, therefore by Lemma D.10 we can conclude the desired result

$$\eta \vdash \delta_1(g_1(\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3)) \approx_{\ell_0} \delta_2(g_2(\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3)) : \sigma \sqcup \ell$$

Sub-Sub-Sub-Case The case for $v_i = \textbf{false}$ is symmetric.

Sub-Sub-Case $\delta_i(\gamma_i(e_1)) \uparrow$.

- Then we know that $\delta_i(g_i(\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3)) \uparrow$ and can use SCLR:DIVR to conclude that

$$\eta \vdash \delta_1(g_1(\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3)) \approx_{\ell_0} \delta_2(g_2(\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3)) : \sigma \sqcup \ell$$

Case

$$\frac{\Delta^\star \mid \Gamma, x{:}\sigma \vdash e : \sigma \qquad \Delta^\star \vdash \sigma}{\Delta^\star \mid \Gamma \vdash \textbf{fix}_n\, x{:}\sigma.e : \sigma}\ \text{WFT:FIXN}$$

- By the definition of substitution, we know that $\delta_i(\gamma_i(\textbf{fix}_n\, x{:}\sigma.e)) = \textbf{fix}_n\, x{:}\sigma.\delta_i(\gamma_i(e))$.
- The case follows from induction upon n.

  Sub-Case $n = 0$.

  - By Lemma C.4 we know that $\textbf{fix}_0\, x{:}\sigma.\delta_i(\gamma_i(e)) \uparrow$. Therefore, by SCLR:DIVR we can conclude that

$$\eta \vdash \textbf{fix}_0\, x{:}\sigma.\delta_1(\gamma_1(e)) \approx_{\ell_0} \textbf{fix}_0\, x{:}\sigma.\delta_2(\gamma_2(e)) : \sigma$$

  - By the above identity, this means that we have

$$\eta \vdash \delta_1(\gamma_1(\textbf{fix}_0\, x{:}\sigma.e)) \approx_{\ell_0} \delta_2(\gamma_2(\textbf{fix}_0\, x{:}\sigma.e)) : \sigma$$

  Sub-Case $n = m + 1$.

  - By appealing to the local induction hypothesis on m gives us that $\eta \vdash \delta_1(\gamma_1(\textbf{fix}_m\, x{:}\sigma.e)) \approx_{\ell_0} \delta_2(\gamma_2(\textbf{fix}_m\, x{:}\sigma.e)) : \sigma$.

– By Definition D.9 and inversion upon $\eta \vdash \gamma_1 \approx_{\ell_0} \gamma_2 : \Gamma$ we can conclude that

$$\eta \vdash \gamma_1, [\gamma_1(\mathbf{fix}_m\, x{:}\sigma.e)/x] \approx_{\ell_0} \gamma_2, [\gamma_2(\mathbf{fix}_m\, x{:}\sigma.e)/x] : \Gamma, x{:}\sigma$$

– Appealing to the global induction hypothesis on $\Delta^\star \mid \Gamma, x{:}\sigma \vdash e : \sigma$ with

$$\eta \vdash \gamma_1, [\gamma_1(\mathbf{fix}_m\, x{:}\sigma.e)/x] \approx_{\ell_0} \gamma_2, [\gamma_2(\mathbf{fix}_m\, x{:}\sigma.e)/x] : \Gamma, x{:}\sigma$$

gives us that

$$\eta \vdash \delta_1((\gamma_1, [\gamma_1(\mathbf{fix}_m\, x{:}\sigma.e)/x])(e)) \approx_{\ell_0} \delta_2((\gamma_2, [\gamma_2(\mathbf{fix}_m\, x{:}\sigma.e)/x])(e)) : \sigma$$

– Trivially, $n - 1 = m$, so using Lemmas D.10 on

$$\eta \vdash \delta_1((\gamma_1, [\gamma_1(\mathbf{fix}_m\, x{:}\sigma.e)/x])(e)) \approx_{\ell_0} \delta_2((\gamma_2, [\gamma_2(\mathbf{fix}_m\, x{:}\sigma.e)/x])(e)) : \sigma$$

we can conclude

$$\eta \vdash \delta_1(\gamma_1(\mathbf{fix}_n\, x{:}\sigma.e)) \approx_{\ell_0} \delta_2(\gamma_2(\mathbf{fix}_n\, x{:}\sigma.e)) : \sigma$$

Case

$$\frac{\Delta^\star \mid \Gamma, x{:}\sigma \vdash e : \sigma \qquad \Delta^\star \vdash \sigma}{\Delta^\star \mid \Gamma \vdash \mathbf{fix}\, x{:}\sigma.e : \sigma}\ \textsc{wft:fix}$$

- Using Lemma C.5 we know that for all$n$, $\Delta^\star \mid \Gamma \vdash \mathbf{fix}_n\, x{:}\sigma.e : \sigma$.
- Therefore, assume an arbitrary m. Appealing to the induction hypothesis on $\Delta^\star \mid \Gamma \vdash \mathbf{fix}_m\, x{:}\sigma.e : \sigma$ with $\eta \vdash \gamma_1 \approx_{\ell_0} \gamma_2 : \Gamma$ gives us that $\eta \vdash \delta_1(\gamma_1(\mathbf{fix}_m\, x{:}\sigma.e)) \approx_{\ell_0} \delta_2(\gamma_2(\mathbf{fix}_m\, x{:}\sigma.e)) : \sigma$.
- By the definition of substitution $\delta_i(\gamma_i(\mathbf{fix}_m\, x{:}\sigma.e)) = \mathbf{fix}_m\, x{:}\delta_i(\sigma).\delta_i(\gamma_i(e))$. Therefore, we have that

$$\eta \vdash \mathbf{fix}_m\, x{:}\delta_1(\sigma).\delta_1(\gamma_1(e)) \approx_{\ell_0} \mathbf{fix}_m\, x{:}\delta_2(\sigma).\delta_2(\gamma_2(e)) : \sigma$$

- Discharging our assumption we have that for all n,

$$\eta \vdash \mathbf{fix}_n\, x{:}\delta_1(\sigma).\delta_1(\gamma_1(e)) \approx_{\ell_0} \mathbf{fix}_n\, x{:}\delta_2(\sigma).\delta_2(\gamma_2(e)) : \sigma$$

Using Lemma 3.3 we can conclude

$$\eta \vdash \mathbf{fix}\, x{:}\delta_1(\sigma).\delta_1(\gamma_1(e)) \approx_{\ell_0} \mathbf{fix}\, x{:}\delta_2(\sigma).\delta_2(\gamma_2(e)) : \sigma$$

- Again by the definition of substitution, $\delta_i(\gamma_i(\textbf{fix}\, x{:}\sigma.e)) = \textbf{fix}\, x{:}\delta_i(\sigma).\delta_i(\gamma_i(e))$. Therefore, we have the desired result

$$\eta \vdash \delta_1(\gamma_1(\textbf{fix}\, x{:}\sigma.e)) \approx_{\ell_0} \delta_2(\gamma_2(\textbf{fix}\, x{:}\sigma.e)) : \sigma$$

Case The case for WFT:TCASE is analogous to WFT:IF and WFT:TAPP.

Case The case for WFT:SUB is analogous to that for WFC:SUB.

□

Corollary D.19 (Confidentiality)**.** *If* $\alpha{:}\star^\top \mid x{:}(\alpha)^\perp \vdash e : (\mathsf{bool})^\perp$ *then for any* $\cdot \vdash v_1 : \tau_1$ *and* $\vdash v_2 : \tau_2$ *if* $e[\tau_1/\alpha][v_2/x]$ *and* $e[\tau_2/\alpha][v_2/x]$ *both terminate, they will produce the same value.*

Proof. Then construct a derivation that $\cdot \mid \cdot \vdash \Lambda\alpha{:}\star^\top.\lambda x{:}(\alpha)^\perp.e : \forall\alpha{:}\star^\top.(\alpha)^\perp \xrightarrow{\perp} (\mathsf{bool})^\perp$ using the appropriate typing rules and then appeal to Theorem D.18 Part 2 to obtain

$$\cdot \vdash \Lambda\alpha{:}\star^\top.\lambda x{:}(\alpha)^\perp.e \sim_\perp \Lambda\alpha{:}\star^\top.\lambda x{:}(\alpha)^\perp.e : \forall\alpha{:}\star^\top.(\alpha)^\perp \xrightarrow{\perp} (\mathsf{bool})^\perp$$

By Lemma D.14 Part 1 we can have that $\tau_1 \approx_\perp \tau_2 : \star^\top$. Next, by inversion on SLR:ALL and instantiation with the constructor relation, $\tau_1 \approx_\perp \tau_2 : \star^\top$, and the relation

$$R^\ell_\rho = \{(v_1, v_2) \mid (\cdot \mid \cdot \vdash v_1 : (\rho\{\tau_1\})^\ell), (\cdot \mid \cdot \vdash v_2 : (\rho\{\tau_2\})^\ell)\}$$

we can conclude that

$$\cdot, \alpha \mapsto R \vdash (\Lambda\alpha{:}\star^\top.\lambda x{:}(\alpha)^\perp.e)[\tau_1] \approx_\perp (\Lambda\alpha{:}\star^\top.\lambda x{:}(\alpha)^\perp.e)[\tau_2] : (\alpha)^\perp \xrightarrow{\perp} (\mathsf{bool})^\perp$$

By straightforward application of SLR:VAR we have that

$$\cdot, \alpha \mapsto R \vdash v_1 \sim_\perp v_2 : (\alpha)^\perp$$

so by application of SCLR:TERM, inversion on SCLR:ARR, and instantiation we know

$$\cdot, \alpha \mapsto R \vdash (\Lambda\alpha{:}\star^\top.\lambda x{:}(\alpha)^\perp.e)[\tau_1]v_1 \approx_\perp (\Lambda\alpha{:}\star^\top.\lambda x{:}(\alpha)^\perp.e)[\tau_2]v_2 : (\mathsf{bool})^\perp$$

Finally, because the relation is closed under reduction we have SLR:ARR and instantiation we have

$$\cdot, \alpha \mapsto R \vdash e[\tau_1/\alpha][v_1/x] \approx_\perp e[\tau_2/\alpha][v_2/x] : (\mathsf{bool})^\perp$$

from which the desired conclusion can be obtained by simple inversion. □

Corollary D.20 (Noninterference)**.** *If* $\cdot, x{:}\sigma_1 \vdash e : \sigma_2$ *where* $\mathcal{L}(\sigma_1) \not\sqsubseteq \mathcal{L}(\sigma_2)$ *then for any* $\vdash v_1 : \sigma_1$ *and* $\vdash v_2 : \sigma_1$ *it is the case that if both* $e[v_1/x]$ *and* $e[v_2/x]$ *terminate, they will both produce the same value*

Proof. Proceeds in a similar fashion to Corollary D.19. □

Corollary D.21 (Integrity)**.** *If* $\alpha{:}\star^\top \mid \cdot \vdash e : (\alpha)^\perp$ *then* $e[\tau/\alpha]$ *for any* τ *must diverge.*

Proof. First construct a derivation that $\cdot \mid \cdot \vdash \Lambda\alpha{:}\star^\top.e : \forall\alpha^\top{:}(\alpha)^\perp$ using the appropriate typing rules, then appeal to Theorem D.18 Part 2 to obtain to obtain

$$\cdot \vdash \Lambda\alpha{:}\star^\top.e \sim_\perp \Lambda\alpha{:}\star^\top.e : \forall\alpha{:}\star^\top.(\alpha)^\perp$$

Now assume an arbitrary τ. It is straightforward to show that $\tau \approx_\perp \tau : \star^\top$. By inversion on SLR:ALL and instantiation we can conclude

$$\cdot, \alpha \mapsto \varnothing \vdash (\Lambda\alpha{:}\star^\top.e)[\tau] \approx_\perp (\Lambda\alpha{:}\star^\top.e)[\tau] : (\alpha)^\perp$$

Because the relation is closed under reduction we have that

$$\cdot, \alpha \mapsto \varnothing \vdash e[\tau/\alpha] \approx_\perp e[\tau/\alpha] : (\alpha)^\perp$$

Furthermore, by inversion either $e[\tau/\alpha] \rightsquigarrow^* v$ or $e[\tau/\alpha] \uparrow$. However in the former case that would mean that

$$\cdot, \alpha \mapsto \varnothing \vdash v \sim_\perp v : (\alpha)^\perp$$

which by inversion on SLR:VAR is impossible because there is no v such that $v \varnothing v$. Therefore $e[\tau/\alpha] \uparrow$. □