Correctness of Analysis-based Program Transformations of Functional Programming Languages

A Thesis

Presented to the Faculty of the Graduate School

of the College of Computer Science of Northeastern University

in Partial Fullfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Igor A. Siveroni

College of Computer Science

Northeastern University

360 Huntington Avenue, 161 CN

Boston, MA 02115, USA

January 2002

Abstract

This dissertation is concerned with the correctness of analysis-based program transformations of functional programming languages, that is, the process of proving that a transformation preserves the meaning of the original program. We claim that the core of this process relies on the specification of the program property that justifies the transformation in terms of the semantics of the language. We achieve this by defining program properties as validation systems based on execution traces of programs which can then be used to derive program analyses using the well-known techniques of abstract interpretation. We then divide the correctness proof into two parts: (1) The transformation is first validated against the property specification while (2) the analysis is proved to satisfy the same specification. Program properties link both parts.

We show the correctness of two program transformations. The first transformation identifies code expressions that do not contribute to the final value of the program and replaces them with less computationally-expensive expressions e.g. constants. The second transformation removes such useless code, a process that also involves the removal of useless variables. In both cases we specify the properties of the information used by the transformations and use these specifications to derive program analyses. We show that the results of the analyses satisfy the specifications and that the transformation is meaning-preserving. We use big-step semantics, where execution traces are defined as proof (derivation) trees, and present the correctness proofs as tree (bi)simulation results.

A mis padres, Jorge y Amparo.

Acknowledgements

I would like thank Mitch Wand for his patience and support during this past years. Mitch gave me enough room to "find myself" and, at the same time, always made sure that I didn't get lost in the way. I consider myself very lucky to have been trained by Mitch in the sometimes obscure but always fascinating area of semantics and formal methods.

I would also like to thank the College of Computer Science - teachers, staff, friends - at Northeastern for providing a friendly and always supportive working environment. I take with me happy memories from Northeastern and Boston.

The last part of this dissertation was written during my stay at the Department of Computing, Imperial College, London. This would have not been possible without the support of Chris Hankin, who kindly allowed me to dedicate long hours to finish this document.

I would have never been able to start this long trip without the support of my parents, who were always there for me. This wouldn't make sense without them. And Deborah, thanks for all your patience and love, this is for you too.

Contents

List of Figures

Chapter 1

Introduction

Program transformations are an important part of optimizing compilers. Optimizing compilers perform a series of program transformations that look to provide code that utilizes computer resources (memory, computer cycles, etc) more efficiently.

The process of optimization by program transformation can be divided into two tasks. The first task deals with the identification and computation of aspects of a program's behavior, *i.e.*, program properties, that can be used to improve the performance of the program. The second task consists on the transformation proper, that is, the generation of code specialized to take advantage of the properties identified by the first part. In most of the cases, the identification of exact program properties, which will lead to an optimal transformation, would require the actual execution of the program. This is obviously unacceptable. Instead, compilers rely on program analysis, the process of determining approximations - though safe - of such properties without actually running the program. These combination elements define an *analysis-based program transformation.*

Compilers must produce code that is efficient and correct. In the context of program transformations, we must make sure that the transformations preserve the meaning of the original program. This is not just a theoretical issue. Some transformations make radical changes to the original program that may alter the behavior of the program in unexpected ways which may be catastrophic for safety critical systems. Correctness is a priority.

This dissertation is concerned with the correctness of analysis-based program transformations of functional programming languages. We claim that the core of the process of proving the correctness of a program transformation relies on the specification of the program property that makes it work in terms of the semantics of the language. We achieve this by defining program properties as validation systems based on execution traces of programs which can then be used to derive program analyses using the well-known techniques of abstract interpretation. In the same way as the process of analysis-based program transformation is divided in two, where properties are the link between the two parts, the correctness of a transformation is divided into two proof obligations: (1) The transformation is validated against the property specification while (2) the analysis is proved to satisfy the same specification separately. This separation could allow us to re-use the property specification with other transformations or different analyses. The examples shown in this dissertation only link properties to a single tranformation and a single family of analyses.

The choice of functional languages as the subject of study of this dissertation does not restrict the applicability of these ideas and techniques to other systems, the only restriction being they must be formally defined e.g. in the case of a programming language, the formal semantics must be given. Functional languages provide the right balance between simplicity and complexity. While their semantics provide a simple and well-defined tool to reason about program behavior, the presence of higher-order functions make program analyses and transformations, standard to imperative languages, non-trivial.

We use natural (or big-step) semantics as the main tool to express program properties. Program properties are defined as properties of proof or derivation *trees*, the equivalent of execution traces in the big-step framework. Relations between program executions are presented as (bi)simulations between derivation trees, *i.e.*, as relations that match the nodes of the derivation trees. Correctness results are presented as (bi)simulation results, provided that one of the trees satisfy a particular property. Similarly, program analyses are proved correct by showing that they safely simulate the derivation trees generated by the property specification, where safety is formulated as a relation between nodes of the trees. We have chosen this approach - the use of natural semantics - because it provides a clearer understanding of the relation between analysis and the property specification: program analyses are usually defined in big-step style.

1.1 Overview: Correctness of Program Transformations

The semantics of a language allows us to reason formally about the execution of programs. In the case of a small-step semantics, execution is denoted by a relation between program states that reflects the execution of a single step of the computation. In particular, we write $S \rightarrow S'$ to denote that state S evaluates to S' in a single step, and say that state S evaluates to value v if $S \rightarrow^* v$, where $\rightarrow^*$ stands for the reflexive, transitive closure of $\rightarrow$.

Let's define $\mathcal{T}_G(S)$ as the transformation of program state S based on proposition G. We will state the correctness of $\mathcal{T}_G$ only for programs that evaluate to constants. Let c denote constants. The correctness of the transformation is stated as follows:

$$\forall S, G, c \;.\quad (\ (S \rightarrow^* c) \quad \Rightarrow \quad (\mathcal{T}_G(S) \rightarrow^* c)\ ) \tag{1.1}$$

that is, the transformation preserves the meaning of the program if it terminates.

We identify three elements in an analysis based transformation: The transformation $\mathcal{T}_G$, a program property G about the evaluation of the program, and an analysis that determines G such that $\mathcal{T}_G$ defines a meaning preserving transformation. For example, G could be a mapping from program expressions to constants such that the expressions considered always evaluate to the mapped constant and $\mathcal{T}_G$ the transformation that replaces those expressions by the respective constant, or G could be the set of expressions that are never executed - or,

$$
\begin{array}{ccccccc}
 & \models G & & & \models G & \\
S & \longrightarrow & S_2 & \ldots & \longrightarrow & c \\
\wr & & \wr & & & \wr \\
\mathcal{R} & & \boxed{\mathcal{R}} & & & \boxed{\mathcal{R}} \\
\wr & & \wr & & & \wr \\
\mathcal{T}_G(S) & \longrightarrow & S'_2 & \ldots & \rightarrow & v'
\end{array}
$$

Figure 1.1: Correctness in small-step style

if executed, do not contribute to the final value of the program - and $\mathcal{T}_G$ the transformation that removes those expressions from the program. The rest of this chapter outlines the relation between these elements.

The transformation and the program property: The usual way of proving (1.1) is by subject reduction. This is done by first defining a relation $\mathcal{R}$ between program states such that $(S\ \mathcal{R}\ \mathcal{T}_G(S))$ and by showing that, given $(S_1\ \mathcal{R}\ S'_1)$ and $S_1 \rightarrow S_2$:

$$
\begin{array}{lll}
(1) & \exists\, S'_2 \text{ s.t. } S'_1 \rightarrow S'_2 & \text{(progress)} \\
(2) & \forall\, S'_2 \text{ s.t. } S'_1 \rightarrow S'_2,\ (S_2\ \mathcal{R}\ S'_2) & \text{(safety)}
\end{array}
$$

We get the desired result by making sure that $c\ \mathcal{R}\ v'$ if and only if $c = v'$. Unfortunately the above is false precisely for the parts of the program affected by the transformation! This is the point where the program property G that the transformation claims to exploit comes into effect.

Let's write $(S_1 \rightarrow S_2) \models G$ to denote that proposition G is true about the evaluation step $S_1 \rightarrow S_2$. The subject reduction result, when unwound, is presented in Figure 1.1. We want to show that the application of the conditions defined by $(S_1 \rightarrow S_2) \models G$ to all of the steps of the computation ensures the correctness of the transformation. We say that execution trace $[S_1 \rightarrow^* v]$ has property G, and write $[S_1 \rightarrow^* v] \models G$, if we can prove $S_1 \overset{\models G}{\rightarrow} S_2 \ldots \overset{\models G}{\rightarrow} c$. Thus, a global property of the execution of a program is specified by constraining, or validating G against, every step of the computation.

The correctness of the transformation is re-written as:

$$\forall S, G, c\ .\quad (\ [S \to^* c] \models G \quad \Rightarrow \quad (\mathcal{T}_G(S) \to^* c)\ ) \tag{1.2}$$

The program property and the analysis: But, what about the analysis? The analysis, as we mentioned before, is in charge of computing G without actually executing $[S \to^* c]$. This is achieved by reasoning about an abstract interpretation $[\widehat{S} \to_a^* \hat{v}]$ that, not only safely simulates $[S \to^* c]$, but it also abstracts over all executions of the program, *i.e.*, all executions $[S_p \to^* c]$ where S_p is an initial configuration of a program P. We say that $[\widehat{S} \to_a^* \hat{v}]$ safely simulates $[S \to^* c]$ if every step $S_i \to S_{i+1}$ in $[S \to^* c]$ is simulated by a step $\widehat{S}_j \to_a \widehat{S}_{j+1}$ in $[\widehat{S} \to_a^* \hat{v}]$.

The analysis $[\widehat{S} \to_a^* \hat{v}] \models G$ is defined by adapting the rules that defined $[S \to^* c] \models G$ to the new interpretation. Let's write $(\widehat{S}_1 \to_a \widehat{S}_2) \models G$ if G is true about the abstract evaluation step $(\widehat{S}_1 \to_a \widehat{S}_2)$. We write $[\widehat{S} \to_a^* \hat{v}] \models G$ if $(\widehat{S}_1 \to_a \widehat{S}_2) \models G$ for all steps $(\widehat{S}_1 \to_a \widehat{S}_2)$ in $[\widehat{S} \to_a^* \hat{v}]$.

The analysis must satisfy the following:

$$\forall S, \widehat{S}, c, \hat{v}, G\ .\quad (\ ([\widehat{S} \to_a^* \hat{v}] \models G \quad \wedge \quad [S \to^* c]) \quad \Rightarrow \quad [S \to^* c] \models G\ ) \tag{1.3}$$

where $[\widehat{S} \to_a^* \hat{v}]$ is an abstract interpretation of $[S \to^* c]$, *i.e.*, $[\widehat{S} \to_a^* \hat{v}] \models G$ is an analysis of $[S \to^* c]$. If G is true about the abstract interpretation then it must be true about the respective concrete interpretation, whenever it exists. This last condition comes from the fact that, in order to be useful, all program analyses must terminate, even though the analyzed program doesn't.

(1.3) is proved by showing that, given $(S' \to S'')$ in $[S \to^* c]$:

$$(\widehat{S}' \to_a \widehat{S}'') \models G \quad \Rightarrow \quad (S' \to S'') \models G \tag{1.4}$$

where $(\widehat{S}' \to_a \widehat{S}'')$ is the abstract step in $[\widehat{S} \to_a^* \hat{v}]$ that simulates $(S' \to S'')$. The existence of such step is ensured by the definition of safety simulation.

1.1.1 Putting Everything Together

We propose that the correctness of analysis-based transformations should be divided into two steps. We must prove the following:

$$\forall\, G, S, \widehat{S}, c, \hat{v} :$$

$$[S \to^* c] \models G \quad \Rightarrow \quad \mathcal{T}_G(S) \to^* c \tag{1.2}$$

$$[\widehat{S} \to_a^* \hat{v}] \models G \quad \wedge \quad [S \to^* c] \quad \Rightarrow \quad [S \to^* c] \models G \tag{1.3}$$

The link between the two steps is the definition of program property $[S \to^* c] \models G$. This definition is based on the execution trace of $S \to^* c$ and is used to prove, in the first step, that the transformation preserves the meaning of the original program whenever G satisfies the alleged property. The second step proves that any analysis of the program provides a G that satisfies the property.

Putting everything together we have:

Theorem 1.1.1 (Correctness of the Analysis wrt the Transformation)
For all $S, \widehat{S}, c, G$: if (1.2) and (1.3) hold then $(\mathcal{T}_G(S) \to^* c)$.

which is another way of saying that the analysis justifies the transformation.

We translate this scheme to a natural semantics setting and apply it to all the transformations presented in this dissertation.

1.2 Big-Step Approach to Correctness

We will use the natural [Kah87] or big-step style of operational semantics for the rest of the dissertation. We have made this choice because we believe that the relations between property definitions and analyses are more clear when presented in big-step style.

In a big-step scenario, the semantics of the language is defined by judgements $\mathcal{C} \Downarrow v$ where configuration $\mathcal{C}$ evaluates to value v. Execution traces are presented as derivation trees where the steps performed during the evaluation of configuration $\mathcal{C}$ can be found by inspecting the tree generated by the proof of $\mathcal{C} \Downarrow v$, if such v exists.

Let $[\mathcal{C} \Downarrow v]$ be the derivation tree generated by a proof of $\mathcal{C} \Downarrow v$. How can we define properties of these derivation trees? We will write $[\mathcal{C} \Downarrow v] \models G$ when G is true about $[\mathcal{C} \Downarrow v]$. A definition like this, when unwound, will look like:

$$\cfrac{\cfrac{\vdots}{\mathcal{C}_1 \Downarrow v_1} \models^{loc} G \quad \cfrac{\vdots}{\mathcal{C}_i \Downarrow v_i} \models^{loc} G \quad \cfrac{\vdots}{\mathcal{C}_n \Downarrow v_n} \models^{loc} G}{\mathcal{C} \Downarrow v} \models^{loc} G$$

where $(\mathcal{C} \Downarrow v) \models^{loc} G$ defines a local constraint placed upon the nodes - or subtrees - of the derivation tree. Properties of global behavior are expressed as properties of local behavior.

Derivation trees are related by (bi)simulations. Given a safety relation $\mathcal{R}$, derivation trees $[\mathcal{N}]$ and $[\mathcal{M}]$ form an $\mathcal{R}$-simulation - $[\mathcal{N}] \preceq^{\mathcal{R}} [\mathcal{M}]$ - if every node in $[\mathcal{N}]$ is $\mathcal{R}$-related to a node in $[\mathcal{M}]$. $[\mathcal{N}]$ and $[\mathcal{M}]$ form an $\mathcal{R}$-bisimulation - $[\mathcal{N}] \cong^{\mathcal{R}} [\mathcal{M}]$ - if the match between nodes of both trees is one-to-one. In both cases the matches must respect the structure of the trees.

The scenario of Figure 1.1 is presented in a big-step setting in Figure 1.2. Given $[\mathcal{C} \Downarrow v] \models G$, we must prove that $\exists v'.\ [\mathcal{T}_G(\mathcal{C}) \Downarrow v'] \preceq^{\mathcal{R}} [\mathcal{C} \Downarrow v]$. This is done by induction on the size of $[\mathcal{C} \Downarrow v]$: simulation between immediate subtrees, and $\models^{loc}$, must be enough to prove that the parent trees form a simulation too. We use a simulation because it is expected that the left-hand side tree contains less nodes than the right-hand side tree. If we define $\mathcal{R}$ such that $c\ \mathcal{R}\ v \Leftrightarrow c = v$ then we can prove $\mathcal{T}_G(\mathcal{C}) \Downarrow c$.

The analysis is derived by performing abstract interpretation to the natural semantics of the language. Given $[\widehat{\mathcal{C}} \widehat{\Downarrow} \hat{v}]$ an abstract interpretation of $[\mathcal{C} \Downarrow v]$ and $[\widehat{\mathcal{C}} \widehat{\Downarrow} \hat{v}] \models G$ if G is

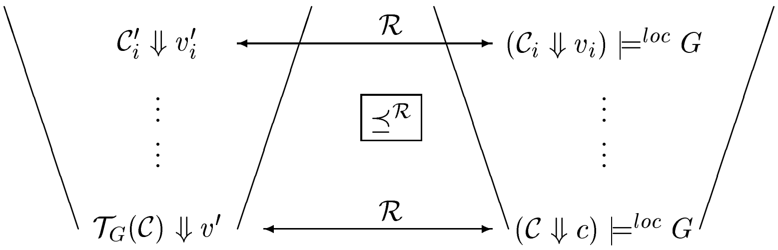

Figure 1.2: Correctness big-step style

true about $[\widehat{\mathcal{C}} \ \widehat{\Downarrow} \ \hat{v}]$, we want to prove the following:

$$([\mathcal{C} \Downarrow v] \preceq^{\widehat{\mathcal{R}}} [\widehat{\mathcal{C}} \ \widehat{\Downarrow} \ \hat{v}]) \ \wedge \ [\widehat{\mathcal{C}} \ \widehat{\Downarrow} \ \hat{v}] \models G \quad \Rightarrow \quad [\mathcal{C} \Downarrow v] \models G \tag{1.5}$$

where is $\preceq^{\widehat{\mathcal{R}}}$ is the simulation relation between concrete and abstract evaluations.

In this dissertation we abstract the concrete interpretation of the program by performing control flow analysis (CFA) to the program. CFA [Shi91, Pal95, JW94] is concerned with the computation of the closures that flow during the evaluation of a program. The result of a CFA can be viewed as an abstract derivation tree that contains all the paths of the original derivation tree. In its simplest version - the 0CFA - the concrete derivation tree is abstracted to the flow graph of the program. The constraints $l \models^{loc} G$ placed on the new tree will determine an analysis in the style of the usual data flow analysis monotone frameworks.

1.3 An Example

We show an example of the kind of transformations and program properties we are interested in. Let's consider the following program:

$$\begin{aligned}
e_0 = (\texttt{let}\quad & f_1 = (\texttt{fn}\, x, y \Rightarrow x)\\
& f_2 = (\texttt{fn}\, a, b \Rightarrow a)\\
& f_3 = (\texttt{fn}\, c, d \Rightarrow (*\ d\ \ d))\\
\texttt{in}\quad & (\texttt{let}\quad g = (\texttt{if}\ e_6\ \texttt{then}\ f_1\ \texttt{else}\ f_2)\\
& \qquad\quad h = (\texttt{if}\ e_{10}\ \texttt{then}\ f_1\ \texttt{else}\ f_3)\\
& \quad\texttt{in}\\
& \qquad\quad (g\ e_{15}\ \ (h\ e_{18}\ \ e_{19}))))
\end{aligned}$$

We would like to know what parts of the program contribute to the evaluation of e_0. A transformation can use that information and remove all the expressions that do not contribute to the final value of e_0. We want to make sure that the transformation preserves the meaning of the program, that is, the transformed program evaluates to the same value.

A quick inspection to the program will determine that only f_1 and f_2 flow to g in $(g\ \ e_{15}\quad (h\ \ e_{18}\quad e_{19}))$. We can also see that f_1 and f_2 only use their first parameter. Therefore, the second actual parameter $(h\ \ e_{18}\quad e_{19})$ is not needed. This will imply that, since h is only used in $(h\ e_{18}\ \ e_{19})$, h and $(\texttt{if}\ e_{10}\ \texttt{then}\ f_1\ \texttt{else}\ f_3)$ are not needed. Similarly, if we know that e_6 always evaluates to true then f_2 either.

Therefore, the original program can be replaced by:

$$\begin{aligned}
\mathcal{T}_{\mathcal{U}}(e_0) = (\texttt{let}\quad & f_1 = (\texttt{fn}\, x \Rightarrow x)\\
\texttt{in}\quad & (\texttt{let}\quad g = f_1\\
& \qquad \texttt{in}\quad (g\ \mathcal{T}_{\mathcal{U}}(e_{15}))))
\end{aligned}$$

We want to define an specification (program property)that determines which expressions can be removed without affecting the behavior of the program. Then, we can use this specfication to derive an analysis that can be used to compute such information. Furthermore, we want to prove that such definitions are correct. These are the main objectives of this dissertation.

1.4 Outline of the Dissertation

In chapter 2 we define the language syntax and semantics. Natural semantics derivation trees are introduced as our choice of representation for execution traces and proposed as the main medium to define program properties. The concept of (bi)simulations between derivation trees is defined as the main tool to express the correctness relations between the different evaluation relations defined in this dissertation. Some of these evaluation relations are defined as abstract interpretations of existent ones. We introduce the main concepts behind abstract interpretation and show that an abstract interpretation is a safe approximation of the concrete interpretation: both derivation trees form a simulation where nodes are related by safety relations (approximations).

We define program analyses as abstract interpretations of program property specifications. In particular, analyses are derived by performing control flow analyses to the standard natural semantics (and their derivation trees) used to define program properties. In chapter 3 we present the context-sensitive control and data flow analysis used in the rest of this dissertation and prove it correct with respect to the standard natural semantics. Correctness is expressed as a simulation result.

We apply the methodology proposed in this dissertation to prove the correctness of two non-trivial analysis-based program transformations. In chapter 4 we define a transformation that replaces useless expressions with dummy values in order to increase the performance of the program. In chapter 5, this transformation is refined to remove useless expressions and variables of a program. Both cases are presented as special cases of program slicing, a concept that reasons about parts of a program that do not affect its meaning. For each case we define, and prove correct, a verification system that checks valid program slices, *i.e.*, slices that define meaning-preserving slices. Then, a series of program analyses are derived from the slice specifications by performing control flow analysis as in chapter 3. The analyses are proven to simulate safely the specification and thus completing the correctness proof: information computed by the analysis is guaranteed to define meaning-preserving transformations.

Chapter 2

Background

In this chapter we present the theoretical basis and the proof machinery used in the presentation of the main results of this thesis. We first define the syntax and natural semantics of the source language. Most of the correctness proofs that accompany the theorems presented in this thesis are based on simulations of natural semantic trees, defined in section 2.2. The concepts of abstract interpretation defined in 2.3 are used by the remaining chapters as the main technique for deducing program analyses from the collecting semantics of a language (defined in chapter 3).

2.1 Source Language: Syntax and Semantics

The source language is an untyped lambda calculus extended with primitive functions and explicit operators for conditionals, `let`, and recursive procedures. The syntax of the source language is shown in Figure 2.1. Labels are used to identify the position of an expression in a program; an expression is a labeled term.

The semantics is a big-step call-by-value environment semantics. It deals with *configurations* consisting of an expression label l and an environment ρ mapping variables to values.

$$
\begin{array}{rcll}
l & \in & \mathit{Lab} & \text{(program labels)} \\
e & \in & \mathit{Exp} ::= t^l & \text{(expressions)} \\
\\
t & \in & \mathit{Term} & \text{(terms)} \\
t & ::= & c & [\mathbf{const}] \\
 & | & x & [\mathbf{var}] \\
 & | & f & [\mathbf{fn}] \\
 & | & (p\ e_1 \ldots e_n) & [\mathbf{app}_p] \\
 & | & (e_0\ e_1 \ldots e_n) & [\mathbf{app}] \\
 & | & (\texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & [\mathbf{if}] \\
 & | & (\texttt{let}\ x = e_1\ \texttt{in}\ e_2) & [\mathbf{let}] \\
\\
f & \in & \mathit{Fun} & \text{(function terms)} \\
f & ::= & (\texttt{fn}\ x_1 \ldots x_n \Rightarrow e_0) & \\
 & | & (\texttt{fun}\ y\ x_1 \ldots x_n \Rightarrow e_0) & \\
\\
p & \in & \mathit{Prim} & \text{(primitives)} \\
c, d & \in & \mathit{Const} & \text{(constants)} \\
x, y, z & \in & \mathit{Var} & \text{(variables)} \\
\bar{g} & \in & \mathit{Const}^n \to \mathit{Const} &
\end{array}
$$

$$
\begin{array}{rcll}
\Sigma \in \mathit{Univ} & & & \text{(universes)} \\
\mathcal{C}, \langle l, \rho \rangle \in \mathit{Config} & = & \mathit{Lab} \times \mathit{Env} & \text{(configurations)} \\
[l,\ \rho] \in \mathit{Closure} & = & \mathit{Lab} \times \mathit{Env} & \text{(closures)} \\
u, v, w \in \mathit{Val} & = & \mathit{Const} \cup \mathit{Closure} & \text{(values)} \\
\rho \in \mathit{Env} & = & \mathit{Var} \to \mathit{Val} & \text{(environments)}
\end{array}
$$

Figure 2.1: Syntax of the Source Language

Values are either constants c or closures consisting of a pair $[l, \rho]$, where l is the label of an abstraction.

We connect labels to expressions by hypothesizing a finite *universe* Σ of labeled terms, with the property that no label occurs more than once in Σ and that all subexpressions are consistently labeled. We write $\Sigma(l) = t$ if the unique occurrence of l in Σ labels the term t.

Let $subexp^*(e)$ - transitive, reflexive closure of *subexp* - be the set of subexpressions of e. Σ ranges over the set of consistent universes *Univ*:

$$Univ = \{\Sigma \in (Var \stackrel{fin}{\rightarrow} Term) \mid Consistent(\Sigma)\} \tag{2.1}$$

where

$$Consistent(\Sigma) \;\Leftrightarrow\; \begin{cases} \forall l \in dom(\Sigma).\ [\forall t^{l'} \in subexp^*(\Sigma(l)).\ \Sigma(l') = t] \\ \forall l, l' \in dom(\Sigma).\ \Sigma(l) = \Sigma(l') \Rightarrow (l = l') \end{cases} \tag{2.2}$$

This enables us to connect the semantics (which deals with expressions) to the program properties and analyses (which deal with labels). It is useful to think of Σ as a store containing the nodes of the parse trees of some expressions, and labels l as L-values pointing to those nodes. In some development environments, an object like Σ is called the "storage map." However, since labeled terms are acyclic, we can use induction on substructures of a labeled term as an induction principle.

We define $\mathcal{L}(\Sigma, l)$ as the set of labels associated to every subexpression of $\Sigma(l)$:

$$\begin{array}{rcl} \mathcal{L}(e) & = & \{l \mid t^l \in subexp^*(e)\} \\ \mathcal{L}(\Sigma, l) & = & \mathcal{L}(\Sigma(l)) \quad \text{where } \Sigma \in Univ \end{array} \tag{2.3}$$

Let $a_1{:}\ldots{:}a_n \in A^*$ be a sequence of elements of set A such that $\forall i \in \{1, \ldots, n\}.\ a_i \in A$. The empty sequence is denoted by ϵ. The :: operator is used to add an element to sequence:

$$a{::}(a_1{:}\ldots{:}a_n) = a{:}a_1{:}\ldots{:}a_n \qquad \wedge \qquad (a_1{:}\ldots{:}a_n){::}a = a_1{:}\ldots{:}a_n{:}a$$

Let $B \subseteq A$ and $s \in A^*$. The $s \backslash B$ operation, defined by:

$$
\begin{aligned}
\epsilon \backslash B &= \epsilon \\
(a{::}s) \backslash B &= \begin{cases} a{::}(s \backslash B) & \text{if } a \notin B \\ s \backslash B & \text{if } a \in B \end{cases}
\end{aligned}
\tag{2.4}
$$

removes all elements in sequence s that belong to B.

We define $\mathit{formals}(\Sigma, l) \in \mathit{Var}^*$ as the sequence of formal parameters of the function term labeled by l:

$$
\mathit{formals}(\Sigma, l) = \begin{cases} x_1{:}\ldots{:}x_n & \text{if } \Sigma(l) = (\texttt{fn}\ x_1 \ldots x_n \Rightarrow e_0) \\ x_1{:}\ldots{:}x_n & \text{if } \Sigma(l) = (\texttt{fun}\ y\ x_1 \ldots x_n \Rightarrow e_0) \\ \epsilon & \text{otherwise} \end{cases}
\tag{2.5}
$$

The semantics is then given by judgments $\Sigma \vdash C \Downarrow v$ defined by the system of Figure 2.2. Except for the systematic use of labels to refer to expressions in the universe, this is an ordinary call-by-value big-step environment semantics. Given $\Sigma \vdash \langle l, \rho \rangle \Downarrow v$, we shall assume that Σ consistently describes l and the closure information in ρ and v. We will refer this as the *standard semantics* of the language. This style is also known as the *natural semantics* [Kah87] of the language.

2.2 Natural Semantics Derivation Trees

A program trace shows a record of all the intermediate steps taken by a program before reaching its final state or producing a final value. Access to the trace of execution of a program enables us to reason about local and global behavior. These features make execution traces an ideal tool for the definition and manipulation of program properties.

A trace is *operational* in nature and its representation and structure depend on the type of operational semantics used. In the case of systems in a natural semantics framework, program execution resembles proof search. If we consider the operational semantics defined

$$\textbf{(r 1.1)} \quad \Sigma \vdash \langle l, \rho \rangle \Downarrow c \qquad \text{if } \Sigma(l) = c$$

$$\textbf{(r 1.2)} \quad \Sigma \vdash \langle l, \rho \rangle \Downarrow v \qquad \text{if } \Sigma(l) = x \ \wedge \ x \in \mathit{dom}(\rho) \ \wedge \ \rho(x) = v$$

$$\textbf{(r 1.3)} \quad \Sigma \vdash \langle l, \rho \rangle \Downarrow [l,\ \rho] \qquad \text{if } \Sigma(l) \in \mathit{Fun}$$

$$\textbf{(r 1.4)} \quad \frac{\Sigma(l) = (p\ t_1^{l_1} \ldots t_n^{l_n}) \ \wedge \ \Sigma \vdash \langle l_i, \rho \rangle \Downarrow c_i}{\Sigma \vdash \langle l, \rho \rangle \Downarrow \bar{p}(c_1 \ldots c_n)}$$

$$\textbf{(r 1.5)} \quad \frac{\begin{array}{l} \Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n}) \ \wedge \\ \Sigma \vdash \langle l_i, \rho \rangle \Downarrow v_i \quad i = 1, \ldots, n \ \wedge \\ \Sigma \vdash \langle l_0, \rho \rangle \Downarrow [l_f,\ \rho_0] \ \wedge \\ \Sigma(l_f) = (\texttt{fn}\ x_1 \ldots x_n \Rightarrow t_b^{l_b}) \ \wedge \\ \Sigma \vdash \langle l_b, \rho_0[x_1 \mapsto v_1] \ldots [x_n \mapsto v_n] \rangle \Downarrow v \end{array}}{\Sigma \vdash \langle l, \rho \rangle \Downarrow v}$$

$$\textbf{(r 1.6)} \quad \frac{\begin{array}{l} \Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n}) \ \wedge \\ \Sigma \vdash \langle l_i, \rho \rangle \Downarrow v_i \quad i = 1, \ldots, n \ \wedge \\ \Sigma \vdash \langle l_0, \rho \rangle \Downarrow [l_f,\ \rho_0] \ \wedge \\ \Sigma(l_f) = (\texttt{fun}\ y\ x_1 \ldots x_n \Rightarrow t_b^{l_b}) \ \wedge \\ \Sigma \vdash \langle l_b, \rho_0[y \mapsto \langle l_f, \rho_0 \rangle][x_1 \mapsto v_1] \ldots [x_n \mapsto v_n] \rangle \Downarrow v \end{array}}{\Sigma \vdash \langle l, \rho \rangle \Downarrow v}$$

$$\textbf{(r 1.7)} \quad \frac{\Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2}) \ \wedge \ \Sigma \vdash \langle l_0, \rho \rangle \Downarrow \mathit{true} \ \wedge \ \Sigma \vdash \langle l_1, \rho \rangle \Downarrow v}{\Sigma \vdash \langle l, \rho \rangle \Downarrow v}$$

$$\textbf{(r 1.8)} \quad \frac{\Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2}) \ \wedge \ \Sigma \vdash \langle l_0, \rho \rangle \Downarrow \mathit{false} \ \wedge \ \Sigma \vdash \langle l_2, \rho \rangle \Downarrow v}{\Sigma \vdash \langle l, \rho \rangle \Downarrow v}$$

$$\textbf{(r 1.9)} \quad \frac{\Sigma(l) = (\texttt{let}\ x = t_1^{l_1}\ \texttt{in}\ t_2^{l_2}) \ \wedge \ \Sigma \vdash \langle l_1, \rho \rangle \Downarrow v_1 \ \wedge \ \Sigma \vdash \langle l_2, \rho[x \mapsto v_1] \rangle \Downarrow v}{\Sigma \vdash \langle l, \rho \rangle \Downarrow v}$$

Figure 2.2: Operational Semantics of the Source Language

in section 2.1, we can use the derivation tree generated to prove $\Sigma \vdash \mathcal{C} \Downarrow v$ as the execution trace of $\mathcal{C}$.

We generalize the notion of derivation trees (called natural semantics trees in [Sch95]) for a relation E defined by a set of rule schemes $Rules_E$. Let $\mathcal{N}$ denote possible elements of E and $Rules_E$ be a set of rules of the form:

$$\frac{\{\mathcal{N}_i \in E\}_{i \in 1,\ldots,n} \quad \Psi_{\mathbf{rule}}}{\mathcal{N} \in E}\ \mathbf{rule} \tag{2.6}$$

where $\Psi_{\mathbf{rule}}$ denotes a set of extra constraints between the elements in the antecedent and consequent. Axioms are defined for $n = 0$.

We define the domain of derivation trees induced by $Rules_E$ as the set of trees of nodes of the form $\mathcal{N} \in E$. The root and the set of immediate subtrees (antecedents) of a derivation tree can be accessed by the *root* and *reach* functions respectively. We write $[\mathcal{N} \in E]$ to denote a tree with root $\mathcal{N} \in E$, *i.e.*, $root([\mathcal{N} \in E]) = \mathcal{N} \in E$.

We write $[\mathcal{N} \in E]$ to denote a derivation tree generated by a proof of $\mathcal{N} \in E$. The set of well-formed derivation trees induced by $Rules_E$ is defined by the set of rules $Rules_E^{\mathbf{t}}$, where each rule in $Rules_E$ is modified to handle derivation trees. For example, the rule shown in 2.6 is transformed to:

$$\frac{\{[\mathcal{N}_i \in E]\}_{i \in 1,\ldots,n} \quad \Psi_{\mathbf{rule}}}{[\mathcal{N} \in E]}, \quad \begin{array}{l} root([\mathcal{N} \in E]) = \mathcal{N} \in E \\ reach([\mathcal{N} \in E]) = \{[\mathcal{N}_i \in E]\}_{i \in 1,\ldots,n} \end{array} \tag{2.7}$$

that is, $[\mathcal{N} \in E]$ is a valid derivation tree if and only if its subtrees $\{[\mathcal{N}_i \in E]\}_{i \in 1,\ldots,n}$ (predecessors) are valid derivation trees and $\Psi_{\mathbf{rule}}$ is true. We write $[\mathcal{N} \in E]$ as an assertion that such a tree is well-formed. Therefore $\forall \mathcal{N},\ [\mathcal{N} \in E] \Leftrightarrow \mathcal{N} \in E$.

$$\frac{\{[\mathcal{N}_i \in E]\}_{i \in 1,\ldots,n} \quad \Psi_{\mathbf{rule}}}{[\mathcal{N} \in E]}\ \mathbf{rule}^{\mathbf{t}} \tag{2.8}$$

An inductive (co-inductive) definition of $[\mathcal{N} \in E]$ follows an inductive (co-inductive) definition of E. A co-inductive definition of $[\mathcal{N} \in E]$ allows the modeling of infinite derivation trees derived from natural semantics defined as greatest fixed points e.g. non-terminating programs, analyses.

In the case of the relation defined by $\Sigma \vdash \mathcal{C} \Downarrow v$ - configuration $\mathcal{C}$ evaluates to v under universe Σ - we write $[\Sigma \vdash \mathcal{C} \Downarrow v]$ to denote a derivation tree generated by $\Sigma \vdash \mathcal{C} \Downarrow v$. It is important to note that we can have more than one derivation tree generated by the evaluation of a single configuration if the semantics contains non-deterministic elements, even if the evaluated value is the same.

We show an example of $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$. If $\Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})$ and $[\Sigma \vdash \langle l_0, \rho\rangle \Downarrow \mathit{true}]$ then by applying the rule $(r\ 1.7)$

$$\frac{[\Sigma \vdash \langle l_0, \rho\rangle \Downarrow \mathit{true}]\ \wedge\ [\Sigma \vdash \langle l_1, \rho\rangle \Downarrow v]}{[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]}$$

then $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$ contains as predecessors subtrees denoted by $[\Sigma \vdash \langle l_0, \rho\rangle \Downarrow \mathit{true}]$ and $[\Sigma \vdash \langle l_1, \rho\rangle \Downarrow v]$.

We define reach^* as the reflexive transitive closure of reach. We say that a node - or subtree - is reachable from $\mathcal{N} \in E$ if it appears in the derivation tree - or proof tree - generated by $\mathcal{N} \in E$, *i.e.*, $[\mathcal{N} \in E]$.

Definition 2.2.1 (Reachability)

- $[\mathcal{N}' \in E]$ *is reachable from* $[\mathcal{N} \in E]$ *if and only if* $[\mathcal{N}' \in E] \in \mathit{reach}^*([\mathcal{N} \in E])$

2.2.1 Relating Execution Traces

We introduce the concepts of simulation and bi-simulation between the derivation trees of two different natural semantics. Given $[\mathcal{N} \in E]$ and $[\mathcal{N}_s \in E_s]$ derivation trees defined by two natural semantics, we say that $[\mathcal{N} \in E]$ simulates $[\mathcal{N}_s \in E_s]$ if both roots are related - in a way that we leave unspecified - and if each immediate derivation tree of $[\mathcal{N}]$[1]

[1] We will write $[\mathcal{N}]$ instead of $[\mathcal{N} \in E]$ whenever it's clear from the context.

- predecessors - is simulated by an immediate derivation tree of $[\mathcal{N}_s]$, or if $[\mathcal{N}]$ is simulated by an immediate derivation tree of $[\mathcal{N}_s]$. We define $\preceq^{\mathcal{R}}$, the simulation induced by $\mathcal{R}$, as the greatest relation that satisfies the following rules:

$$\begin{array}{rcl} [\mathcal{N}] \;\; \preceq^{\mathcal{R}} \;\; [\mathcal{N}_s] & \Leftrightarrow & \mathcal{N} \; \mathcal{R} \; \mathcal{N}_s \quad \wedge \\ & & \left(\begin{array}{c} \forall [\mathcal{N}'] \in reach([\mathcal{N}]),\ \exists [\mathcal{N}'_s] \in reach([\mathcal{N}_s]) : [\mathcal{N}'] \preceq^{\mathcal{R}} [\mathcal{N}'_s] \\ \text{or} \\ \exists [\mathcal{N}'_s] \in reach([\mathcal{N}_s]) : [\mathcal{N}] \preceq^{\mathcal{R}} [\mathcal{N}'_s] \end{array} \right) \end{array} \tag{2.9}$$

where the definition of $\mathcal{N} \;\; \mathcal{R} \;\; \mathcal{N}_s$ is specific for each case. Similarly we say that $[\mathcal{N}]$ and $[\mathcal{N}_s]$ are bisimilar if both roots are related - again, we leave this part unspecified - and if each immediate derivation tree of $[\mathcal{N}]$ is bi-simulated by an immediate derivation tree of $[\mathcal{N}_s]$, and vice-versa. We define $\cong^{\mathcal{R}}$, the bisimulation induced by $\mathcal{R}$, as the greatest relation that satisfies the following rules:

$$\begin{array}{rcl} [\mathcal{N}] \;\; \cong^{\mathcal{R}} \;\; [\mathcal{N}_s] & \Leftrightarrow & \mathcal{N} \; \mathcal{R} \; \mathcal{N}_s \quad \wedge \\ & & \forall [\mathcal{N}'] \in reach([\mathcal{N}]),\ \exists [\mathcal{N}'_s] \in reach[\mathcal{N}_s] : [\mathcal{N}'] \cong^{\mathcal{R}} [\mathcal{N}'_s] \\ & & \forall [\mathcal{N}'_s] \in reach[\mathcal{N}_s],\ \exists [\mathcal{N}'] \in reach([\mathcal{N}]) : [\mathcal{N}'] \cong^{\mathcal{R}} [\mathcal{N}'_s] \end{array} \tag{2.10}$$

where the definition of $\mathcal{N} \;\; \mathcal{R} \;\; \mathcal{N}_s$ is specific for each case.

We present most correctness theorems as simulations and bi-simulations between the derivation of different natural semantics. Whenever we define a new simulation $\preceq^{\mathcal{R}}$ or bi-simulation $\cong^{\mathcal{R}}$, we only need to define the $\mathcal{N} \;\; \mathcal{R} \;\; \mathcal{N}_s$ relation and assume that the rest of the definition follows 2.9 and 2.10, respectively, unless stated otherwise.

2.3 Abstract Interpretation

Abstract interpretation is a formal-semantics-based methodology for deriving provably correct program analyses. The main idea, formalized by Cousot and Cousot [CC77], is that a program can be given different interpretations by modifying the domain of values that it manipulates. The concrete interpretation, defined by the standard semantics, gives exact information about the behavior of a program. When the initial set of values is replaced by the new set of abstract values and the derivation (evaluation) rules are modified accordingly (see Figure 2.3), the abstract interpretation calculates information that safely approximates the behavior of the concrete interpretation. The goal of an abstract interpretation is to provide a less computationally expensive version of the original semantics while keeping the information regarding the program properties of interest as precise as possible. In order to achieve this, the choice of abstract values is essential.

We denote the new domain of abstract values with the complete lattice $(\widehat{Val}, \sqsubseteq)$, where the order relation $\sqsubseteq$ defined on $\widehat{Val}$ is used as an approximation relation [CC92]. The definition of $\widehat{Val}$ depends on what properties we want to keep track of and what information we are willing to abstract away. As an example, we follow a common technique and abstract the constant values of the language to a single value - which we denote by $\bullet$. In other words, constants are ignored and, as we will see next, only control flow information is considered.

Next, we have to decide how to represent abstract closures and abstract environments (we denote the latter by $\hat{\rho}$). In order to do this let's have a look at one of the evaluation rules. The rule corresponding to `if`-expressions initially looks like:

$$\frac{\Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})\ \wedge\ \Sigma \vdash \langle l_0, \hat{\rho}\rangle \hat{\Downarrow} \hat{v}_i \quad i = 1,2,3}{\Sigma \vdash \langle l, \hat{\rho}\rangle \hat{\Downarrow} (\hat{v}_1 \sqcup \hat{v}_2)}$$

In the standard semantics, the value resulting from the evaluation of the conditional expression e_0 is used to decide which branch should be evaluated next. In the new semantics, constant values are abstracted to $\bullet$ and this information is lost. We choose to evaluate both branches and merge the results with the l.u.b.[2] operator $\sqcup$. For example, if we have closures with functional labels l_1 and l_2 flowing from each branch then the result of evaluating the `if`-expression should consider both closures. An abstract closure should be able to contain information of more than one closure.

[2]least upper bound.

(r 2.1) $\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \{\bullet\} \qquad \text{if } \Sigma(l) = c$

(r 2.2) $\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \hat{\rho}(x) \quad \text{if } \Sigma(l) = x \ \wedge \ x \in dom(\hat{\rho})$

(r 2.3) $\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \{l\} \qquad \text{if } \Sigma(l) \in Fun$

(r 2.4) $$\frac{\Sigma(l) = (p\ t_1^{l_1} \ldots t_n^{l_n}) \ \wedge \ \Sigma \vdash \langle l_i, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}_i}{\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \{\bullet\}}$$

(r 2.5) $$\frac{\begin{array}{l} \Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n}) \ \wedge \\ \Sigma \vdash \langle l_i, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}_i \quad i = 0, \ldots, n \ \wedge \\ \begin{array}{c} \forall\, l_j \in \hat{v}_0 \\ j = 1, \ldots, m \end{array} : \begin{cases} \Sigma(l_j) = (\texttt{fn}\ x_1 \ldots x_n \Rightarrow t_b^{l_b}) : \\ \quad \begin{cases} \Sigma \vdash \langle l_b, \hat{\rho} \rangle \hat{\Downarrow} \hat{w}_j \\ \hat{v}_i \subseteq \hat{\rho}(x_i) \quad i = 1, \ldots, n \end{cases} \\ \Sigma(l_j) = (\texttt{fun}\ y\ x_1 \ldots x_n \Rightarrow t_b^{l_b}) : \\ \quad \begin{cases} \Sigma \vdash \langle l_b, \hat{\rho} \rangle \hat{\Downarrow} \hat{w}_j \\ \hat{v}_i \subseteq \hat{\rho}(x_i) \quad i = 1, \ldots, n \\ l_j \in \hat{\rho}(y) \end{cases} \end{cases} \end{array}}{\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} (\hat{w}_1 \cup \ldots \cup \hat{w}_m)}$$

(r 2.6) $$\frac{\Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2}) \ \wedge \ \Sigma \vdash \langle l_i, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}_i \quad i = 1, 2, 3 \ \wedge}{\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} (\hat{v}_1 \cup \hat{v}_2)}$$

(r 2.7) $$\frac{\Sigma(l) = (\texttt{let}\ x = t_1^{l_1}\ \texttt{in}\ t_2^{l_2}) \ \wedge \ \Sigma \vdash \langle l_i, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}_i \quad i = 1, 2 \ \wedge \ \hat{v}_1 \subseteq \hat{\rho}(x)}{\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}_2}$$

Figure 2.3: An Abstract Semantics of the Source Language

The desired functionality is obtained by working with sets and using $\subseteq$ as the ordering relation for $\widehat{Val}$ and $\cup$ as the l.u.b. operator. The domain of abstract values and closures is defined by:

$$\begin{array}{rcl} \hat{v}, \hat{w} \in \widehat{Val} & = & \mathcal{P}(\{\bullet\} \cup Lab) \\ \widehat{Closure} & = & \mathcal{P}(Lab) \end{array}$$

where $\bullet$'s and labels are allowed to coexist in the same value because an expression might evaluate to a constant and a closure in the same program (this is an untyped language).

Next, let's consider the evaluation of a `let`-expression:

$$\frac{\Sigma(l) = (\texttt{let}\ x = t_1^{l_1}\ \texttt{in}\ t_2^{l_2}) \ \wedge\ \Sigma \vdash \langle l_1, \hat{\rho}\rangle \Downarrow \hat{v}_1 \ \wedge\ \Sigma \vdash \langle l_2, \hat{\rho}'\rangle \Downarrow \hat{v}}{\Sigma \vdash \langle l, \hat{\rho}\rangle \Downarrow \hat{v}} \quad \hat{\rho}' = ?$$

We have to decide the mechanism for variable binding - $\hat{\rho}'$ - in the new semantics. One option is to write $\hat{\rho}' = \hat{\rho}[x \mapsto \hat{v}_1]$, which would lead to a polyvariant analysis. Another solution is to work with a single global environment, *i.e.*, make $\hat{\rho}' = \hat{\rho}$ and add the constraint $\hat{v}_1 \subseteq \hat{\rho}(x)$. This means that the abstract interpretation will fail for certain environments, *i.e.*, $\hat{\rho}$ too small. This approach is known as 0CFA and assumes that all expressions of the program are evaluated in the same context or environment. It makes the analysis much simpler - one configuration per label - and still provides useful information for optimizations.

Now we can give a complete definition of the abstract domains:

$$\begin{array}{rcl} \hat{C} \in \widehat{Config} & = & Lab \times \widehat{Env} \\ \hat{\rho} \in \widehat{Env} & = & Var \to \widehat{Val} \end{array}$$

The abstract semantics of the language is given by the judgement $\Sigma \vdash \langle l, \rho\rangle \hat{\Downarrow} \hat{v}$ defined by the rules of Figure 2.3. We emphasize the fact that the evaluation relation is defined as the greatest fixed-point of the specification. Therefore, proofs will be based on co-induction. This is an important point because programs that successfully terminate[3] may not do so in the abstract semantics if defined inductively. Such is the case for:

$$\Sigma(l) = ((\texttt{fun}\ f\ x \Rightarrow (\texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ (f\ (dec\ x)^{l_2})^{l_b}))\ 3)$$

[3] Where termination means proving $\Sigma \vdash \langle l, \rho\rangle \Downarrow v$ for some v.

where, in order to prove $\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}$ we need to prove $\Sigma \vdash \langle l_b, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}_b$, for some $\hat{v}_b$, which in turn requires - since f is applied recursively - a proof for $\Sigma \vdash \langle l_b, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}_b$. This apparent never ending circularity can be solved by using co-induction.

We formalize the intuitive relation between concrete and abstract values by defining a representation function [NNH99, Section 4.1.2]. A representation function $\beta : \mathit{Val} \rightarrow \widehat{\mathit{Val}}$ maps a concrete value to the best abstract value describing it. We define β for values and extend it to deal with configurations and environments:

$$\beta : \mathit{Config} \rightarrow \widehat{\mathit{Config}} \quad \beta(\langle l, \rho \rangle) \;\; = \langle l, \beta(\rho) \rangle$$

$$\begin{array}{lrl} \beta : \mathit{Val} \rightarrow \widehat{\mathit{Val}} & \beta(c) & = \{\bullet\} \\ & \beta([l,\, \rho]) & = \{l\} \end{array}$$

$$\begin{array}{lrl} \beta : \mathit{Env} \rightarrow \widehat{\mathit{Env}} & \beta(\rho) = & \{(x, \hat{v}) \mid x \in \mathit{dom}(\rho) \;\wedge\; \hat{v} = \beta(\rho(x))\} \;\cup \\ & & \{\bigcup \beta(\rho') \mid x \in \mathit{dom}(\rho) \;\wedge\; \rho(x) = [l,\, \rho']\} \end{array}$$

A global abstract environment must correctly describe all environments used during the evaluation of a program. Therefore, the representation function $\beta(\rho)$ considers all nested environments in ρ.

2.3.1 Correctness of an Abstract Interpretation

An abstract interpretation is correct if it safely approximates the concrete interpretation. The definition of "safe approximation" depends on the representation functions and the ordering relation $\sqsubseteq$ defined for the interpretation. The abstract value $\hat{v}$ safely approximates v if and only if $\beta(v) \sqsubseteq \hat{v}$. In general, the following must be proven:

$$\beta(\mathcal{C}) \sqsubseteq \widehat{\mathcal{C}} \;\;\wedge\;\; \Sigma \vdash \mathcal{C} \Downarrow v \;\;\wedge\;\; \Sigma \vdash \widehat{\mathcal{C}} \hat{\Downarrow} \hat{v} \quad \Rightarrow \quad \beta(v) \sqsubseteq \hat{v} \tag{2.11}$$

The definition of $\sqsubseteq$ for our choice of abstract domains is straightforward:

$$\begin{aligned}
\hat{v}_1 \sqsubseteq \hat{v}_2 \quad &\Leftrightarrow \quad \hat{v}_1 \subseteq \hat{v}_2 \\
\langle l, \hat{\rho} \rangle \sqsubseteq \langle l', \hat{\rho}' \rangle \quad &\Leftrightarrow \quad l = l' \ \wedge \ \hat{\rho} \sqsubseteq \hat{\rho}' \\
\hat{\rho} \sqsubseteq \hat{\rho}' \quad &\Leftrightarrow \quad \forall x \in dom(\hat{\rho}).\ \hat{\rho}(x) \subseteq \hat{\rho}'(x)
\end{aligned}$$

The current abstract semantics considers that all closures present in an abstract interpretation of an expression or program share the same context or abstract environment. The notation takes advantage of this and does not include environments in closures - one may consider $\Sigma \vdash \langle l, \hat{\rho} \rangle \ \hat{\Downarrow} \ \{[l,\ \hat{\rho}]\}$ or $\Sigma \vdash \langle l, \hat{\rho} \rangle \ \hat{\Downarrow} \ (\{l\}, \hat{\rho})$ as more faithful versions. However, a correct safety relation must consider the abstract - global - environments. In fact, we need to prove a stronger result:

$$\forall \mathcal{C}, l, \hat{\rho}, v, \hat{v} :$$

$$\beta(\mathcal{C}) \sqsubseteq \langle l, \hat{\rho} \rangle \ \wedge \ \Sigma \vdash \mathcal{C} \Downarrow v \ \wedge \ \Sigma \vdash \langle l, \hat{\rho} \rangle \ \hat{\Downarrow} \ \hat{v} \ \Rightarrow \ \begin{array}{l} \beta(v) \sqsubseteq \hat{v} \\ v = [l',\ \rho'] \Rightarrow \beta(\rho') \sqsubseteq \hat{\rho} \end{array} \tag{2.12}$$

The second line - $\beta(\rho') \sqsubseteq \hat{\rho}$ - relates a closure to the abstract global environment. As we mentioned before, the (single) abstract global environment must correctly describe all environments used during the evaluation of the program. On the practical side, this condition will also be needed to preserve the initial conditions during an inductive proof where $[l',\ \rho']$ flows to the operator side of an application.

Alternatively, the correctness criterion can be presented more succinctly by a series of safety relations between the concrete and abstract elements of the interpretations [NN97], as follows. We define the relations $\mathcal{R}$, for environments and configurations, and $\mathcal{V}$, for values, as follows:

$$\begin{aligned}
v \ \mathcal{V}_{\hat{\rho}} \ \hat{v} \quad &\Leftrightarrow \quad \forall l_0, \rho_0, c : \\
& \qquad v = [l_0,\ \rho_0] \Rightarrow (l_0 \in \hat{v}) \ \wedge \ \rho_0 \ \mathcal{R} \ \hat{\rho} \\
& \qquad v = c \Rightarrow (\bullet \in \hat{v}) \\
\rho \ \mathcal{R} \ \hat{\rho} \quad &\Leftrightarrow \quad \forall x \in dom(\rho) : \rho(x) \ \mathcal{V}_{\hat{\rho}} \ \hat{\rho}(x) \\
\langle l, \rho \rangle \ \mathcal{R} \ \langle l', \hat{\rho} \rangle \quad &\Leftrightarrow \quad l = l' \ \wedge \ \rho \ \mathcal{R} \ \hat{\rho}
\end{aligned} \tag{2.13}$$

The safety relations must encode the composition of $\beta(-)$ and the $\sqsubseteq$ ordering, *i.e.*, $(a\ \mathcal{R}\ \hat{a}) \Leftrightarrow \beta(a) \sqsubseteq \hat{a}$. They also take into account the relation between environments defined in 2.12. In 2.13, the abstract value $\hat{v}$ safely approximates the closure $\langle l, \rho\rangle$ if and only if $l \in \hat{v}$ and ρ is safely approximated by the global abstract environment. Similarly, abstract environment $\hat{\rho}$ safely approximates ρ if each value mapped to a variable in the domain of ρ is safely approximated by its counterpart in $\hat{\rho}$. Two configurations are related if they share the same label and their environments are related. The correctness criterion defined in 2.12 is rewritten in terms of these safety relations:

$$\langle l, \rho\rangle\ \mathcal{R}\ \langle l', \hat{\rho}\rangle \quad \wedge \quad \Sigma \vdash \langle l, \rho\rangle \Downarrow v \quad \wedge \quad \Sigma \vdash \langle l', \hat{\rho}\rangle \hat{\Downarrow} \hat{v} \quad \Rightarrow \quad v\ \mathcal{V}_{\hat{\rho}}\ \hat{v} \tag{2.14}$$

This says that the safety relation must be preserved under evaluation. This notion of correctness leads to a stronger safety result: the abstract interpretation provides a safe simulation of the concrete interpretation [Sch95]. Every node - and consequently every path - of the derivation tree of the concrete evaluation is safely approximated by a node in the abstract interpretation.

The safety relation $\mathcal{R}$ between concrete and abstract nodes is defined by:

$$(\Sigma \vdash \langle l, \rho\rangle \Downarrow v)\ \mathcal{R}\ (\Sigma' \vdash \langle l', \hat{\rho}\rangle \hat{\Downarrow} \hat{v}) \quad \Leftrightarrow \quad \begin{cases} \Sigma = \Sigma' \quad \wedge \quad \langle l, \rho\rangle\ \mathcal{R}\ \langle l', \hat{\rho}\rangle \\ v\ \mathcal{V}_{\hat{\rho}}\ \hat{v} \end{cases} \tag{2.15}$$

Let $\preceq^{\mathcal{R}}$ be the simulation relation between concrete and abstract trees induced by $\mathcal{R}$ (2.9). Let $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$ and $[\Sigma \vdash \langle l, \hat{\rho}\rangle \Downarrow \hat{v}]$ denote the derivation trees of $\Sigma \vdash \langle l, \rho\rangle \Downarrow v$ and $\Sigma \vdash \langle l, \hat{\rho}\rangle \Downarrow \hat{v}$ respectively. We must prove the following:

Theorem 2.3.1 (Correctness of the Abstract Interpretation)
For all $l, \rho, v, \hat{\rho}, \hat{v}$: if $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$, $[\Sigma \vdash \langle l, \hat{\rho}\rangle \Downarrow \hat{v}]$ and $\langle l, \rho\rangle\ \mathcal{R}\ \langle l, \hat{\rho}\rangle$ then:

$$[\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l, \hat{\rho}\rangle \Downarrow \hat{v}]$$

Proof: *By rule induction on $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$.*

We know $\langle l, \rho \rangle \ \mathcal{R} \ \langle l, \hat{\rho} \rangle$ *and* $\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}$. *We consider each of the rules of figure 2.2 for* $\Sigma \vdash \langle l, \rho \rangle \Downarrow v$.

The first three rules correspond to axioms. Given $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v]$, $[\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}]$ *and* $\langle l, \rho \rangle \ \mathcal{R} \ \langle l, \hat{\rho} \rangle$ *we only need to show* $(v \ \mathcal{V}_{\hat{\rho}} \ \hat{v})$. *This is enough to prove*

$$(\Sigma \vdash \langle l, \rho \rangle \Downarrow v) \ \mathcal{R} \ (\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}) \qquad \textit{eq. (2.15)}$$

Given that $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v]$ *has no subtrees, we can easily derive*

$$[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}]$$

The axioms are:

(**r 1.1**) $\Sigma(l) = c$ *and* $v = c$.

Given $\Sigma(l) = c$, $[\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}]$ *can only be the result of applying rule* (**r 2.1**) *where* $\hat{v} = \{\bullet\}$. *Clearly* $(c \ \mathcal{V}_{\hat{\rho}} \ \{\bullet\})$, *as defined by equation (2.13). Therefore* $(v \ \mathcal{V}_{\hat{\rho}} \ \hat{v})$.

(**r 1.2**) $\Sigma(l) = x$, $x \in dom(\rho)$ *and* $\rho(x) = v$.

Given $\Sigma(l) = x$, $[\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}]$ *can only be the result of applying rule* (**r 2.2**) *to* $x \in dom(\hat{\rho})$ *and deriving* $\hat{v} = \hat{\rho}(x)$. *Using* $\langle l, \rho \rangle \ \mathcal{R} \ \langle l, \hat{\rho} \rangle$, $x \in dom(\rho)$ *and* $x \in dom(\hat{\rho})$ *we derive* $(\rho(x) \ \mathcal{V}_{\hat{\rho}} \ \hat{\rho}(x))$, *i.e.*, $(v \ \mathcal{V}_{\hat{\rho}} \ \hat{v})$.

(**r 1.3**) $\Sigma(l) \in \mathit{Fun}$ *and* $v = [l, \ \rho]$.

$[\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}]$ *must be the result of applying rule* (**r 2.3**) *to* $\Sigma(l) \in \mathit{Fun}$ *where* $\hat{v} = \{l\}$. *Clearly* $([l, \ \rho] \ \mathcal{V}_{\hat{\rho}} \ \{l\})$, *i.e.*, $(v \ \mathcal{V}_{\hat{\rho}} \ \hat{v})$, *as defined by equation (2.13).*

The rest of the rules correspond to derivation trees with height greater than one (non-axioms). We need to find, for each subtree $[\Sigma \vdash \langle l', \rho' \rangle \Downarrow v']$ *of* $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v]$, *a subtree* $[\Sigma \vdash \langle l', \hat{\rho} \rangle \hat{\Downarrow} \hat{v}']$ *of* $[\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}]$ *such that*

$$[\Sigma \vdash \langle l', \rho' \rangle \Downarrow v'] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l', \hat{\rho} \rangle \hat{\Downarrow} \hat{v}']$$

We complete the proof by showing that $(v \ \mathcal{V}_{\hat{\rho}} \ \hat{v})$, *which is used to show*

$$(\Sigma \vdash \langle l, \rho \rangle \Downarrow v) \ \mathcal{R} \ (\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \hat{v})$$

The following result will be of great use:

$$\forall v, \hat{v}, \hat{v}'. \ (v \ \mathcal{V}_{\hat{\rho}} \ \hat{v}) \ \wedge \ \hat{v} \subseteq \hat{v}' \Rightarrow (v \ \mathcal{V}_{\hat{\rho}} \ \hat{v}') \quad [\mathbf{f\ 1.1}]$$

which is a direct application of the definition of $\mathcal{V}_{\hat{\rho}}$ *in equation (2.13).*

The rest of the rules go as follows:

(r 1.4) $\Sigma(l) = (p \ t_1^{l_1} \dots t_n^{l_n})$ *and:*

$$\Sigma \vdash \langle l_i, \rho \rangle \Downarrow c_i \ \wedge \ v = \bar{p}(c_1 \dots c_n) \in \mathit{Const}$$

$[\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}]$ *must be the result of applying rule* (**r 2.4**) *to* $\Sigma(l) = (p \ t_1^{l_1} \dots t_n^{l_n})$ *where*

$$i = 1, \dots, n : \quad [\Sigma \vdash \langle l_i, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}_i] \quad \wedge \quad \hat{v} = \{\bullet\}$$

Using equation **2.15** *and* $(\rho \ \mathcal{R} \ \hat{\rho})$ *we derive* $\langle l_i, \rho \rangle \ \mathcal{R} \ \langle l_i, \hat{\rho} \rangle$, *for* $i = 1, \dots, n$. *Then, we can apply the inductive hypothesis to each* $\Sigma \vdash \langle l_i, \rho \rangle \Downarrow c_i$ *and derive*

$$i = 1, \dots, n : \quad [\Sigma \vdash \langle l_i, \rho \rangle \Downarrow c_i] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l_i, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}_i]$$

Using equation (2.13) we can easily check $(v \ \mathcal{V}_{\hat{\rho}} \ \{\bullet\})$, *i.e.,* $(v \ \mathcal{V}_{\hat{\rho}} \ \hat{v})$.

(r 1.5) $\Sigma(l) = (t_0^{l_0} \ t_1^{l_1} \dots t_n^{l_n})$ *and:*

$$\begin{aligned}
&[\Sigma \vdash \langle l_i, \rho \rangle \Downarrow v_i] \quad i = 1, \dots, n \ \wedge \\
&[\Sigma \vdash \langle l_0, \rho \rangle \Downarrow [l_f, \ \rho_0]] \ \wedge \\
&\Sigma(l_f) = (\texttt{fn}\, x_1 \dots x_n \Rightarrow t_b^{l_b}) \ \wedge \\
&[\Sigma \vdash \langle l_b, \rho_0' \rangle \Downarrow v]
\end{aligned}$$

where $\rho_0' = \rho_0[x_1 \mapsto v_1] \dots [x_n \mapsto v_n]$.

$[\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}]$ *must be the result of applying rule* $(\mathbf{r\ 2.5})$ *to* $\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n})$ *where:*

$$[\Sigma \vdash \langle l_i, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}_i] \quad i = 0, \ldots, n \ \wedge$$
$$\begin{array}{c} \forall\, l_j \in \hat{v}_0 \\ j = 1, \ldots, m \end{array} : \begin{cases} \Sigma(l_j) = (\mathtt{fn}\, x_1 \ldots x_n \Rightarrow t_b^{l_b}) : \quad [\mathbf{f\ 1.2}] \\ \begin{cases} [\Sigma \vdash \langle l_b, \hat{\rho} \rangle \hat{\Downarrow} \hat{w}_j] \\ \hat{v}_i \subseteq \hat{\rho}(x_i) \quad i = 1, \ldots, n \end{cases} \\ \Sigma(l_j) = (\mathtt{fun}\, y\ x_1 \ldots x_n \Rightarrow t_b^{l_b}) : \\ \quad \ldots \end{cases}$$
$$\hat{v} = (\hat{w}_1 \cup \ldots \cup \hat{w}_m) \quad [\mathbf{f\ 1.3}]$$

We need to prove the following:

$$(v\ \mathcal{V}_{\hat{\rho}}\ \hat{v}) \text{ which implies } (\Sigma \vdash \langle l, \rho \rangle \Downarrow v)\ \mathcal{R}\ (\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}) \quad \text{eq. (2.15)}$$
$$[\Sigma \vdash \langle l_0, \rho \rangle \Downarrow [l_f,\ \rho_0]] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l_0, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}_0]$$
$$[\Sigma \vdash \langle l_i, \rho \rangle \Downarrow v_i] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l_i, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}_i] \qquad i = 1, \ldots, n$$
$$[\Sigma \vdash \langle l_b, \rho_0' \rangle \Downarrow v] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l_b, \hat{\rho} \rangle \hat{\Downarrow} \hat{w}_j]$$

Using $(\rho\ \mathcal{R}\ \hat{\rho})$ *and equation (2.13) we get* $\langle l_i, \rho \rangle\ \mathcal{R}\ \langle l_i, \hat{\rho} \rangle$, *for* $= i, \ldots, n$. *Then we can apply the inductive hypothesis to* $[\Sigma \vdash \langle l_i, \rho \rangle \Downarrow v_i]$ *and derive:*

$$[\Sigma \vdash \langle l_0, \rho \rangle \Downarrow [l_f,\ \rho_0]] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l_0, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}_0]$$
$$[\Sigma \vdash \langle l_i, \rho \rangle \Downarrow v_i] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l_i, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}_i] \qquad i = 1, \ldots, n$$

Now we need to prove that there exists a subtree $[\Sigma \vdash \langle l', \hat{\rho} \rangle \hat{\Downarrow} \hat{v}']$ *of* $[\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}]$ *such that* $[\Sigma \vdash \langle l_b, \rho_0' \rangle \Downarrow v] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l', \hat{\rho} \rangle \hat{\Downarrow} \hat{v}']$.

By applying the definitions of $\preceq^{\mathcal{R}}$ *and* $\mathcal{R}$ *to* $[\Sigma \vdash \langle l_0, \rho \rangle \Downarrow [l_f,\ \rho_0]] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l_0, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}_0]$ *we get* $[l_f,\ \rho_0]\ \mathcal{V}_{\hat{\rho}}\ \hat{v}_0$. *Therefore, according to equation (2.13), we have :*

$$l_f \in \hat{v}_0 \quad \wedge \quad (\rho_0\ \mathcal{R}\ \hat{\rho})$$

This means that, using $l_f \in \hat{v}_0$, *we can instantiate* $[\mathbf{f\ 1.2}]$ *- from the conditions of* $[\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}]$ *mentioned above - with* $l_j = l_f$ *for some* $j \in \{1, \ldots, n\}$, *where:*

$$\Sigma(l_j) = (\mathtt{fn}\, x_1 \ldots x_n \Rightarrow t_b^{l_b}) : \quad [\Sigma \vdash \langle l_b, \hat{\rho} \rangle \hat{\Downarrow} \hat{w}_j] \quad \wedge \quad \hat{v}_i \subseteq \hat{\rho}(x_i) \quad i = 1, \ldots, n$$

and $\hat{w}_j \subseteq \hat{v}$ *(using* [**f 1.3**]*).*

Before applying the inductive hypothesis to $[\Sigma \vdash \langle l_b, \rho_0' \rangle \Downarrow v]$*, we need to prove* $(\rho_0' \;\; \mathcal{R} \;\; \hat{\rho})$*. We already have* $(\rho_0 \;\; \mathcal{R} \;\; \hat{\rho})$*. For elements* $x_i \notin dom(\rho_0)$*, we need to prove* $(\rho(x_i) \;\; \mathcal{V}_{\hat{\rho}} \;\; \hat{\rho}(x_i))$*, i.e.,* $(v_i \;\; \mathcal{V}_{\hat{\rho}} \;\; \hat{\rho}(x_i))$*. This is done by applying the definition of* $\preceq^{\mathcal{R}}$ *and* $\mathcal{R}$ *to* $[\Sigma \vdash \langle l_i, \rho \rangle \Downarrow v_i] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l_i, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}_i]$ *where we get* $(v_i \;\; \mathcal{V}_{\hat{\rho}} \;\; \hat{v}_i)$ *which, together with* $\hat{v}_i \subseteq \hat{\rho}(x_i)$ *and* [**f 1.1**]*, gives us* $(v_i \;\; \mathcal{V}_{\hat{\rho}} \;\; \hat{\rho}(x_i))$*.*

Therefore $(\rho_0' \;\; \mathcal{R} \;\; \hat{\rho})$*. Now we can apply the inductive hypothesis to* $[\Sigma \vdash \langle l_b, \rho_0' \rangle \Downarrow v]$ *and derive*

$$[\Sigma \vdash \langle l_b, \rho_0' \rangle \Downarrow v] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l_b, \hat{\rho} \rangle \hat{\Downarrow} \hat{w}_j]$$

We are left to prove $(v \;\; \mathcal{V}_{\hat{\rho}} \;\; \hat{v})$*. By applying the definition of* $\preceq^{\mathcal{R}}$ *and* $\mathcal{R}$ *to* $[\Sigma \vdash \langle l_b, \rho_0' \rangle \Downarrow v] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l_b, \hat{\rho} \rangle \hat{\Downarrow} \hat{w}_j]$ *we get* $(v \;\; \mathcal{V}_{\hat{\rho}} \;\; \hat{w}_j)$*, which together with* $\hat{w}_j \subseteq \hat{v}$ *and* [**f 1.1**] *gives us* $(v \;\; \mathcal{V}_{\hat{\rho}} \;\; \hat{v})$*.*

(**r 1.6**) $\Sigma(l) = (t_0^{l_0} \; t_1^{l_1} \ldots t_n^{l_n})$ *and:*

$$\begin{array}{l}
\Sigma \vdash \langle l_i, \rho \rangle \Downarrow v_i \quad i = 1, \ldots, n \;\; \wedge \\
\Sigma \vdash \langle l_0, \rho \rangle \Downarrow [l_f, \; \rho_0] \;\; \wedge \\
\Sigma(l_f) = (\texttt{fun}\; y \; x_1 \ldots x_n \Rightarrow t_b^{l_b}) \;\; \wedge \\
\Sigma \vdash \langle l_b, \rho_0[y \mapsto \langle l_f, \rho_0 \rangle][x_1 \mapsto v_1] \ldots [x_n \mapsto v_n] \rangle \Downarrow v
\end{array}$$

The proof is similar to the one shown for rule (**r 1.5**) *with the difference that we have to show* $(\langle l_f, \rho_0 \rangle \;\; \mathcal{V}_{\hat{\rho}} \;\; \hat{\rho}(y))$ *as part of the proof of* $(\rho_0[y \mapsto \langle l_f, \rho_0 \rangle][x_1 \mapsto v_1] \ldots [x_n \mapsto v_n] \;\; \mathcal{R} \;\; \hat{\rho})$*.*

Again, $[\Sigma \vdash \langle l, \hat{\rho} \rangle \hat{\Downarrow} \hat{v}]$ *must be the result of applying rule* (**r 2.5**) *to* $\Sigma(l) = (t_0^{l_0} \; t_1^{l_1} \ldots t_n^{l_n})$

where

$$[\Sigma \vdash \langle l_i, \hat{\rho}\rangle \hat{\Downarrow} \hat{v}_i] \quad i = 0, \ldots, n \ \wedge$$

$$\begin{array}{c} \forall\, l_j \in \hat{v}_0 \\ j = 1, \ldots, m \end{array} : \begin{cases} \Sigma(l_j) = (\texttt{fn}\ x_1 \ldots x_n \Rightarrow t_b^{l_b}) : \\ \quad \ldots \\ \Sigma(l_j) = (\texttt{fun}\ y\ x_1 \ldots x_n \Rightarrow t_b^{l_b}) : \\ \quad \begin{cases} \Sigma \vdash \langle l_b, \hat{\rho}\rangle \hat{\Downarrow} \hat{w}_j \\ \hat{v}_i \subseteq \hat{\rho}(x_i) \quad i = 1, \ldots, n \\ l_j \in \hat{\rho}(y) \end{cases} \end{cases}$$

Using $l_f \in \hat{v}_0$ *we derive* $l_f \in \hat{\rho}(y)$. *Clearly* $(\langle l_f, \rho_0\rangle\ \mathcal{V}_{\hat{\rho}}\ \{l_f\})$ *(eq. 2.13) which, together with* $\{l_f\} \subseteq \hat{\rho}(y)$ *and* $[\mathbf{f}\ \mathbf{1.1}]$, *is enough to prove* $\langle l_f, \rho_0\rangle\ \mathcal{V}_{\hat{\rho}}\ \hat{\rho}(y)$.

$(\mathbf{r}\ \mathbf{1.7})$ $\quad \Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})$ *and:*

$$[\Sigma \vdash \langle l_0, \rho\rangle \Downarrow \mathit{true}] \ \wedge\ [\Sigma \vdash \langle l_1, \rho\rangle \Downarrow v]$$

$[\Sigma \vdash \langle l, \hat{\rho}\rangle \hat{\Downarrow} \hat{v}]$ *must be the result of applying rule* $(\mathbf{r}\ \mathbf{2.6})$ *to* $\Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})$ *where*

$$i = 1, 2, 3 : \quad [\Sigma \vdash \langle l_i, \hat{\rho}\rangle \hat{\Downarrow} \hat{v}_i] \quad \wedge \quad \hat{v} = (\hat{v}_1 \cup \hat{v}_2)$$

Using $(\rho\ \mathcal{R}\ \hat{\rho})$ *we get* $\langle l_i, \rho\rangle\ \mathcal{R}\ \langle l_i, \hat{\rho}\rangle$, *for* $i = 0, 1$. *Then we can apply the inductive hypothesis to* $[\Sigma \vdash \langle l_i, \rho\rangle \Downarrow v_i]$, $i = 0, 1$, *and derive*

$$[\Sigma \vdash \langle l_0, \rho\rangle \Downarrow \mathit{true}] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l_0, \hat{\rho}\rangle \hat{\Downarrow} \hat{v}_i]$$
$$[\Sigma \vdash \langle l_1, \rho\rangle \Downarrow v] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l_1, \hat{\rho}\rangle \hat{\Downarrow} \hat{v}_1]$$

By applying the definition of $\preceq^{\mathcal{R}}$ *and* $\mathcal{R}$ *to* $[\Sigma \vdash \langle l_1, \rho\rangle \Downarrow v] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l_1, \hat{\rho}\rangle \hat{\Downarrow} \hat{v}_1]$ *we get* $(v\ \mathcal{V}_{\hat{\rho}}\ \hat{v}_1)$ *which, together with* $\hat{v} = (\hat{v}_1 \cup \hat{v}_2)$ *and* $[\mathbf{f}\ \mathbf{1.1}]$, *is enough to prove* $(v\ \mathcal{V}_{\hat{\rho}}\ \hat{v})$.

$(\mathbf{r}\ \mathbf{1.8})$ $\quad \Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})$ *and:*

$$\Sigma \vdash \langle l_0, \rho\rangle \Downarrow \mathit{false} \ \wedge\ \Sigma \vdash \langle l_2, \rho\rangle \Downarrow v$$

Similar to the $(\mathbf{r}\ \mathbf{1.7})$ *case using* $i = 0, 2$.

(r 1.9) $\Sigma(l) = (\texttt{let}\ x = t_1^{l_1}\ \texttt{in}\ t_2^{l_2})$ *and:*

$$[\Sigma \vdash \langle l_1, \rho\rangle \Downarrow v_1] \ \wedge\ [\Sigma \vdash \langle l_2, \rho[x \mapsto v_1]\rangle \Downarrow v]$$

$[\Sigma \vdash \langle l, \hat{\rho}\rangle \hat{\Downarrow} \hat{v}]$ *must be the result of applying rule* **(r 2.7)** *to* $\Sigma(l) = (\texttt{let}\ x = t_1^{l_1}\ \texttt{in}\ t_2^{l_2})$ *where*

$$\begin{array}{l} [\Sigma \vdash \langle l_1, \hat{\rho}\rangle \hat{\Downarrow} \hat{v}_1] \quad \wedge \quad \hat{v}_1 \subseteq \hat{\rho}(x) \\ [\Sigma \vdash \langle l_2, \hat{\rho}\rangle \hat{\Downarrow} \hat{v}] \end{array}$$

Using $(\rho\ \ \mathcal{R}\ \ \hat{\rho})$ *we get* $\langle l_1, \rho\rangle\ \ \mathcal{R}\ \ \langle l_1, \hat{\rho}\rangle$. *We apply the inductive hypothesis to* $[\Sigma \vdash \langle l_1, \rho\rangle \Downarrow v_1]$ *and derive*

$$[\Sigma \vdash \langle l_1, \rho\rangle \Downarrow v_1] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l_1, \hat{\rho}\rangle \hat{\Downarrow} \hat{v}_1]$$

We need to show $(\rho[x \mapsto v_1]\ \ \mathcal{R}\ \ \hat{\rho})$. *We already have* $(\rho\ \ \mathcal{R}\ \ \hat{\rho})$. *We are left to show* $(\rho(x)\ \ \mathcal{V}_{\hat{\rho}}\ \ \hat{\rho}(x))$, *i.e.,* $(v_1\ \ \mathcal{V}_{\hat{\rho}}\ \ \hat{\rho}(x))$. *By applying the definition of* $\preceq^{\mathcal{R}}$ *and* $\mathcal{R}$ *to* $[\Sigma \vdash \langle l_1, \rho\rangle \Downarrow v_1] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l_1, \hat{\rho}\rangle \hat{\Downarrow} \hat{v}_1]$ *we get* $(v_1\ \ \mathcal{V}_{\hat{\rho}}\ \ \hat{v}_1)$ *which, together with* $(\hat{v}_1 \subseteq \hat{\rho}(x))$ *and* **[f 1.1]** *is enough to show* $(v_1\ \ \mathcal{V}_{\hat{\rho}}\ \ \hat{\rho}(x))$.

Therefore $(\rho[x \mapsto v_1]\ \ \mathcal{R}\ \ \hat{\rho})$ *and, by applying equation (2.15),* $(\langle l_2, \rho[x \mapsto v_1]\rangle)\ \ \mathcal{R}\ \ (\langle l_2, \hat{\rho}\rangle)$. *Now we can apply the inductive hypothesis to* $[\Sigma \vdash \langle l_2, \rho[x \mapsto v_1]\rangle \Downarrow v]$ *and derive*

$$[\Sigma \vdash \langle l_2, \rho[x \mapsto v_1]\rangle \Downarrow v] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l_2, \hat{\rho}\rangle \hat{\Downarrow} \hat{v}]$$

We get $(v\ \ \mathcal{V}_{\hat{\rho}}\ \ \hat{v})$ *by applying the definitions of* $\preceq^{\mathcal{R}}$ *and* $\mathcal{R}$ *to* $[\Sigma \vdash \langle l_2, \rho[x \mapsto v_1]\rangle \Downarrow v] \preceq^{\mathcal{R}} [\Sigma \vdash \langle l_2, \hat{\rho}\rangle \hat{\Downarrow} \hat{v}]$.

[**QED**]

Clearly, the correctness criterion defined in 2.14 is just a corollary of theorem 2.3.1.

This simulation enables us to prove properties of the concrete interpretation using the results generated by the abstract interpretation. That is precisely what program analyses do: they compute approximations of program properties based on abstractions performed on the concrete semantics.

But not all properties can be approximated through this methodology. Properties based on *all* execution paths (safety properties) can be safely approximated by the abstract interpretation because all concrete paths are considered by the abstract trace. On the other hand, properties based on *some* of the paths (liveness properties) of the concrete interpretation cannot be safely approximated by its abstract counterpart [Sch98]. A good example of the latter is the reachability property defined in section 2.1: A configuration that is reachable in the abstract execution tree is not necessarily reachable in the actual execution of the program. In these cases, analyses are defined to work with the negation of the original liveness properties. In the reachability case, an analysis can safely approximate the configurations that are *not* reachable (dead code).

2.3.2 Using Abstract Interpretation

We have used the system defined in Figure 2.3 as an example of the application of the main concepts behind abstract interpretation. Abstract interpretation techniques are used in chapter 3 to define the family of control and data flow analyses used throughout the dissertation. Furthermore, the standard slice and removable slice analyses defined in chapters 4 and 5 are derived by performing abstract interpretation on the respective standard slice and removable slice specifications.

This section also sets up the style of the theorems that state the correctness of the analyses defined in the following chapters - theorem 2.3.1 is the blueprint for such theorems. Elements of the concrete and abstract interpretations (analyses) are related using safety relations similar to $\mathcal{V}$ and $\mathcal{R}$. Correctness results are presented by stating that the analysis safely simulates (as defined by $\mathcal{R}$) the concrete interpretation, that is, their derivation trees form an $\mathcal{R}$-simulation ($\preceq^{\mathcal{R}}$). These results are proved by rule induction on the derivation tree of the concrete interpretation.

Chapter 3

Control and Data Flow Information

Control flow information is essential to perform optimizations in compilers of higher-order languages [Shi91]. In this chapter we show a framework to derive provably correct control and data flow analyses by abstracting the context collecting semantics of the language.

In section 3.1 we introduce the concept of binding contexts and define the context collecting semantics of the language. This semantics is only used to derive and prove correct the context CFA presented in section 3.2. In order to do this, we use the abstract interpretation techniques presented in section 2.3 of the previous chapter.

The definitions presented in section 2.3 and figure 3.2 are crucial. The analyses defined in sections 4 and 5 are based on the context CFA. We finish the chapter by presenting an instantiation of the context CFA, the 0CFA.

3.1 Collecting Semantics as a validation system

A collecting interpretation of a program [HY91] collects all the values evaluated during the execution of the program. The interpretation can collect different types of information and can have different levels of detail. In this section we deal with interpretations that collect only values.

The collecting semantics of the natural semantics of a language introduces a new structure called a *cache*, which is used to record the values produced by each configuration during the evaluation of a program. A cache Φ is a function that maps program configurations to values:

$$\Phi \in \mathit{Cache} : \mathit{Config} \ \rightarrow \ \mathit{Val},$$

We say that a cache Φ has correct information of the execution of $[\Sigma \vdash \mathcal{C} \Downarrow v]$ if,

$$\forall \mathcal{C}', v'.\ [\Sigma \vdash \mathcal{C}' \Downarrow v'] \in \mathit{reach}^*([\Sigma \vdash \mathcal{C} \Downarrow v]) \quad \Rightarrow \quad \Phi(\mathcal{C}') = v', \tag{3.1}$$

that is, the cache contains correct information of every configuration in $[\Sigma \vdash \mathcal{C} \Downarrow v]$, the derivation tree of $\Sigma \vdash \mathcal{C} \Downarrow v$.

The collecting semantics of the language will be defined as set of rules that compute or validate a cache $\widehat{\Phi}$ that satisfies the conditions in 3.1. We show two approaches.

The first approach is to modify the value set and derivation rules of the original semantics so that the cache is *computed* by each step of the execution of the program. In this approach, the semantics is given by judgements $\Sigma \vdash \langle l, \rho \rangle \Downarrow (v, \Phi)$, where v is evaluated in the same way as before and Φ is the result of combining the caches obtained from the subtrees of the derivation tree, if any. For example, the rule for `if`-expressions will look like:

$$\frac{\Sigma \vdash \langle l_0, \rho \rangle \Downarrow (\mathit{true}, \Phi_0) \quad \Sigma \vdash \langle l_1, \rho \rangle \Downarrow (v, \Phi_1)}{\Sigma \vdash \langle l, \rho \rangle \Downarrow (v, \Phi_0 \cup \Phi_1 \cup [\langle l, \rho \rangle \mapsto v])}$$

where $\Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})$. The rest of the rules follow the same structure.

The second approach defines a collecting semantics that *validates* the contents of the cache. Initially, the collecting semantics is presented by judgements of the form $(\Sigma, \Phi) \vdash \mathcal{C} \Downarrow v$ defined by the same rules that define $\Sigma \vdash \mathcal{C} \Downarrow v$ with the addition of the constraint $\Phi(\mathcal{C}) = v$. In other words, the collecting semantics can be re-written as $(\Sigma, \Phi) \vdash \mathcal{C} \Downarrow \Phi(\mathcal{C})$.

We prefer to write $\mathcal{C} \models (\Sigma, \Phi)$, a judgement that claims that Φ represents a consistent description of the evaluation of $\mathcal{C}$. We adopt this notation and re-write the `if`-rule as:

$$\frac{\langle l_0, \rho\rangle \models (\Sigma, \Phi) \quad \langle l_1, \rho\rangle \models (\Sigma, \Phi) \quad \Phi(\langle l_0, \rho\rangle) = \mathit{true} \quad \Phi(\langle l_1, \rho\rangle) = \Phi(\langle l, \rho\rangle)}{\langle l, \rho\rangle \models (\Sigma, \Phi)}$$

We choose the second approach because it relates more clearly global information - used by program optimizations - to local behavior - the evaluation of configurations. The collective semantics is then presented by the judgement $\mathcal{C} \models (\Sigma, \Phi)$ that validates the contents of the global cache Φ against the evaluation of $\mathcal{C}$. This will be the style used to define our analyses.

$\mathcal{C} \models (\Sigma, \Phi)$ correctly defines the collecting semantics of the evaluation of $\mathcal{C}$ if the derivation trees of the standard and the collecting semantics form a $\mathcal{R}^{co}$-bisimulation, *i.e.*, $[\Sigma \vdash \mathcal{C} \Downarrow v] \;\cong^{co}\; [\mathcal{C}' \models (\Sigma', \Phi)]$, where $\mathcal{R}^{co}$ is defined by:

$$(\Sigma \vdash \mathcal{C} \Downarrow v) \;\; \mathcal{R}^{co} \;\; (\mathcal{C}' \models (\Sigma', \Phi)) \qquad \Leftrightarrow \qquad \Sigma = \Sigma' \;\wedge\; \mathcal{C} = \mathcal{C}' \;\wedge\; \Phi(\mathcal{C}) = v \quad , \tag{3.2}$$

that is, every node of $[\Sigma \vdash \mathcal{C} \Downarrow v]$ is matched by a node of the collecting semantics $[\mathcal{C}' \models (\Sigma', \Phi)]$ and hence $\Phi(\mathcal{C}) = v$. It follows from the definition of $\cong^{co}$ in equation (2.10) that if $[\Sigma \vdash \mathcal{C} \Downarrow v] \;\cong^{co}\; [\mathcal{C}' \models (\Sigma', \Phi)]$ then (3.1) holds for Φ and $[\Sigma \vdash \mathcal{C} \Downarrow v]$.

3.1.1 Collecting Semantics and Context Information

The collecting semantics can be enhanced to keep track of different kinds of information. This information can later be used as input to program optimizations. The collecting semantics can also be instrumented with elements that make it more amenable to further manipulation.

In particular, we are concerned with providing a family of correct abstract interpretations of the language that can later be used to derive program analyses from property specifications. Higher-order languages pose a special challenge and the treatment of closures is crucial. A great amount of research has been dedicated to tackle this issue and as a result a great amount of work has been reported on the development of frameworks for the definition of closure analysis [Pal95, Ses91, JW94, JW95] and, as it is also known, control flow analysis [Shi88, Shi91].

The core of a control flow analysis is the design of a strategy to differentiate the points where closures are created and applied. In this presentation we deal with the notion of *binding contexts*. A (binding) context identifies the point where variable binding is performed.

We decorate our semantics with *contexts* - $m \in Context$. A new, fresh context is assigned to every configuration that performs variable binding or that extends the environment, *i.e.*, application and `let` rules. All other configurations keep the context of their immediate root. Thus, we write $\langle l, \rho \rangle \models^m (\Sigma, \Phi)$ and provide the new semantics with a context instantiator function, the total, injective function $\mathcal{I}$:

$$\mathcal{I} : Univ \times Lab \times Context \times Lab \rightarrow Context$$

where we write $m' = \mathcal{I}(\Sigma, l, m, l_f)$ to instantiate m' during the execution of application l at current context m to be used for the execution of the applied function l_f.

The use of contexts is similar to the use of contours introduced by Shivers [Shi91] and extended by Jaganathan and Weeks [JW94]. Our definition could be seen as a special case of the framework developed by Nielson and Nielson [NN97], where *mementa* are used instead. These frameworks are more complex and their goal is to cover a wider variety of abstractions and well-known flow analyses.

For example, a context can be defined as a sequence of labels $l_1 : \ldots : l_n$ that records the history of binding operations performed before reaching a particular configuration of the evaluation of the program. New contexts are created by appending the label of the binding expression (an application or a `let`-expression) to the current context[1]. Therefore, given context $l_1 : \ldots : l_n$, l_n marks the last binding operation and l_1 denotes the initial context. For this particular case, the domain of contexts and the instantiators would be defined by:

$$\begin{aligned} m \in \mathit{Context} &= \mathit{Lab}^* \\ \mathcal{I}(\Sigma, l, m, l') &= m :: l \end{aligned} \tag{3.3}$$

where Lab^* denotes the domain of label sequences and :: is the operation that adds a label to the end of a sequence.

We leave the definition of the instantiators and *Context* unspecified except for one restriction: that in all derivation trees there exists a one to one mapping between the configurations $\langle l, \rho \rangle$ of the tree and the pairs (l, m). The combination of label and context should uniquely identify a configuration. We take advantage of this and redefine caches as mappings from pairs (l, m) to values. Thus, if $(\Sigma, \Phi) \vdash^m \langle l, \rho \rangle$ is the collecting semantics of $\Sigma \vdash \langle l, \rho \rangle \Downarrow v$ then $\Phi(l, m)$ should be v.

The presence of binding contexts makes it simple to keep track of all variable bindings during the execution of a program. In order to do this we need to modify the collecting semantics specification in two ways. First we make the cache a sort of global environment by extending its domain to pairs of variables and contexts. The value of variable x in context m can be found in $\Phi(x, m)$. The second change is to replace the original environments with context environments. A context environment is a mapping from variables to the contexts where they were initially bound. Thus, a configuration $\langle l, \rho \rangle$ is written as $\langle l, me \rangle$ where $\rho(x) = \Phi(x, me(x))$. In the same spirit, closures are now represented by pairs of the form $[l,\ me]$.

[1] The call chain can be extracted by considering only the application labels.

$$\textbf{(r 3.1)}\quad \langle l, me\rangle \models^m (\Sigma,\Phi) \quad \text{if } \Sigma(l)=c \ \wedge\ \Phi(l,m)=c$$

$$\textbf{(r 3.2)}\quad \langle l, me\rangle \models^m (\Sigma,\Phi) \quad \text{if } \Sigma(l)=x \ \wedge\ x\in dom(me) \ \wedge\ \Phi(l,m)=\Phi(x,me(x))$$

$$\textbf{(r 3.3)}\quad \langle l, me\rangle \models^m (\Sigma,\Phi) \quad \text{if } \Sigma(l)\in \mathit{Fun} \ \wedge\ \Phi(l,m)=[l,\ me]$$

$$\textbf{(r 3.4)}\quad \frac{\begin{array}{l}\Sigma(l)=(p\ t_1^{l_1}\ldots t_n^{l_n})\\ \langle l_i, me\rangle \models^m (\Sigma,\Phi)\ \wedge\ \Phi(l_i,m)=c_i \quad i\in\{1,\ldots,n\}\\ \Phi(l,m)=\bar{p}(c_1\ldots c_n)\end{array}}{\langle l, me\rangle \models^m (\Sigma,\Phi)}$$

$$\textbf{(r 3.5)}\quad \frac{\begin{array}{l}\Sigma(l)=(t_0^{l_0}\ t_1^{l_1}\ldots t_n^{l_n})\ \wedge\\ \langle l_i, me\rangle \models^m (\Sigma,\Phi) \quad i=0,\ldots,n\ \wedge\\ \Phi(l_0,m)=[l_f,\ me_0]\ \wedge\ \Sigma(l_f)=(\texttt{fn}\,x_1\ldots x_n \Rightarrow t_b^{l_b})\ \wedge\\ \langle l_b, me_0[x_1\mapsto m']\ldots[x_n\mapsto m']\rangle \models^{m'} (\Sigma,\Phi)\\ \Phi(x_i,m')=\Phi(l_i,m) \quad i=1,\ldots,n\\ \Phi(l,m)=\Phi(l_b,m')\end{array}}{\langle l, me\rangle \models^m (\Sigma,\Phi)}\quad m'=\mathcal{I}(\Sigma,l,m,l_f)$$

$$\textbf{(r 3.6)}\quad \frac{\begin{array}{l}\Sigma(l)=(t_0^{l_0}\ t_1^{l_1}\ldots t_n^{l_n})\ \wedge\\ \langle l_i, me\rangle \models^m (\Sigma,\Phi) \quad i=0,\ldots,n\ \wedge\\ \Phi(l_0,m)=[l_f,\ me_0]\ \wedge\ \Sigma(l_f)=(\texttt{fun}\,y\ x_1\ldots x_n \Rightarrow t_b^{l_b})\ \wedge\\ \langle l_b, me_0[y\mapsto m'][x_1\mapsto m']\ldots[x_n\mapsto m']\rangle \models^{m'} (\Sigma,\Phi)\\ \Phi(x_i,m')=\Phi(l_i,m) \quad i=1,\ldots,n\\ \Phi(y,m')=[l_f,\ me_0]\ \wedge\ \Phi(l,m)=\Phi(l_b,m')\end{array}}{\langle l, me\rangle \models^m (\Sigma,\Phi)}\quad m'=\mathcal{I}(\Sigma,l,m,l_f)$$

$$\textbf{(r 3.7)}\quad \frac{\begin{array}{l}\Sigma(l)=(\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})\ \wedge\\ \langle l_0, me\rangle \models^m (\Sigma,\Phi)\ \wedge\ i=\begin{cases}1 & \text{if } \Phi(l_0,m)=\mathit{true}\\ 2 & \text{if } \Phi(l_0,m)=\mathit{false}\end{cases}\\ \langle l_i, me\rangle \models^m (\Sigma,\Phi)\ \wedge\ \Phi(l,m)=\Phi(l_i,m)\end{array}}{\langle l, me\rangle \models^m (\Sigma,\Phi)}$$

$$\textbf{(r 3.8)}\quad \frac{\begin{array}{l}\Sigma(l)=(\texttt{let}\ x=t_1^{l_1}\ \texttt{in}\ t_2^{l_2})\ \wedge\\ \langle l_1, me\rangle \models^m (\Sigma,\Phi)\ \wedge\ \langle l_2, me[x\mapsto m']\rangle \models^{m'} (\Sigma,\Phi)\\ \Phi(x,m')=\Phi(l_1,m)\ \wedge\ \Phi(l,m)=\Phi(l_2,m')\end{array}}{\langle l, me\rangle \models^m (\Sigma,\Phi)}\quad m'=\mathcal{I}(\Sigma,l,m,l)$$

Figure 3.1: Exact Context Collecting Semantics

The new semantics contains the following domains:

$$\begin{array}{rcl} m \in \mathit{Context} & & \\ \Phi \in \mathit{Cache} & = & (\mathit{Lab} \cup \mathit{Var}) \times \mathit{Context} \to \mathit{Val} \\ me \in \mathit{CtxtEnv} & = & \mathit{Var} \to \mathit{Context} \end{array}$$

A cache Φ determines a set of well-defined contexts environments. Given Φ, $\Phi \models me$ is inductively defined by:

$$\begin{array}{lcl} \Phi \models me & \Leftrightarrow & \forall x \in dom(me),\ (x, me(x)) \in dom(\Phi) \ \ \wedge \ \ \Phi \models \Phi(x, me(x)) \\ \Phi \models c & \Leftrightarrow & \text{always} \\ \Phi \models [l,\ me] & \Leftrightarrow & \Phi \models me \end{array}$$

A context environment is well-defined, with respect to Φ, if it maps variables to contexts in Φ and if it has a finite number of unfoldings or finite depth. The depth of a context environment is defined by:

$$\begin{array}{rcl} depth(\Phi, me) & = & max\{d \mid x \in dom(me) \ \wedge \ d = depth(\Phi, \rho(x, \Phi(x)))\} \\ depth(\Phi, c) & = & 0 \\ depth(\Phi, [l,\ me]) & = & 1 + depth(\Phi, me) \end{array}$$

From now on, we will only consider well-defined context environments.

The new satisfiability relation is defined by the judgement $\langle l, me\rangle \models^m (\Sigma, \Phi)$ which claims that Φ has correct information about configuration $\langle l, me\rangle$ under context m.

Definition 3.1.1 (Exact Context Collecting Semantics)
$\langle l, me\rangle \models^m (\Sigma, \Phi)$ is defined as the least relation that satisfies the rules in Figure 3.1.

Correctness of the Exact Context Collecting Semantics

We state the correctness of the collecting semantics as a bisimulation result between the derivation trees of $\Sigma \vdash \langle l, \rho\rangle \Downarrow v$ and $\langle l, me\rangle \models^m (\Sigma, \Phi)$, whenever certain conditions for ρ and me are true. This is a similar result to the one demanded in (3.2).

Following equation (2.10) we start by defining the relations between the nodes of the standard and collecting semantics. The relations between environments and values are defined by:

$$
\begin{array}{lcl}
\rho\ \mathcal{R}^{co}_{\Phi}\ me & \Leftrightarrow & \begin{cases} dom(\rho) = dom(me) \\ \forall x \in dom(\rho),\ \rho(x)\ \mathcal{V}^{co}_{\Phi}\ \Phi(x, me(x)) \end{cases} \\
c\ \mathcal{V}^{co}_{\Phi}\ c' & \Leftrightarrow & c = c' \\
[l,\, \rho]\ \mathcal{V}^{co}_{\Phi}\ [l',\, me] & \Leftrightarrow & (l = l')\ \wedge\ (\rho\ \mathcal{R}^{co}_{\Phi}\ me)
\end{array}
\tag{3.4}
$$

These relations are parameterized by a cache Φ. A value environment ρ and a context environment me are $\mathcal{V}^{co}_{\Phi}$-related if they share the same domains and if they map to values that are $\mathcal{R}^{co}_{\Phi}$-related. Configurations and nodes are related according to:

$$
\begin{array}{lcl}
\langle l, \rho\rangle\ \ \mathcal{R}^{co}_{\Phi}\ \ \langle l', me\rangle & \Leftrightarrow & (l = l')\ \wedge\ \rho\ \mathcal{R}^{co}_{\Phi}\ me \\
(\Sigma \vdash \langle l, \rho\rangle \Downarrow v)\ \ \mathcal{R}^{co}\ \ (\langle l', me\rangle \models^{m} (\Sigma', \Phi)) & \Leftrightarrow & \begin{cases} (\Sigma = \Sigma') \\ \langle l, \rho\rangle\ \ \mathcal{R}^{co}_{\Phi}\ \ \langle l', me\rangle \\ v\ \ \mathcal{V}^{co}_{\Phi}\ \ \Phi(l', m) \end{cases}
\end{array}
\tag{3.5}
$$

A standard and a context configuration are related if they share the same label and their environments are related. Similarly, two nodes are related if their configurations are related and if the entry in the cache associated to the context configuration is the result of evaluating the related standard configuration.

Now we are ready to state the correctness of the collecting semantics. We claim that the derivation trees of two related configurations form a bisimulation:

Theorem 3.1.2 (Correctness of the Exact Context Collecting Semantics)
$[\langle l, me\rangle \models^{m} (\Sigma, \Phi)]$ *and* $(\rho\ \mathcal{R}^{co}_{\Phi}\ me) \Rightarrow \exists v.\,[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]\ \cong^{co}\ [\langle l, me\rangle \models^{m} (\Sigma, \Phi)]$.

Proof: *By rule induction on* $[\langle l, me\rangle \models^{m} (\Sigma, \Phi)]$. *We have the following cases:*

The first three cases correpond to the axioms of the system. Since axioms generate single-node trees (no subtrees), according to the definition of bisimulation we only need *to show* $(\Sigma \vdash \langle l, \rho\rangle \Downarrow v \;\; \mathcal{R}^{co} \;\; \langle l, me\rangle \models^m (\Sigma, \Phi))$. *Since we already have* $(\rho \;\; \mathcal{R}^{co}_{\Phi} \;\; me)$, *we are left to show* $(v \;\; \mathcal{V}^{co}_{\Phi} \;\; \Phi(l, m))$.

The first three axioms are:

(r 3.1) $\Sigma(l) = c$ *and* $\Phi(l, m) = c$

By applying rule **(r 1.1)** *to* $\Sigma(l) = c$ *we get* $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$ *where* $v = c$. *We use equation (3.4) and* $\Phi(l, m) = c$ *to derive* $(c \;\; \mathcal{V}^{co}_{\Phi} \;\; \Phi(l, m))$, *i.e.,* $(v \;\; \mathcal{V}^{co}_{\Phi} \;\; \Phi(l, m))$.

(r 3.2) $\Sigma(l) = x$, $x \in dom(me)$ *and* $\Phi(l, m) = \Phi(x, me(x))$

We use $(\rho \;\; \mathcal{R}^{co}_{\Phi} \;\; me)$, $x \in dom(me)$ *and equation (3.4) to derive* $x \in dom(\rho)$ *and* $\rho(x) \;\; \mathcal{V}^{co}_{\Phi} \;\; \Phi(x, me(x))$. *We apply* **(r 1.2)** *to* $\Sigma(l) = x$ *and* $x \in dom(\rho)$ *to derive* $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$ *where* $v = \rho(x)$.
Using all the above we get $(v \;\; \mathcal{V}^{co}_{\Phi} \;\; \Phi(l, m))$.

(r 3.3) $\Sigma(l) \in Fun$ *and* $\Phi(l, m) = [l,\ me]$

By applying **(r 1.3)** *to* $\Sigma(l) \in Fun$ *we derive* $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$ *where* $v = [l,\ \rho]$. *Using* $(\rho \;\; \mathcal{R}^{co}_{\Phi} \;\; me)$ *and the definition in equation (3.4) we get* $([l,\ \rho] \;\; \mathcal{V}^{co}_{\Phi} \;\; [l,\ me])$, *i.e.,* $(v \;\; \mathcal{V}^{co}_{\Phi} \;\; \Phi(l, m))$

The rest of the rules (non-axioms) generate trees of height greater than one. This means that we also have to relate the subtrees of the derivation trees. We must show that there exists v *such that* $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$ *is true and, according to equation (2.9): (1) for each subtree* $\mathcal{N}$ *of* $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$ *we must find a subtree* $\mathcal{M}$ *of* $[\langle l, me\rangle \models^m (\Sigma, \Phi)]$ *such that* $\mathcal{N} \;\cong^{co}\; \mathcal{M}$ *and, (2) The roots* $\Sigma \vdash \langle l, \rho\rangle \Downarrow v$ *and* $\langle l, me\rangle \models^m (\Sigma, \Phi)$ *are* $\mathcal{R}^{co}$*-related.*

The first condition is satisfied by applying the inductive hypothesis to each subtree of $[\langle l, me\rangle \models^m (\Sigma, \Phi)]$. *For each subtree* $[\langle l', me'\rangle \models^{m'} (\Sigma, \Phi)]$ *of* $[\langle l, me\rangle \models^m (\Sigma, \Phi)]$ *we can find* v' *such that* $[\Sigma \vdash \langle l', \rho'\rangle \Downarrow v'] \;\cong^{co}\; [\langle l', me'\rangle \models^{m'} (\Sigma, \Phi)]$. *We get the desired* v *by* *applying one of the rules in figure 2.2 to exactly all the derived trees* $[\Sigma \vdash \langle l', \rho' \rangle \Downarrow v']$.

For the second condition we just need to show that $(v \ \mathcal{V}^{co}_{\Phi} \ \Phi(l, m))$ *is true.*

We only show the primitive and `fn` *application cases:*

(r 3.4) $\Sigma(l) = (p\ t_1^{l_1} \dots t_n^{l_n})$ *and*

$$i \in \{1, \dots, n\} : [\langle l_i, me \rangle \models^m (\Sigma, \Phi)] \ \wedge \ \Phi(l_i, m) = c_i$$
$$\Phi(l, m) = \bar{p}(c_1 \dots c_n)$$

We apply the inductive hypothesis to $[\langle l_i, me \rangle \models^m (\Sigma, \Phi)]$ *and* $(\rho \ \mathcal{R}^{co}_{\Phi} \ me)$ *and derive*

$$[\Sigma \vdash \langle l_i, \rho \rangle \Downarrow v_i] \ \cong^{co} \ [\langle l_i, me \rangle \models^m (\Sigma, \Phi)] \quad i \in \{1, \dots, n\}$$

By applying the definitions of $\cong^{co}$ *and* $\mathcal{R}^{co}_{\Phi}$ *(eq. 3.5), to the results above we get*

$$v_i = \Phi(l_i, m) = c_i \quad i \in \{1, \dots, n\}$$

We apply rule (**r 1.4**) *to* $\Sigma(l) = (p\ t_1^{l_1} \dots t_n^{l_n})$ *and* $[\Sigma \vdash \langle l_i, \rho \rangle \Downarrow v_i], i \in \{1, \dots, n\}$ *to derive*

$$[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \ \text{ where } \ v = \bar{p}(c_1 \dots c_n)$$

that is, $v = \Phi(l, m)$ *and* $(v \ \mathcal{V}^{co}_{\Phi} \ \Phi(l, m))$ *(eq. 3.4). Therefore*

$$(\Sigma \vdash \langle l, \rho \rangle \Downarrow v) \ \mathcal{R}^{co}_{\Phi} \ (\langle l, me \rangle \models^m (\Sigma, \Phi)) \quad \text{equation (3.5)}$$
$$[\Sigma \vdash \langle l_i, \rho \rangle \Downarrow v_i] \ \cong^{co} \ [\langle l_i, me \rangle \models^m (\Sigma, \Phi)] \quad i \in \{1, \dots, n\}$$

enough conditions to prove $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \ \cong^{co} \ [\langle l_i, me \rangle \models^m (\Sigma, \Phi)]$ *(eq. 2.10).*

(r 3.5) $\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \dots t_n^{l_n})$ *and* $\Sigma(l_f) = (\texttt{fn}\, x_1 \dots x_n \Rightarrow t_b^{l_b})$

$$[\langle l_i, me \rangle \models^m (\Sigma, \Phi)] \quad i = 0, \dots, n \ \wedge \ \Phi(l_0, m) = [l_f,\ me_0]$$
$$[\langle l_b, me_0' \rangle \models^{m'} (\Sigma, \Phi)] \ \wedge \ \Phi(x_i, m') = \Phi(l_i, m) \quad i = 1, \dots, n \quad [\mathbf{f\ 2.1}]$$
$$\Phi(l, m) = \Phi(l_b, m') \quad [\mathbf{f\ 2.2}]$$

where $me_0' = me_0[x_1 \mapsto m'] \dots [x_n \mapsto m']$ *and* $m' = \mathcal{I}(\Sigma, l, m, l_f)$.

We first apply the inductive hypothesis to $[\langle l_i, me\rangle \models^m (\Sigma, \Phi)]$ *and derive*

$$i \in \{0, \ldots, n\} : \quad \exists v_i.\ [\Sigma \vdash \langle l_i, \rho\rangle \Downarrow v_i] \ \cong^{co}\ [\langle l_i, me\rangle \models^m (\Sigma, \Phi)]$$

By applying the definitions of $\cong^{co}$ *and* $\mathcal{R}^{co}_{\Phi}$ *to the previous results we get:*

$$i \in \{0, \ldots, n\} : \quad (v_i\ \ \mathcal{V}^{co}_{\Phi}\ \ \Phi(l_i, m)) \quad [\mathbf{f\ 2.3}]$$

We replace $\Phi(l_0, m) = [l_f,\ me_0]$ *above and get* $(v_0\ \ \mathcal{V}^{co}_{\Phi}\ \ [l_f,\ me_0])$*. Then, by applying the definition of* $\mathcal{V}^{co}_{\Phi}$ *we get:*

$$\begin{aligned}&[\Sigma \vdash \langle l_0, \rho\rangle \Downarrow [l_f,\ \rho_0]] \ \cong^{co}\ [\langle l_0, me\rangle \models^m (\Sigma, \Phi)] \quad [\mathbf{f\ 2.4}]\\ &\textit{where } (\rho_0\ \ \mathcal{R}^{co}_{\Phi}\ \ me_0)\end{aligned}$$

Let $\rho'_0 = \rho_0[x_i \mapsto v_i]^n_1$*. We want to apply the inductive hypothesis to* $[\langle l_b, me'_0\rangle \models^{m'} (\Sigma, \Phi)]$ *with* $\rho'_0\ \ \mathcal{R}^{co}_{\Phi}\ \ me'_0$*. We need to show:*

$$\forall x \in dom(\rho'_0).\ \rho'_0(x)\ \ \mathcal{V}^{co}_{\Phi}\ \ \Phi(x, me'_0(x))$$

If $x \in dom(\rho_0)$ *then the result follows from* $(\rho_0\ \ \mathcal{R}^{co}_{\Phi}\ \ me_0)$*. If* $x = x_i$*, for some* $i \in \{1, \ldots, n\}$*, then* $\rho'_0(x) = v_i$ *and* $\Phi(x, me'_0(x)) = \Phi(x_i, m')$*. By replacing* $[\mathbf{f\ 2.1}]$ *in* $[\mathbf{f\ 2.3}]$ *we get* $(v_i\ \ \mathcal{V}^{co}_{\Phi}\ \ \Phi(x_i, m'))$*, i.e.,* $(\rho'_0(x)\ \ \mathcal{V}^{co}_{\Phi}\ \ \Phi(x, me'_0(x)))$*.*

Therefore $(\rho'_0\ \ \mathcal{R}^{co}_{\Phi}\ \ me'_0)$*. Now we can apply the inductive hypothesis to* $[\langle l_b, me'_0\rangle \models^{m'} (\Sigma, \Phi)]$ *and derive:*

$$\exists v.\ [\Sigma \vdash \langle l_b, \rho'_0\rangle \Downarrow v] \ \cong^{co}\ [\langle l_b, me'_0\rangle \models^{m'} (\Sigma, \Phi)] \quad [\mathbf{f\ 2.5}]$$

We apply rule $(\mathbf{r\ 1.5})$ *to* $[\mathbf{f\ 2.3}]$*,* $[\mathbf{f\ 2.4}]$ *and* $[\mathbf{f\ 2.5}]$*, using* $\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n})$ *and* $\Sigma(l_f) = (\mathtt{fn}\ x_1 \ldots x_n \Rightarrow t_b^{l_b})$*, and derive:*

$$[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$$

By inspecting $[\mathbf{f\ 2.3}]$*,* $[\mathbf{f\ 2.4}]$ *and* $[\mathbf{f\ 2.5}]$*, we can see that there is a one-to-one match between the subtrees of* $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$ *and* $[\langle l, me\rangle \models^m (\Sigma, \Phi)]$*. We just need to show*

*$(v\ \mathcal{V}^{co}_{\Phi}\ \Phi(l,m))$ in order to show $(\Sigma \vdash \langle l,\rho\rangle \Downarrow v\ \mathcal{R}^{co}_{\Phi}\ \langle l, me\rangle \models^{m} (\Sigma,\Phi))$. By applying the definitions of $\cong^{co}$ and $\mathcal{R}^{co}_{\Phi}$ to [**f 2.5**] we get $(v\ \mathcal{V}^{co}_{\Phi}\ \Phi(l_b,m'))$. We get the desired result by replacing [**f 2.2**] in $(v\ \mathcal{V}^{co}_{\Phi}\ \Phi(l_b,m'))$.*

[**QED**]

3.2 Control and Data Flow Analysis

The immediate task of a control and data flow analysis is to compute an approximation of data and flow information generated during the execution of a program. In particular, a *control flow analysis* is concerned with the collection of flow information of a program, that is, the set of closures that flow to the call sites of a program.

We define the control and data flow analysis as an abstraction of the context collecting semantics defined in section 3.1. The use of contexts and context environments provides great flexibility to apply different kind of abstractions. Given the domains *Context* and *CtxtEnv*, and the instantiator function $\mathcal{I}$, we define the following:

$$
\begin{aligned}
\widehat{m} &\in \widehat{Context} \\
\widehat{me} &\in \widehat{CtxtEnv} &&= \quad Var \to \widehat{Context} \\
\widehat{\mathcal{I}} &\in && \quad Univ \times Lab \times \widehat{Context} \to \widehat{Context}
\end{aligned}
$$

We leave these abstractions unspecified though certain conditions must apply to ensure correctness - we make these conditions explicit later on. A common example of abstract contexts is the one used to define k-CFA analyses. This approach uses the sequence of binding points referred in section 3.1.1 but only considers the last - if any - k contexts. The case where only sequences of length 0 are considered, *i.e.*, there is a single possible context, is known as 0-CFA.

Abstract configurations and caches maintain the same structure but carry abstract contexts and abstract context environments instead. We have:

$$\begin{array}{rcl}\hat{C} \in \widehat{Config} & ::= & \langle l, \widehat{me} \rangle \\ \hat{\Phi} \in \widehat{Cache} & = & (Lab \cup Var) \times \widehat{Context} \to \widehat{Val}\end{array}$$

Our next concern is to define the domain of abstract values $\widehat{Val}$. Most control flow analyses abstract away - ignore - constants and deal only with closures. Our intention is to provide a definition that gives room to independently choose abstractions for constants and closures. In such a framework, pure control flow analysis would be a special case.

One way of implementing this is to define a separate structure - similar to our cache - to keep track solely of constants, as in [NNH99, section 3.5]. Another alternative is to keep both parts together, as in [SF94], where abstract values are pairs of constants and closure elements. We follow an approach similar to the latter one.

We hypothesize a domain of abstract values $\widehat{Val}$ with elements that contain a data part, denoted by the domain of abstract constants $\widehat{Const}$, and a flow part, denoted by the domain of abstract closures $\widehat{Closure}$. We have:

$$\begin{array}{rcl}\hat{v} & \in & \widehat{Val} \\ \Pi_{\mathsf{c}} & = & \widehat{Val} \to \widehat{Const} \\ \Pi_{\lambda} & = & \widehat{Val} \to \widehat{Closure}\end{array}$$

The constant and closure components of an element in $\widehat{Val}$ can be obtained with the projection functions Π_{c} and Π_{λ}, respectively. We impose an ordering relation $\sqsubseteq$ on $\widehat{Const}$ and $\widehat{Closure}$. The ordering relation for $\widehat{Val}$ is defined component-wise:

$$\hat{v}_1 \sqsubseteq \hat{v}_2 \;\Leftrightarrow\; (\Pi_{\mathsf{c}}(\hat{v}_1) \sqsubseteq \Pi_{\mathsf{c}}(\hat{v}_2)) \;\wedge\; (\Pi_{\lambda}(\hat{v}_1) \sqsubseteq \Pi_{\lambda}(\hat{v}_2))$$

Each analysis must provide a definition of $\widehat{Const}$, which must include:

$$\begin{array}{rcll}\perp^{\mathsf{c}} & \in & \widehat{Const} & \text{(bottom element)} \\ \beta_{\mathsf{c}} & \in & Const \to \widehat{Const} & \end{array}$$

We define the domain of abstract closures $\widehat{Closure}$. We might be inclined to define abstract closures component-wise and denote an abstract closure by $[l,\ \widehat{me}]$ but, as it was shown in section 2.3, information of several closures may be merged into a single value. As before, this problem is solved by using powersets. We have:

$$\begin{array}{rcl} \widehat{Closure} & = & \mathcal{P}(Lab \times \widehat{CtxtEnv}) \\ \bot^{\lambda} & = & \{\,\} \end{array}$$

The expressiveness of the analysis is by no means restricted by fixing $\widehat{Closure}$[2]. The type of analysis is determined by the choice of the *Context* and $\widehat{Context}$ domains. Nevertheless, it would be interesting to explore a more abstract representation of closures.

The analysis is specified by the judgement $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})$ defined co-inductively by the rules in Figure 3.2. The semantics of these propositions is a consequence of the semantics of $\langle l, me\rangle \models^{m} (\Sigma, \Phi)$ (Figure 3.1).

3.2.1 Correctness of the Analysis with respect to the Collecting Semantics

We prove that the control and data flow analysis safely approximates the exact collecting semantics of the language. The safety relation $\mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}}$ between elements of the exact and abstract semantics is defined as the smallest relation that satisfies:

$$\begin{array}{rcl} me\ \mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{me} & \Leftrightarrow & \forall x \in dom(me),\ \Phi(x, me(x))\ \ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \ \widehat{\Phi}(x, \widehat{me}(x)) \\ c\ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \hat{v} & \Leftrightarrow & \beta_{\mathsf{c}}(c) \sqsubseteq \Pi_{\mathsf{c}}(\hat{v}) \\ [l,\ me]\ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \hat{v} & \Leftrightarrow & \exists\, [l,\ \widehat{me}] \in \Pi_{\lambda}(\hat{v}),\ me\ \mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{me} \end{array} \qquad (3.6)$$

It is important to point out that only well-defined - equation (3.1.1) - context environments satisfy the $\mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}}$ relation. The $\mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}$ relation enjoys the following property:

Lemma 3.2.1 *If* $(v\ \ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \ \hat{v})$ *and* $\hat{v} \sqsubseteq \hat{v}'$ *then* $(v\ \ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \ \hat{v}')$

[2]Though the range of analyses is restricted by our choice of abstract semantics.

$$\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}) \Leftrightarrow$$

$$\textbf{(r 4.1)}\quad \Sigma(l) = c \;\Rightarrow\; \beta_{\mathsf{c}}(c) \sqsubseteq \Pi_{\mathsf{c}}(\widehat{\Phi}(l, \widehat{m}))$$

$$\textbf{(r 4.2)}\quad \Sigma(l) = x \;\Rightarrow\; \widehat{\Phi}(x, \widehat{me}(x)) \sqsubseteq \widehat{\Phi}(l, \widehat{m})$$

$$\textbf{(r 4.3)}\quad \Sigma(l) \in \mathit{Fun} \;\Rightarrow\; \{[l,\ \widehat{me}]\} \subseteq \Pi_{\lambda}(\widehat{\Phi}(l, \widehat{m}))$$

$$\textbf{(r 4.4)}\quad \Sigma(l) = (p\ t_1^{l_1} \ldots t_n^{l_n}) \;\Rightarrow\; \begin{cases} \langle l_i, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}) \quad i = 1, \ldots, n \\ \hat{p}(\widehat{\Phi}(l_1, \widehat{m}), \ldots, \widehat{\Phi}(l_n, \widehat{m})) \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \end{cases}$$

$$\textbf{(r 4.5)}\quad \Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n}) \;\Rightarrow$$
$$\begin{cases} \langle l_i, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}) \quad i = 0, \ldots, n \\ \forall [l_f,\ \widehat{me}_0] \in \Pi_{\lambda}(\widehat{\Phi}(l_0)) : \\ \begin{cases} (\Sigma(l_f) = (\mathtt{fn}\, x_1 \ldots x_n \Rightarrow t_b^{l_b}) \;\wedge\; \widehat{me}'_0 = \widehat{me}_0[x_i \mapsto \widehat{m}']) \vee \\ (\Sigma(l_f) = (\mathtt{fun}\, y\ x_1 \ldots x_n \Rightarrow t_b^{l_b})) \;\wedge\; \widehat{me}'_0 = \widehat{me}_0[y \mapsto \widehat{m}'][x_i \mapsto \widehat{m}']) \Rightarrow \\ \quad \begin{cases} \widehat{m}' = \widehat{\mathcal{I}}(\Sigma, l, \widehat{m}, l_f) \\ \langle l_b, \widehat{me}'_0\rangle \models^{\widehat{m}'} (\Sigma, \widehat{\Phi}) \\ \widehat{\Phi}(l_i, \widehat{m}) \sqsubseteq \widehat{\Phi}(x_i, \widehat{m}') \quad i = 1, \ldots, n \\ \widehat{\Phi}(l_b, \widehat{m}') \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \end{cases} \\ \Sigma(l_f) = (\mathtt{fun}\, y\ x_1 \ldots x_n \Rightarrow t_b^{l_b}) \Rightarrow \{[l_f,\ \widehat{me}_0]\} \subseteq \Pi_{\lambda}(\widehat{\Phi}(y, \widehat{m}')) \end{cases} \end{cases}$$

$$\textbf{(r 4.6)}\quad \Sigma(l) = (\mathtt{if}\ t_0^{l_0}\ \mathtt{then}\ t_1^{l_1}\ \mathtt{else}\ t_2^{l_2}) \Rightarrow$$
$$\begin{cases} \langle l_i, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}) \quad i \in \{0, 1, 2\} \\ \beta_{\mathsf{c}}(\mathit{true}) \sqsubseteq \Pi_{\mathsf{c}}(\widehat{\Phi}(l_0, \widehat{m})) \;\Rightarrow\; \widehat{\Phi}(l_1, \widehat{m}) \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \\ \beta_{\mathsf{c}}(\mathit{false}) \sqsubseteq \Pi_{\mathsf{c}}(\widehat{\Phi}(l_0, \widehat{m})) \;\Rightarrow\; \widehat{\Phi}(l_2, \widehat{m}) \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \end{cases}$$

$$\textbf{(r 4.7)}\quad \Sigma(l) = (\mathtt{let}\ x = t_1^{l_1}\ \mathtt{in}\ t_2^{l_2}) \Rightarrow \begin{cases} \langle l_1, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}) \\ \langle l_2, \widehat{me}[x \mapsto \widehat{m}']\rangle \models^{\widehat{m}'} (\Sigma, \widehat{\Phi}) \\ \widehat{\Phi}(l_1, \widehat{m}) \sqsubseteq \widehat{\Phi}(x, \widehat{m}') \\ \widehat{\Phi}(l_2, \widehat{m}') \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \end{cases} \wedge\; \widehat{m}' = \widehat{\mathcal{I}}(\Sigma, l, \widehat{m})$$

Figure 3.2: Control and Data Flow Analysis

Finally, the relation between the nodes of the derivation trees of $[\langle l, me\rangle \models^{m} (\Sigma, \Phi)]$ and $[\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})]$ is defined by:

$$(\langle l, me\rangle \models^{m} (\Sigma, \Phi)) \;\; \mathcal{R}^{cfa}_{co} \;\; (\langle l', \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})) \;\Leftrightarrow\; \begin{cases} (\Sigma = \Sigma') \;\wedge\; (l = l') \\ me \;\; \mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}} \;\; \widehat{me} \\ \Phi(l, m) \;\; \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}} \;\; \widehat{\Phi}(l, \widehat{m}) \end{cases} \tag{3.7}$$

The correctness of the control flow analysis states that the control flow analysis safely approximates - according to $\mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}}$ - the collecting semantics of the language. Let $\preceq^{cfa}_{co}$ be the simulation induced by $\mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}}$:

Theorem 3.2.2 (Correctness of the Control Flow Analysis)
If $\langle l, me\rangle \models^{m} (\Sigma, \Phi)$, $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})$ *and* $(me \;\; \mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}} \;\; \widehat{me})$
then

$$[\langle l, me\rangle \models^{m} (\Sigma, \Phi)] \;\; \preceq^{cfa}_{co} \;\; [\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})]$$

Proof: *Induction on the structure of* $[\langle l, me\rangle \models^{m} (\Sigma, \Phi)]$.

The first three rules correspond to axioms. Since these rules generate derivation trees with no subtrees (sigle-node trees), according to the definition of simulation in equation 2.9, we only need to prove $(\langle l, me\rangle \models^{m} (\Sigma, \Phi)) \;\; \mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}} \;\; (\langle l', \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}))$*, that is,*

$$(me \;\; \mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}} \;\; \widehat{me}) \quad \wedge \quad \Phi(l, m) \;\; \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}} \;\; \widehat{\Phi}(l, \widehat{m})$$

Since we already have $(me \;\; \mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}} \;\; \widehat{me})$ *from the initial conditions we only need to show* $(\Phi(l, m) \;\; \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}} \;\; \widehat{\Phi}(l, \widehat{m}))$.

The first three rules are:

(r 3.1) $\Sigma(l) = c$ *and* $\Phi(l, m) = c$

By applying the definition of $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})$ *to* $\Sigma(l) = c$ *we get* $\beta_{\mathsf{c}}(c) \sqsubseteq \Pi_{\mathsf{c}}(\widehat{\Phi}(l, \widehat{m}))$ *which , using equation (3.6), implies* $c \;\; \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}} \;\; \widehat{\Phi}(l, \widehat{m})$*, i.e.,* $\Phi(l, m) \;\; \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}} \;\; \widehat{\Phi}(l, \widehat{m})$

(r 3.2) $\Sigma(l) = x$, $x \in dom(me)$ *and* $\Phi(l,m) = \Phi(x, me)$ **[f 3.1]**.

We proceed as follows:

using		*derive*	
$\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}) \ \wedge\ \Sigma(l) = x$	$\rightarrow$	$\widehat{\Phi}(x, \widehat{me}(x)) \sqsubseteq \widehat{\Phi}(l, \widehat{m})$	**[f 3.2]**
$(me\ \mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{me})$, $x \in dom(me)\ \wedge$			
eq. (3.6)	$\rightarrow$	$\Phi(x, me(x))\ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{\Phi}(x, \widehat{me}(x))$	**[f 3.3]**
lemma 3.2.1, **[f 3.2]** $\wedge$ **[f 3.3]**	$\rightarrow$	$\Phi(x, me(x))\ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{\Phi}(l, \widehat{m})$	

Thus, by using **[f 3.1]** *and the last result, we finally get* $\Phi(l,m)\ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{\Phi}(l,\widehat{m})$.

(r 3.3) $\Sigma(l) \in Fun$ *and* $\Phi(l,m) = [l,\ me]$ **[f 3.4]**

using		*derive*	
$\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}) \ \wedge\ \Sigma(l) \in Fun$	$\rightarrow$	$\{[l,\ \widehat{me}]\} \subseteq \Pi_\lambda(\widehat{\Phi}(l, \widehat{m}))$	**[f 3.5]**
equation (3.6), **[f 3.5]** $\wedge$ $(me\ \mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{me})$	$\rightarrow$	$[l,\ me]\ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{\Phi}(l, \widehat{m})$	

Thus, using **[f 3.4]** *and the last result, we finally derive* $\Phi(l,m)\ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{\Phi}(l,\widehat{m})$.

The rest of the rules generate derivation trees with subtrees. The proof is divided into two parts. In the first part, for each subtree $[\langle l', me'\rangle \models^{m'} (\Sigma, \Phi)]$ *of* $[\langle l, me\rangle \models^{m} (\Sigma, \Phi)]$ *we must find a subtree* $[\langle l', \widehat{me}'\rangle \models^{\widehat{m}'} (\Sigma, \widehat{\Phi})]$ *of* $[\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})]$ *such that*

$$[\langle l', me'\rangle \models^{m'} (\Sigma, \Phi)]\ \preceq^{cfa}_{co}\ [\langle l', \widehat{me}'\rangle \models^{\widehat{m}'} (\Sigma, \widehat{\Phi})]$$

This part is done by applying the inductive hypothesis to $[\langle l', me'\rangle \models^{m'} (\Sigma, \Phi)]$. *In order to do this we just need to show* $(me'\ \mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{me}')$.

The second part of the proof deals with showing that the roots of $[\langle l, me\rangle \models^{m} (\Sigma, \Phi)]$ *and* $[\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})]$ *are* $\mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}}$*-related. As before, we only need to show* $(\Phi(l,m)\ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{\Phi}(l,\widehat{m}))$.

We show the rules that correspond to the conditional and `fn`*-aplication cases:*

(r 3.7) $\Sigma(l) = (\texttt{if } t_0^{l_0} \texttt{ then } t_1^{l_1} \texttt{ else } t_2^{l_2})$ *and* $\Phi(l_0, m) = \mathit{true}$. *We have:*

$$i \in \{0,1\} : \ [\langle l_i, me\rangle \models^m (\Sigma, \Phi)] \quad [\mathbf{f\ 3.6}] \qquad \wedge \qquad \Phi(l, m) = \Phi(l_1, m) \quad [\mathbf{f\ 3.7}]$$

By applying the definition of $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})$ *to* $\Sigma(l) = (\texttt{if } t_0^{l_0} \texttt{ then } t_1^{l_1} \texttt{ else } t_2^{l_2})$ *we get:*

$$i \in \{0,1,2\} \quad [\langle l_i, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})] \qquad [\mathbf{f\ 3.8}]$$

$$\beta_{\mathsf{c}}(\mathit{true}) \sqsubseteq \Pi_{\mathsf{c}}(\widehat{\Phi}(l_0, \widehat{m})) \ \Rightarrow\ \widehat{\Phi}(l_1, \widehat{m}) \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \quad [\mathbf{f\ 3.9}]$$

Using $(me\ \ \mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}}\ \ \widehat{me})$ *and* [**f 3.8**], *we apply the inductive hypothesis to* $i \in \{0,1\}$: $[\langle l_0, me\rangle \models^m (\Sigma, \Phi)]$, *and derive:*

$$i \in \{0,1\} : \quad [\langle l_i, me\rangle \models^m (\Sigma, \Phi)] \ \preceq^{cfa}_{co}\ [\langle l_i, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})] \quad [\mathbf{f\ 3.10}]$$

which completes the first part of the proof. Now we need to show $(\Phi(l, m)\ \ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \ \widehat{\Phi}(l, \widehat{m}))$. *By applying the definitions of* $\preceq^{cfa}_{co}$ *and* $\mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}}$ *to* [**f 3.10**] *we get:*

$$i \in \{0,1\} : \quad (\Phi(l_i, m)\ \ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \ \widehat{\Phi}(l_i, \widehat{m})) \quad [\mathbf{f\ 3.11}]$$

By replacing $\Phi(l_0, m) = \mathit{true}$ *in* [**f 3.11**] *we get* $(\mathit{true}\ \ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \ \widehat{\Phi}(l_0, \widehat{m}))$, *that is,* $\beta_{\mathsf{c}}(\mathit{true}) \sqsubseteq \Pi_{\mathsf{c}}(\widehat{\Phi}(l_0, \widehat{m}))$ *(equation (3.6)). Then:*

using		*derive*	
[**f 3.11**] $\wedge\ i = 1$	$\rightarrow$	$(\Phi(l_1, m)\ \ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \ \widehat{\Phi}(l_1, \widehat{m}))$	[**f 3.12**]
$\beta_{\mathsf{c}}(\mathit{true}) \sqsubseteq \Pi_{\mathsf{c}}(\widehat{\Phi}(l_0, \widehat{m}))\ \wedge$ [**f 3.9**]	$\rightarrow$	$\widehat{\Phi}(l_1, \widehat{m}) \sqsubseteq \widehat{\Phi}(l, \widehat{m})$	[**f 3.13**]
lemma 3.2.1, [**f 3.12**] $\wedge$ [**f 3.13**]	$\rightarrow$	$(\Phi(l_1, m)\ \ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \ \widehat{\Phi}(l, \widehat{m}))$	

Thus, using the last result and [**f 3.7**], *we finally get* $(\Phi(l, m)\ \ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \ \widehat{\Phi}(l, \widehat{m}))$.

(r 3.5) $\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n})$ *and*

$$\begin{aligned} i \in \{0, \ldots, n\} :\ & [\langle l_i, me\rangle \models^m (\Sigma, \Phi)] \\ & \Phi(l_0, m) = [l_f,\ me_0] \\ & [\langle l_b, me'_0\rangle \models^{m'} (\Sigma, \Phi)] \\ i \in \{1, \ldots, n\} :\ & \Phi(x_i, m') = \Phi(l_i, m) \qquad [\mathbf{f\ 3.14}] \\ & \Phi(l, m) = \Phi(l_b, m') \qquad [\mathbf{f\ 3.15}] \end{aligned}$$

where $\Sigma(l_f) = (\mathtt{fn}\, x_1 \ldots x_n \Rightarrow t_b^{l_b})$ *and* $me'_0 = me_0[x_i \mapsto m']_1^n$. *By applying the definition of* $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})$ *to* $\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n})$ *we get:*

$$i \in \{0, \ldots, n\} : \quad \langle l_i, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})$$

Therefore, since $(me\ \ \mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}}\ \ \widehat{me})$, *we can apply the inductive hypothesis to* $[\langle l_i, me\rangle \models^{m} (\Sigma, \Phi)]$ *and derive:*

$$i \in \{0, \ldots, n\} : \quad [\langle l_i, me\rangle \models^{m} (\Sigma, \Phi)] \ \preceq^{cfa}_{co}\ [\langle l_i, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})] \quad [\mathbf{f\ 3.16}]$$

which, using the definitions of $\preceq^{cfa}_{co}$ *and* $\mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}}$, *implies:*

$$i \in \{0, \ldots, n\} : \quad \Phi(l_i, m)\ \ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \ \widehat{\Phi}(l_i, \widehat{m}) \quad [\mathbf{f\ 3.17}]$$

In particular, we know $\Phi(l_0, m)\ \ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \ \widehat{\Phi}(l_0, \widehat{m})$ *and* $\Phi(l_0, m) = [l_f,\ me_0]$. *Then, by applying equation (3.6) to* $[l_f,\ me_0]\ \ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \ \widehat{\Phi}(l_0, \widehat{m})$, *we derive:*

$$\exists [l_f,\ \widehat{me}_0] \in \Pi_\lambda(\widehat{\Phi}(l_0, \widehat{m})),\ me_0\ \ \mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}}\ \ \widehat{me}_0 \quad [\mathbf{f\ 3.18}]$$

Then, by applying $[l_f,\ \widehat{me}_0] \in \Pi_\lambda(\widehat{\Phi}(l_0, \widehat{m}))$ *to the definition of* $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})$ *for* $\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n})$ *and* $\Sigma(l_f) = (\mathtt{fn}\, x_1 \ldots x_n \Rightarrow t_b^{l_b})$, *we get:*

$$\begin{array}{rll} & [\langle l_b, \widehat{me}'_0\rangle \models^{\widehat{m}'} (\Sigma, \widehat{\Phi})] & \\ i \in \{1, \ldots, n\} : & \widehat{\Phi}(l_i, \widehat{m}) \sqsubseteq \widehat{\Phi}(x_i, \widehat{m}') & [\mathbf{f\ 3.19}] \\ & \widehat{\Phi}(l_b, \widehat{m}') \sqsubseteq \widehat{\Phi}(l, \widehat{m}) & [\mathbf{f\ 3.20}] \end{array}$$

where $\widehat{m}' = \widehat{\mathcal{I}}(\Sigma, l, \widehat{m}, l_f)$ *and* $\widehat{me}'_0 = \widehat{me}_0[x_i \mapsto \widehat{m}']_1^n)$.

We want to apply the inductive hypothesis to $[\langle l_b, me'_0\rangle \models^{m'} (\Sigma, \Phi)]$. *In order to do this we need to prove* $me'_0\ \ \mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}}\ \ \widehat{me}'_0$:

$$\forall x \in dom(me'_0).\ \Phi(x, me'_0(x))\ \ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \ \widehat{\Phi}(x, \widehat{me}'_0(x))$$

We have two cases. Let $x \in dom(me'_0)$:

- $x \neq x_i$, *for all* $i \in \{1,\dots,n\}$. *Then* $me'_0(x) = me_0(x)$ *and* $\widehat{me}'_0(x) = \widehat{me}_0(x)$. *By applying* $x \in dom(me_0)$ *to* [**f 3.18**] *we get* $\Phi(x, me_0(x))\ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{\Phi}(x, \widehat{me}_0(x))$, *that is,* $\Phi(x, me'_0(x))\ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{\Phi}(x, \widehat{me}'_0(x))$.

- $x = i$, *for some* $i \in \{1,\dots,n\}$. *Then* $me'_0(x) = m'$ *and* $\widehat{me}'_0(x) = \widehat{m}'$. *We need to prove* $\Phi(x_i, m')\ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{\Phi}(x_i, \widehat{m}')$. *Using* [**f 3.14**] *and* [**f 3.17**] *we get* $\Phi(x_i, m')\ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{\Phi}(l_i, \widehat{m})$. *By applying lemma 3.2.1 to* $\Phi(x_i, m')\ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{\Phi}(l_i, \widehat{m})$ *and* [**f 3.19**] *we derive* $\Phi(x_i, m')\ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{\Phi}(x_i, \widehat{m}')$.

Therefore $(me'_0\ \mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{me}'_0)$. *Now we can apply the inductive hypothesis to* $[\langle l_b, me'_0\rangle \models^m (\Sigma, \Phi)]$ *and, using* $[\langle l_b, \widehat{me}'_0\rangle \models^{\widehat{m}'} (\Sigma, \widehat{\Phi})]$, *derive:*

$$[\langle l_b, me'_0\rangle \models^{m'} (\Sigma, \Phi)]\ \preceq^{cfa}_{co}\ [\langle l_b, \widehat{me}'_0\rangle \models^{\widehat{m}'} (\Sigma, \widehat{\Phi})] \quad [\mathbf{f\ 3.21}]$$

which, together with [**f 3.16**], *complete the first part of the proof. Now we need to prove* $\Phi(l, m)\ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{\Phi}(l, \widehat{m})$. *We have:*

using		*derive*	
$\preceq^{cfa}_{co}, \mathcal{R}^{cfa}_{\Phi,\widehat{\Phi}} \wedge$ [**f 3.21**]	$\rightarrow$	$\Phi(l_b, m')\ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{\Phi}(l_b, \widehat{m}')$	[**f 3.22**]
[**f 3.15**] *in* [**f 3.22**]	$\rightarrow$	$\Phi(l, m)\ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{\Phi}(l_b, \widehat{m}')$	[**f 3.23**]
lemma 3.2.1, [**f 3.23**] $\wedge$ [**f 3.20**]	$\rightarrow$	$\Phi(l, m)\ \mathcal{V}^{cfa}_{\Phi,\widehat{\Phi}}\ \widehat{\Phi}(l, \widehat{m})$	

which completes the proof.

[**QED**]

3.2.2 Correctness of the Analysis with respect to the Standard Semantics

Our next step is to relate the executions of the standard semantics and the control and data flow analysis. We define the $\mathcal{V}^{cfa}_{\widehat{\Phi}}$ and $\mathcal{R}^{cfa}_{\widehat{\Phi}}$ relations for this purpose. Given abstract cache

$\widehat{\Phi}$, $\mathcal{V}^{cfa}_{\widehat{\Phi}}$ and $\mathcal{R}^{cfa}_{\widehat{\Phi}}$ are the smallest relations that satisfy:

$$\begin{array}{rcl} \rho \ \mathcal{R}^{cfa}_{\widehat{\Phi}} \ \widehat{me} & \Leftrightarrow & \forall x \in dom(\rho),\ \rho(x) \ \mathcal{V}^{cfa}_{\widehat{\Phi}} \ \widehat{\Phi}(x, \widehat{me}(x)) \\ c \ \mathcal{V}^{cfa}_{\widehat{\Phi}} \ \hat{v} & \Leftrightarrow & \beta_{\mathsf{c}}(c) \sqsubseteq \Pi_{\mathsf{c}}(\hat{v}) \\ [l,\ \rho] \ \mathcal{V}^{cfa}_{\widehat{\Phi}} \ \hat{v} & \Leftrightarrow & \exists [l,\ \widehat{me}] \in \Pi_{\lambda}(\hat{v}),\ (\rho \ \mathcal{R}^{cfa}_{\widehat{\Phi}} \ \widehat{me}) \end{array} \tag{3.8}$$

Lemma 3.2.3 *If $v \ \mathcal{V}^{cfa}_{\widehat{\Phi}} \ \hat{v}$ and $\hat{v} \sqsubseteq \hat{v}'$ then $v \ \mathcal{V}^{cfa}_{\widehat{\Phi}} \ \hat{v}'$.*

As usual, the relation is extended to configurations and nodes:

$$\begin{array}{rcl} \langle l, \rho \rangle \ \mathcal{R}^{cfa}_{\widehat{\Phi}} \ \langle l', \widehat{me} \rangle & \Leftrightarrow & (l = l') \ \wedge \ (\rho \ \mathcal{R}^{cfa}_{\widehat{\Phi}} \ \widehat{me}) \\ (\Sigma \vdash \langle l, \rho \rangle \Downarrow v) \ \mathcal{R}^{cfa} \ (\langle l', \widehat{me} \rangle \vdash^{\widehat{m}} (\Sigma', \widehat{\Phi})) & \Leftrightarrow & \begin{cases} (\Sigma = \Sigma') \\ (\langle l, \rho \rangle \ \mathcal{R}^{cfa}_{\widehat{\Phi}} \ \langle l', \widehat{me} \rangle) \\ (v \ \mathcal{V}^{cfa}_{\widehat{\Phi}} \ \widehat{\Phi}(l', \widehat{m})) \end{cases} \end{array} \tag{3.9}$$

It is also useful - and appropriate - to read the judgement $\langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})$ as "an analysis" of the expression labelled by l in universe Σ. We say then that $\widehat{\Phi}$ is the result of the analysis and claim that it represents a consistent description of the executions of l under environments consistently described by $(\widehat{me}, \widehat{\Phi})$ under context $\widehat{m}$. We make this precise by defining:

$$\langle l, \rho \rangle \models^{\widehat{m}}_{\widehat{me}} (\Sigma, \widehat{\Phi}) \ \Leftrightarrow \ \langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}) \ \wedge \ (\rho \ \mathcal{R}^{cfa}_{\widehat{\Phi}} \ \widehat{me}) \tag{3.10}$$

and read $\langle l, \rho \rangle \models^{\widehat{m}}_{\widehat{me}} (\Sigma, \widehat{\Phi})$ as an analysis of configuration $\langle l, \rho \rangle$.

The correctness of the analysis states that the analysis represents a consistent description of the evaluation of the configuration being analyzed. In other words, that the analysis safely approximates the derivation tree of the evaluation of the configuration, *i.e.*, both derivation trees form a $\mathcal{R}^{cfa}_{\widehat{\Phi}}$-simulation:

Theorem 3.2.4 *Let $\preceq^{cfa}$ be the simulation induced by $\mathcal{R}^{cfa}$.*
If $[\langle l, \rho \rangle \models^{\widehat{m}}_{\widehat{me}} (\Sigma, \widehat{\Phi})]$ and $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v]$ then $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \ \preceq^{cfa} \ [\langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})]$

Proof: *By rule induction on* $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v]$.

Using equation (3.10) we get $[\langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})]$ *and* $(\rho \ \mathcal{R}^{cfa}_{\widehat{\Phi}} \ \widehat{me})$. *As we did in the proof of lemma 3.2.2, we just need to show* $v \ \mathcal{V}^{cfa}_{\widehat{\Phi}} \ \widehat{\Phi}(l, \widehat{m})$.

We show the first three axioms:

(r 1.1) $\Sigma(l) = c$ *and* $v = c$.

By applying the definition of $[\langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})]$ *to* $\Sigma(l) = c$ *we get* $\beta_{\mathsf{c}}(c) \sqsubseteq \Pi_{\mathsf{c}}(\widehat{\Phi}(l, \widehat{m}))$. *Using equation (3.8) we get* $c \ \mathcal{V}^{cfa}_{\widehat{\Phi}} \ \widehat{\Phi}(l, \widehat{m})$, *i.e.,* $v \ \mathcal{V}^{cfa}_{\widehat{\Phi}} \ \widehat{\Phi}(l, \widehat{m})$

(r 1.2) $\Sigma(l) = x$, $x \in dom(\rho)$ *and* $\rho(x) = v$ [**f 4.1**].

We proceed as follows:

using		*derive*	
$[\langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})] \ \wedge \ \Sigma(l) = x$	$\rightarrow$	$\widehat{\Phi}(x, \widehat{me}(x)) \sqsubseteq \widehat{\Phi}(l, \widehat{m})$	[**f 4.2**]
$(\rho \ \mathcal{R}^{cfa}_{\widehat{\Phi}} \ \widehat{me}), x \in dom(\rho) \ \wedge$ *eq. (3.8)*	$\rightarrow$	$\rho(x) \ \mathcal{V}^{cfa}_{\widehat{\Phi}} \ \widehat{\Phi}(x, \widehat{me}(x))$	[**f 4.3**]
lemma 3.2.3, [**f 4.2**] $\wedge$ [**f 4.3**]	$\rightarrow$	$\rho(x) \ \mathcal{V}^{cfa}_{\widehat{\Phi}} \ \widehat{\Phi}(l, \widehat{m})$	

Thus, by using [**f 4.1**] *and the last result, we finally get* $(v \ \mathcal{V}^{cfa}_{\widehat{\Phi}} \ \widehat{\Phi}(l, \widehat{m}))$.

(r 1.3) $\Sigma(l) \in Fun$ *and* $v = [l, \rho]$ [**f 4.4**].

using		*derive*	
$[\langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})] \ \wedge \ \Sigma(l) \in Fun$	$\rightarrow$	$\{[l, \widehat{me}]\} \subseteq \Pi_{\lambda}(\widehat{\Phi}(l, \widehat{m}))$	[**f 4.5**]
equation (3.8), [**f 4.5**] $\wedge$ $(\rho \ \mathcal{R}^{cfa}_{\widehat{\Phi}} \ \widehat{me})$	$\rightarrow$	$[l, \rho] \ \mathcal{V}^{cfa}_{\widehat{\Phi}} \ \widehat{\Phi}(l, \widehat{m})$	

Thus, using [**f 4.4**] *and the last result, we finally derive* $(v \ \mathcal{V}^{cfa}_{\widehat{\Phi}} \ \widehat{\Phi}(l, \widehat{m}))$.

The other rules deal with trees of height greater than one. In such cases, the simulation relation requieres that all the subtrees of $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v]$ *form a simulation with a subtree of* $[\langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})]$. *We show in detail the rule that corresponds to* `fn`*-application:*

(r 1.5) $\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n})$ *and:*

$$\begin{aligned}
&[\Sigma \vdash \langle l_i, \rho \rangle \Downarrow v_i] \quad i = 1, \ldots, n \ \wedge \\
&[\Sigma \vdash \langle l_0, \rho \rangle \Downarrow [l_f,\ \rho_0]] \ \wedge \\
&\Sigma(l_f) = (\texttt{fn}\ x_1 \ldots x_n \Rightarrow t_b^{l_b}) \ \wedge \\
&[\Sigma \vdash \langle l_b, \rho_0' \rangle \Downarrow v]
\end{aligned}$$

where $\rho_0' = \rho_0[x_1 \mapsto v_1] \ldots [x_n \mapsto v_n]$.

By applying the definition of $[\langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})]$ *to* $\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n})$ *we get:*

$$i \in \{0, \ldots, n\} : \quad \langle l_i, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})$$

Therefore, since $(\rho\ \ \mathcal{R}_{\widehat{\Phi}}^{cfa}\ \ \widehat{me})$, *we can apply the inductive hypothesis to* $[\Sigma \vdash \langle l_i, me \rangle \Downarrow v_i]$ *and derive:*

$$\begin{aligned}
&[\Sigma \vdash \langle l_0, \rho \rangle \Downarrow [l_f,\ \rho_0]] \ \preceq^{cfa}\ [\langle l_0, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})] \quad [\textbf{f 4.6}] \\
i \in \{1, \ldots, n\} : \quad &[\Sigma \vdash \langle l_i, \rho \rangle \Downarrow v_i] \ \preceq^{cfa}\ [\langle l_i, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})] \qquad [\textbf{f 4.7}]
\end{aligned}$$

which, using the definitions of $\preceq^{cfa}$ *and* $\mathcal{R}_{\widehat{\Phi}}^{cfa}$, *imply:*

$$\begin{aligned}
&[l_f,\ \rho_0]\ \ \mathcal{V}_{\widehat{\Phi}}^{cfa}\ \ \widehat{\Phi}(l_0, \widehat{m}) \quad [\textbf{f 4.8}] \\
i \in \{1, \ldots, n\} : \quad &v_i\ \ \mathcal{V}_{\widehat{\Phi}}^{cfa}\ \ \widehat{\Phi}(l_i, \widehat{m}) \qquad [\textbf{f 4.9}]
\end{aligned}$$

Therefore, by applying equation (3.8) to [**f 4.8**] *we get:*

$$\exists [l_f,\ \widehat{me}_0] \in \Pi_\lambda(\widehat{\Phi}(l_0, \widehat{m})),\ \rho_0\ \ \mathcal{R}_{\widehat{\Phi}}^{cfa}\ \ \widehat{me}_0 \quad [\textbf{f 4.10}]$$

Then, by applying $[l_f,\ \widehat{me}_0] \in \Pi_\lambda(\widehat{\Phi}(l_0, \widehat{m}))$ *to the definition of* $\langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})$ *for* $\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n})$ *and* $\Sigma(l_f) = (\texttt{fn}\ x_1 \ldots x_n \Rightarrow t_b^{l_b})$, *we get:*

$$\begin{aligned}
&[\langle l_b, \widehat{me}_0' \rangle \models^{\widehat{m}'} (\Sigma, \widehat{\Phi})] \\
i \in \{1, \ldots, n\} : \quad &\widehat{\Phi}(l_i, \widehat{m}) \sqsubseteq \widehat{\Phi}(x_i, \widehat{m}') \qquad [\textbf{f 4.11}] \\
&\widehat{\Phi}(l_b, \widehat{m}') \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \qquad [\textbf{f 4.12}]
\end{aligned}$$

where $\widehat{m}' = \widehat{\mathcal{I}}(\Sigma, l, \widehat{m}, l_f)$ *and* $\widehat{me}_0' = \widehat{me}_0[x_i \mapsto \widehat{m}']_1^n)$.

We want to apply the inductive hypothesis to $[\Sigma \vdash \langle l_b, \rho'_0\rangle \Downarrow v]$. *In order to do this we need to prove* $\rho'_0 \;\mathcal{R}^{cfa}_{\widehat{\Phi}}\; \widehat{me}'_0$:

$$\forall x \in dom(\rho'_0).\,(\rho'_0(x) \;\mathcal{V}^{cfa}_{\widehat{\Phi}}\; \widehat{\Phi}(x, \widehat{me}'_0(x)))$$

We have two cases. Let $x \in dom(\rho'_0)$:

- $x \neq x_i$, *for all* $i \in \{1, \ldots, n\}$. *Then* $\rho'_0(x) = \rho_0(x)$ *and* $\widehat{me}'_0(x) = \widehat{me}_0(x)$. *By applying* $x \in dom(\rho_0)$ *to* [**f 4.10**] *we get* $\rho_0(x) \;\mathcal{V}^{cfa}_{\widehat{\Phi}}\; \widehat{\Phi}(x, \widehat{me}_0(x))$, *that is,* $\rho'_0(x) \;\mathcal{V}^{cfa}_{\widehat{\Phi}}\; \widehat{\Phi}(x, \widehat{me}'_0(x))$.
- $x = i$, *for some* $i \in \{1, \ldots, n\}$. *Then* $\rho'_0(x) = v_i$ *and* $\widehat{me}'_0(x) = \widehat{m}'$. *We need to prove* $v_i \;\mathcal{V}^{cfa}_{\widehat{\Phi}}\; \widehat{\Phi}(x_i, \widehat{m}')$. *Using* [**f 4.9**] *we get* $(v_i \;\mathcal{V}^{cfa}_{\widehat{\Phi}}\; \widehat{\Phi}(l_i, \widehat{m}))$. *By applying lemma 3.2.3 to* $(v_i \;\mathcal{V}^{cfa}_{\widehat{\Phi}}\; \widehat{\Phi}(l_i, \widehat{m}))$ *and* [**f 4.11**] *we derive* $(v_i \;\mathcal{V}^{cfa}_{\widehat{\Phi}}\; \widehat{\Phi}(x_i, \widehat{m}'))$.

We have shown $\rho'_0 \;\mathcal{R}^{cfa}_{\widehat{\Phi}}\; \widehat{me}'_0$. *Now we can apply the inductive hypothesis to* $[\Sigma \vdash \langle l_b, \rho'_0\rangle \Downarrow v]$ *and, using* $[\langle l_b, \widehat{me}'_0\rangle \models^{\widehat{m}'} (\Sigma, \widehat{\Phi})]$, *derive:*

$$[\Sigma \vdash \langle l_b, \rho'_0\rangle \Downarrow v] \;\preceq^{cfa}\; [\langle l_b, \widehat{me}'_0\rangle \models^{\widehat{m}'} (\Sigma, \widehat{\Phi})] \quad [\mathbf{f\ 4.13}]$$

which, together with [**f 4.6**] *and* [**f 4.7**], *completes the first part of the proof. Now we need to prove* $v \;\mathcal{V}^{cfa}_{\widehat{\Phi}}\; \widehat{\Phi}(l, \widehat{m})$. *We have:*

using		*derive*
$\preceq^{cfa}, \mathcal{R}^{cfa}_{\widehat{\Phi}} \wedge$ [**f 4.13**]	$\to$	$v \;\mathcal{V}^{cfa}_{\widehat{\Phi}}\; \widehat{\Phi}(l_b, \widehat{m}')$ [**f 4.14**]
lemma 3.2.3, [**f 4.14**] $\wedge$ [**f 4.12**]	$\to$	$v \;\mathcal{V}^{cfa}_{\widehat{\Phi}}\; \widehat{\Phi}(l, \widehat{m})$

which completes the proof.

[**QED**]

Corollary 3.2.5

If $\langle l, \rho\rangle \models^{\widehat{m}}_{\widehat{me}} (\Sigma, \widehat{\Phi})$ *and* $\Sigma \vdash \langle l, \rho\rangle \Downarrow v$ *then* $v \;\mathcal{V}^{cfa}_{\widehat{\Phi}}\; (\widehat{\Phi}(l, \widehat{m}), \widehat{\Phi})$.

3.3 0-CFA

The framework presented in section 3.2 allows us to choose from different abstractions for constants and contexts. In this section we derive the most common of the flow analyses: the zero control flow analysis (0CFA).

A 0CFA considers that all expressions are evaluated in the same context: 0CFA does not distinguish between the evaluation of functions at different application points. We define the domain of abstract contexts as a singleton, with $\bullet$ as its only element, and a special abstract context environment $me^\bullet$ that maps all variables to $\bullet$:

$$
\begin{aligned}
\widehat{Context} &= \{\bullet\} \\
me^\bullet &= \lambda x.\, \bullet
\end{aligned}
$$

Having a single context $\bullet$ allows us to simplify the definitions of the abstract domains and most of the rules shown in Figure 3.2. Variables can only be bound to the $\bullet$ context. Therefore, for each analysis $\langle l, me\rangle \models^\bullet (\Sigma, \widehat{\Phi})$ we can write $\langle l, me^\bullet\rangle \models^\bullet (\Sigma, \widehat{\Phi})$ and replace all $[l,\ me]$ closures by $[l,\ me^\bullet]$. Furthermore, we can remove contexts all together from the definition. We get the following domains:

$$
\begin{aligned}
\widehat{\Phi} \in \widehat{Cache} &= (Lab \cup Var) \to \widehat{Val} \\
\widehat{Closure} &= \mathcal{P}(Lab)
\end{aligned}
$$

and write $\widehat{\Phi}(l)$ instead of $\widehat{\Phi}(l, \bullet)$ and $\widehat{\Phi}(x)$ instead of $\widehat{\Phi}(x, me^\bullet(x))$.

We throw away constants from the analysis. We do this by abstracting all constants to the single element $\bullet_c$. The domain of abstract values matches the domain of abstract closures:

$$
\begin{aligned}
\widehat{Val} &= \widehat{Closure} \\
\widehat{Const} &= \{\bullet_c\} \\
\beta_{\mathsf{c}} &= \lambda d.\, \bullet_c
\end{aligned}
$$

$\widehat{Val}$ remains consistent with the definition in section 3.2 by defining the projection functions as:

$$\Pi_{\mathsf{c}}(\hat{v}) = \bullet_c \quad \wedge \quad \Pi_\lambda(\hat{v}) = \hat{v}$$

An immediate consequence of the definitions above is that $\beta_{\mathsf{c}}(c) \sqsubseteq \Pi_{\mathsf{c}}(\hat{v})$ always. The relations $\mathcal{R}_\bullet$ and $\mathcal{V}_\bullet$ can be presented more succinctly as:

$$\begin{array}{rcl} \rho \ \mathcal{R}_\bullet \ \widehat{\Phi} & \Leftrightarrow & \forall x \in dom(\rho),\ \rho(x)\ \mathcal{V}_\bullet\ (\widehat{\Phi}(x), \widehat{\Phi}) \\ c\ \mathcal{V}_\bullet\ (\hat{v}, \widehat{\Phi}) & \Leftrightarrow & \text{always} \\ [l,\ \rho]\ \mathcal{V}_\bullet\ (\hat{v}, \widehat{\Phi}) & \Leftrightarrow & (l \in \hat{v}) \ \wedge \ \rho\ \mathcal{R}_\bullet\ \widehat{\Phi} \end{array} \tag{3.11}$$

Similarly, the rules that define $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})$ - Figure 3.2 - are specialized to a new set of rules - see Figure 3.3 - that take advantage of the properties of the new abstract domains. We write $l \models^\bullet (\Sigma, \widehat{\Phi})$ and define the 0CFA as the greatest relation that satisfies these rules. The judgement $l \models^\bullet (\Sigma, \widehat{\Phi})$ claims that $\widehat{\Phi}$ represents a consistent description of the evaluation of l under universe Σ and environments consistently described by $\widehat{\Phi}$. We make the latter condition precise by extending the relation to configurations:

$$\langle l, \rho\rangle \models^\bullet (\Sigma, \widehat{\Phi}) \ \Leftrightarrow \ l \models^\bullet (\Sigma, \widehat{\Phi}) \quad \wedge \quad (\rho\ \mathcal{R}_\bullet\ \widehat{\Phi})$$

Our definition only analyzes function bodies that are proved to flow to an application site. Therefore, some unreachable code will not be analyzed. We consider this a good thing. On the other hand, this style of analysis requires a co-inductive definition in order to be valid for all programs though the cases where we need to prove by co-induction are few. We favour this style because it mimics the natural semantics of the language.

Another style of analysis imposes the condition that the whole program should be analyzed. In this case, an inductive definition suffices.

3.3.1 Correctness of 0CFA

Given an analysis $\langle l, \rho\rangle \models^\bullet (\Sigma, \widehat{\Phi})$ of configuration $\langle l, \rho\rangle$, the correctness of the analysis states that every node of the derivation tree of the evaluation of $\langle l, \rho\rangle$ is safely simulated

$$l \models^{\bullet} (\Sigma, \widehat{\Phi}) \Leftrightarrow$$

$$(\mathbf{r\ 5.1}) \quad \Sigma(l) = c \Rightarrow \mathit{true}$$

$$(\mathbf{r\ 5.2}) \quad \Sigma(l) = x \Rightarrow \widehat{\Phi}(x) \subseteq \widehat{\Phi}(l)$$

$$(\mathbf{r\ 5.3}) \quad \Sigma(l) \in \mathit{Fun} \Rightarrow l \in \widehat{\Phi}(l)$$

$$(\mathbf{r\ 5.4}) \quad \Sigma(l) = (p\ t_1^{l_1} \ldots t_n^{l_n}) \Rightarrow l_i \models^{\bullet} (\Sigma, \widehat{\Phi}) \quad i = 1, \ldots, n$$

$$(\mathbf{r\ 5.5}) \quad \Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n}) \Rightarrow \begin{cases} l_i \models^{\bullet} (\Sigma, \widehat{\Phi}) \quad i = 0, \ldots, n \\ \forall l_f \in \widehat{\Phi}(l_0): \\ \quad \begin{cases} (\Sigma(l_f) = (\texttt{fn}\ x_1 \ldots x_n \Rightarrow t_b^{l_b}) \vee \\ \ \Sigma(l_f) = (\texttt{fun}\ y\ x_1 \ldots x_n \Rightarrow t_b^{l_b})) \Rightarrow \\ \quad \begin{cases} l_b \models^{\bullet} (\Sigma, \widehat{\Phi}) \\ \widehat{\Phi}(l_i) \subseteq \widehat{\Phi}(x_i) \quad i = 1, \ldots, n \\ \widehat{\Phi}(l_f) \subseteq \widehat{\Phi}(l) \end{cases} \\ \Sigma(l_f) = (\texttt{fun}\ y\ x_1 \ldots x_n \Rightarrow t_f^{l_f}) \Rightarrow l_f \in \widehat{\Phi}(y) \end{cases} \end{cases}$$

$$(\mathbf{r\ 5.6}) \quad \Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2}) \Rightarrow \begin{cases} l_i \models^{\bullet} (\Sigma, \widehat{\Phi}) \quad i \in \{0, 1, 2\} \\ \widehat{\Phi}(l_1) \subseteq \widehat{\Phi}(l) \\ \widehat{\Phi}(l_2) \subseteq \widehat{\Phi}(l) \end{cases}$$

$$(\mathbf{r\ 5.7}) \quad \Sigma(l) = (\texttt{let}\ x = t_1^{l_1}\ \texttt{in}\ t_2^{l_2}) \Rightarrow \begin{cases} l_1 \models^{\bullet} (\Sigma, \widehat{\Phi}) \\ l_2 \models^{\bullet} (\Sigma, \widehat{\Phi}) \\ \widehat{\Phi}(l_1) \subseteq \widehat{\Phi}(x) \\ \widehat{\Phi}(l_2) \subseteq \widehat{\Phi}(l) \end{cases}$$

Figure 3.3: Zero Control Flow Analysis

by the analysis. In other words, both derivation trees form a $\mathcal{R}_\bullet$-simulation:

Theorem 3.3.1 (Correctness of 0CFA)
If $\langle l, \rho \rangle \models^\bullet (\Sigma, \widehat{\Phi})$ *and* $\Sigma \vdash \langle l, \rho \rangle \Downarrow v$ *then* $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \preceq_\bullet [l \models^\bullet (\Sigma, \widehat{\Phi})]$.

The correctness of the control flow analysis says that the values evaluated during the evaluation of the program are predicted by $\widehat{\Phi}$. In particular, if a configuration evaluates to a closure, then the closure is predicted by the abstract cache:

Corollary 3.3.2 *If* $\langle l, \rho \rangle \models^\bullet (\Sigma, \widehat{\Phi})$ *and* $\Sigma \vdash \langle l, \rho \rangle \Downarrow \langle l', \rho' \rangle$ *then* $l' \in \widehat{\Phi}(l)$ *and* $(\rho \ \mathcal{R}_\bullet \ \widehat{\Phi})$.

3.4 The role of Control Flow Analysis

The role of control flow analysis goes beyond the need of finding an approximation of the closures evaluated during the evaluation of a program, information which can later be used for a particular program optimization. CFA is, indeed, an essential part of the process of calculating a variety of program analyses for higher-order languages.

Most classic program analyses are defined by a series of equations made at the nodes of the control flow graph of a program. In languages where functions are not first class values, getting the control flow graph is trivial since it relies basically on static information. In higher-order languages, the computation of the control flow graph relies on non-static information and, therefore, transporting those analyses to this new setting becomes non-trivial.

From this point of view, the role of the control flow analysis is to provide some version of the control flow graph where other analyses can be built. 0CFA provides the basic control flow graph - call graph - where all execution paths are merged. More precise analyses can be seen as unfoldings of this graph, each producing a more faithful representation of the actual execution of the program.

Hence, the most exact control flow analysis matches the derivation tree of the execution of a program. What we propose is to define program properties as constraint systems based on concrete derivation trees. Such systems become computable when the derivation is abstracted (we choose CFA in this dissertation) to a more succinct representation. The new constraint system based now on an abstract derivation tree is what we call "the analysis". We must prove that the properties accepted by the abstract definition are also accepted by the concrete definition.

The context CFA framework presented in this chapter is used to derive the analyses of chapters 4 and 5. Unfortunately we were unable to re-use the correctness result (theorem 3.2.4) because the slice analyses defined in the following chapters modify the original CFA. A modified version of theorem 3.2.4 is presented instead. On the other hand, the correctness of standard program analyses built directly on top (separately) of the CFA will depend on the initial CFA's correctness result.

Chapter 4

Replacing Useless Expressions

4.1 Motivation: Removing Useless Expressions

An expression is useless if its evaluation doesn't affect the final result of a program. This definition of useless expressions leads to another one, more concrete, which says that if we replace a useless expression by any constant in a program then the result, if it was initially a constant, should remain the same.

For example, in the program fragment below,

```
let x = (g 10)                    let x = 0
in                   ==>          in
  (f 5)                             (f 5)
```

if we replace the right-hand side of the `let`-expression by any constant - we will use 0 - then the result of the program doesn't change, as long as `(g 10)` terminates. According to our definition, `(g 10)` is a useless expression.

The notion of useless expressions includes expressions considered as dead code, *i.e.*, code that is never executed. For example, the code fragment shown below,

```
if true then (+ x 5)      ==>       if true then (+ x 5)
else (f x)                          else 0
```

can be replaced by the expression on its right[1]. The performance of the new program is not affected (number of steps) but its size is reduced.

The main source of useless expressions is the presence of useless parameters. Consider the following example:

```
let fun loop(a, bogus, j) =                let fun loop (a, bogus, j) =
     if (j > 100)                               if (j > 100)
         then a                     ==>             then a
     else loop(f(a,j), bogus+2, j+1)            else loop (f(a,j), 0, j+1)
in loop (a, (fact 50), 1)                  in loop (a, 0, 1)
end                                        end
```

It is clear in the program on the left-hand side that the second argument does not contribute to the value the computation. We can replace the second actual argument on all the calls to `loop`, as shown in the right-hand side.

Our definition also considers useless expressions in higher-order procedures. For example, consider:

```
let f1 = fn(x, y, z) x                    let f1 = fn(x, y, z) x
    f2 = fn(x, y, z) y                        f2 = fn(x, y, z) y
    f3 = fn(x, y, z) x + y       ==>          f3 = 0
    g = if p then f1 else f2                  g = if p then f1 else f2
    h = if q then f1 else f3                  h = 0
in g(x,y,h)                                in g(x,y,0)
```

[1] The whole `if`-expression can be replaced by its true-branch too but that falls into a different kind of transformation

Assuming that `p` takes both *true* and *false* values during the execution of the program, it is clear that `f1` and `f2` are the only functions that flow to the application site `g(x,y,h)`. `f1` does not use its second and third arguments and `f2` does not use its first and third argument. Combining the information of both functions, we infer that the third actual parameter of `g(x,y,h)` is useless and can be replaced. We observe that `h` does not contribute to the value of the computation. Something similar happens to `f3`. We can replace the right-hand side of the assignments, getting the second program.

We would like to define a transformation that, given a program $\Sigma(l_p) = P$, replaces its useless expressions with smaller, less computationally-expensive expressions e.g. constants.

4.2 The Transformation: Replacing Useless Expressions

We formalize the transformation described in the previous section. The mapping:

$$\mathcal{U} : Lab \ \rightarrow \ Bool,$$

is used as input to the transformation and marks the terms that are believed to contribute to the evaluation of the program[2].

We define the transformation $\mathcal{T}_{\mathcal{U}}^{d}$ as a function that replaces all the expressions whose labels are not in $\mathcal{U}$ - $\mathcal{U}(l)$ is *false* - with constant d. The transformation function is parameterized by the mapping $\mathcal{U}$ and the constant used to replace all expressions not in $\mathcal{U}$, *i.e.*, $\mathcal{T}_{\mathcal{U}}^{0}$, $\mathcal{T}_{\mathcal{U}}^{1}$ are valid transformations.

Definition 4.2.1 *Given a mapping $\mathcal{U}$, the transformation $\mathcal{T}_{\mathcal{U}}^{d}$ takes a universe and a label as input and produces a labelled term following the rules listed in Figure 4.1. We write $\mathcal{T}_{\mathcal{U}}^{d}(l)$ instead of $\mathcal{T}_{\mathcal{U}}^{d}(\Sigma, l)$ whenever Σ is clear from the context.*

[2] Alternatively, $\mathcal{U}$ can be read as a set of labels where $\mathcal{U}(l) \ \Leftrightarrow \ l \in \mathcal{U}$.

$$
\begin{array}{rcl}
\neg\mathcal{U}(l) & \Rightarrow & \mathcal{T}^d_{\mathcal{U}}(l) = \boldsymbol{d}^l \\
\mathcal{U}(l) & \Rightarrow &
\end{array}
$$

$$
\begin{array}{lcl}
\Sigma(l) = c & \Rightarrow & \mathcal{T}^d_{\mathcal{U}}(l) = c^l \\
\Sigma(l) = x & \Rightarrow & \mathcal{T}^d_{\mathcal{U}}(l) = x^l \\
\Sigma(l) = (\texttt{fn}\, x_1 \ldots x_n \Rightarrow t_b^{l_b}) & \Rightarrow & \mathcal{T}^d_{\mathcal{U}}(l) = (\texttt{fn}\, x_1 \ldots x_n \Rightarrow \mathcal{T}^d_{\mathcal{U}}(l_b))^l \\
\Sigma(l) = (\texttt{fun}\, y\; x_1 \ldots x_n \Rightarrow t_b^{l_b}) & \Rightarrow & \mathcal{T}^d_{\mathcal{U}}(l) = (\texttt{fun}\, y\; x_1 \ldots x_n \Rightarrow \mathcal{T}^d_{\mathcal{U}}(l_b))^l \\
\Sigma(l) = (p\; t_1^{l_1} \ldots t_n^{l_n}) & \Rightarrow & \mathcal{T}^d_{\mathcal{U}}(l) = (p\;\; \mathcal{T}^d_{\mathcal{U}}(l_1)\;\; \ldots\;\; \mathcal{T}^d_{\mathcal{U}}(l_n))^l \\
\Sigma(l) = (t_0^{l_0}\; t_1^{l_1} \ldots t_n^{l_n}) & \Rightarrow & \mathcal{T}^d_{\mathcal{U}}(l) = (\mathcal{T}^d_{\mathcal{U}}(l_0)\;\; \mathcal{T}^d_{\mathcal{U}}(l_1)\;\; \ldots\;\; \mathcal{T}^d_{\mathcal{U}}(l_n))^l \\
\Sigma(l) = (\texttt{if}\; t_0^{l_0}\; \texttt{then}\; t_1^{l_1}\; \texttt{else}\; t_2^{l_2}) & \Rightarrow & \mathcal{T}^d_{\mathcal{U}}(l) = (\texttt{if}\; \mathcal{T}^d_{\mathcal{U}}(l_0)\; \texttt{then}\; \mathcal{T}^d_{\mathcal{U}}(l_1)\; \texttt{else}\; \mathcal{T}^d_{\mathcal{U}}(l_2))^l \\
\Sigma(l) = (\texttt{let}\; x = t_1^{l_1}\; \texttt{in}\; t_2^{l_2}) & \Rightarrow & \mathcal{T}^d_{\mathcal{U}}(l) = (\texttt{let}\; x = \mathcal{T}^d_{\mathcal{U}}(l_1)\; \texttt{in}\; \mathcal{T}^d_{\mathcal{U}}(l_2))^l
\end{array}
$$

Figure 4.1: Transformation: Replacing Useless Expressions

We extend the transformation to operate on universes:

$$\mathcal{T}^d_{\mathcal{U}}(\Sigma) \;=\; \{\mathcal{T}^d_{\mathcal{U}}(l) \mid l \in \mathrm{dom}(\Sigma)\}$$

The transformation keeps the labels of the original expressions. Thus, if $\Sigma(l) = t$ and $\mathcal{U}(l)$ is false then $\mathcal{T}^d_{\mathcal{U}}(l) = d^l$ and $\mathcal{T}^d_{\mathcal{U}}(\Sigma)(l) = d$.

We want to find $\mathcal{U}$ such that the meaning of the program is preserved by the transformation.

4.3 Useless Expressions and Program Slices

A program slice consists of the parts of the program that *may* affect the values computed at some point of interest. The point of interest in question is referred as the *slicing criterion* and it determines what parts of the evaluation of the program are to be inspected or considered. Program slicing is used in a variety of applications, which include program debugging and program specialization. A great deal of research has been dedicated to program slicing, mainly in the context of imperative programming languages [Wei84, HRB84, Ven91, Bis97b]. Program slicing techniques have also been adapted and developed for functional and higher-order languages [RT96, Bis97b].

In an imperative language, the slicing criterion is usually defined by a pair (p, V), where V is a set of variables and p a program point. A program slice is defined as a set of statements that affect the values mapped to the variables in V at program point p. This is usually modeled by removing all the statements outside the slice from the program and checking that the *slice* is still executable and displays the same behavior as the original: the values mapped to all $x \in V$ at program point p are the same. This notion of slice is usually called backward slicing.

In a functional, higher-order language the process of removing expressions from a program is no longer guaranteed to produce a syntactically valid program. The slicing criterion, determined by values mapped to variables at a particular program point, is not appropriate either. The concept of slicing has to be adapted to this new setting. We follow the style used by Biswas in [Bis97b]. A program slice is modeled by reasoning about different versions of the program where expressions outside the slice are *replaced*, not removed, by a dummy element. If the evaluation of a program is defined by a big-step semantics $\Sigma \vdash \mathcal{C} \Downarrow v$ then a slice should produce the same value v. The result of the evaluation of a program is the new slicing criterion.

Program slicing can be regarded also as the process of finding useless expressions: expressions outside a program slice are deemed useless. It is clear that our transformation, when fed with a program slice, should produce a program that exhibits the same behavior as the original. The task of finding $\mathcal{U}$ such that $\mathcal{T}^d_{\mathcal{U}}(l)$ exhibits the same behavior as l is reduced to finding a valid slice of the evaluation of l. The rest of this section is dedicated to the formal definition of slices and the development of methods (analyses) to compute them.

4.3.1 Slice Operational Semantics and Relevant Slices

A program slice consists of the set of expressions that might affect the evaluation of a program. We generalize the concept of slices to configurations. Given $\Sigma \vdash \langle l, \rho \rangle \Downarrow v$, a program slice of configuration $\langle l, \rho \rangle$ is defined as the set of labels $\mathcal{U} \subseteq \mathcal{L}(\Sigma, l)$ that may affect v. As mentioned in the previous section, we have chosen the evaluated value v as our slicing criterion.

A key point in the definition of program slices is to make precise the concept of an expression "affecting the final value of a program". In particular, we are interested with the following concept of program slice:

> Given a terminating program, if the evaluations of the expressions outside the slice of a program are replaced by **any other value** then the final value of the program is not affected.

That is, a program slice is determined by subjecting the program to a special test where the evaluation of all expressions outside the slice candidate - or the expressions themselves - are replaced by any other possible value. If the value v remains "unchanged" then the candidate is a valid slice.

We introduce the new constant $\star$ to model all possible executions of a slice of a program for all possible substitutions. We modify the standard semantics such that expressions outside the program slice always evaluate to $\star$. The new relation is called the **slice operational**

semantics. A slight modification to the semantic domains has to be made:

$$\begin{aligned}
\dot{v} \in \mathit{Val}^\star &= \mathit{Const}^\star \cup \mathit{Closure}^\star \\
\dot{c} \in \mathit{Const}^\star &= \mathit{Const} \cup \{\star\} \\
[l,\ \dot{\rho}] \in \mathit{Closure}^\star &= \mathit{Lab} \times \mathit{Env}^\star \\
\dot{\rho} \in \mathit{Env}^\star &= \mathit{Var} \rightarrow \mathit{Val}^\star \\
\langle l, \dot{\rho} \rangle \in \mathit{Config}^\star &= \mathit{Lab} \times \mathit{Env}^\star
\end{aligned}$$

The structure of environments, closures and configurations hasn't changed. The only difference is that a value in $\mathit{Val}^\star$ can be $\star$ and that variables in environments can be bound to $\star$ as well. Note that $\mathit{Val}^\star = \mathit{Val} \cup \{\star\}$ won't suffice because it doesn't consider closures with environments mapping to $\star$.

Definition 4.3.1 (Slice Operational Semantics)
$(\Sigma, \mathcal{U}) \vdash \langle l, \dot{\rho} \rangle \Downarrow^\star \dot{v}$ *is defined as the least relation that satisfies the rules in Figure 4.2.*

Expressions marked[3] by $\mathcal{U}$ evaluate in a similar way as defined by the standard evaluation relation $\Sigma \vdash \langle l, \rho \rangle \Downarrow v$. Unmarked expressions - $\mathcal{U}(l) = \mathit{false}$ - are treated differently: they always evaluate to the new constant $\star$. This is enforced by the rule:

$$\textbf{(r 6.1)} \quad \frac{\neg \mathcal{U}(l)}{(\Sigma, \mathcal{U}) \vdash \langle l, \dot{\rho} \rangle \Downarrow^\star \star}$$

Note, however, that no new rules have been added to manipulate the new constant and, therefore, fewer programs will successfully terminate.

This is not a new technique. Vinkatesh [Ven91] uses a similar approach but applied to a denotational setting for an imperative language. Biswas [Bis97b] creates new versions of the program by replacing useless terms with a dummy constant. We do something similar, we introduce the dummy constants during the evaluation of the program..

We have defined the test mechanism to identify valid program slices: the slice operational semantics. We are left with the task of formalizing what we mean by "v remains

[3]As before, we write $\mathcal{U}(l) \Leftrightarrow l \in \mathcal{U}$

$$\textbf{(r 6.1)}\quad \frac{\neg\mathcal{U}(l)}{(\Sigma,\mathcal{U}) \vdash \langle l, \dot\rho\rangle \Downarrow^\star \star}$$

$$\textbf{(r 6.2)}\quad \frac{\Sigma(l) = c \ \wedge\ \mathcal{U}(l)}{(\Sigma,\mathcal{U}) \vdash \langle l, \dot\rho\rangle \Downarrow^\star c}$$

$$\textbf{(r 6.3)}\quad \frac{\begin{array}{l}\Sigma(l) = x \ \wedge\ \mathcal{U}(l)\\ x \in dom(\dot\rho)\end{array}}{(\Sigma,\mathcal{U}) \vdash \langle l, \dot\rho\rangle \Downarrow^\star \dot\rho(x)}$$

$$\textbf{(r 6.4)}\quad \frac{\Sigma(l) \in Fun \ \wedge\ \mathcal{U}(l)}{(\Sigma,\mathcal{U}) \vdash \langle l, \dot\rho\rangle \Downarrow^\star [l,\ \dot\rho]}$$

$$\textbf{(r 6.5)}\quad \frac{\begin{array}{l}\Sigma(l) = (p\ t_1^{l_1} \ldots t_n^{l_n}) \ \wedge\ \mathcal{U}(l)\\ (\Sigma,\mathcal{U}) \vdash \langle l_i, \dot\rho\rangle \Downarrow^\star c_i \quad 0 \le i \le n\end{array}}{(\Sigma,\mathcal{U}) \vdash \langle l, \dot\rho\rangle \Downarrow^\star \bar{p}(c_1 \ldots c_n)}$$

$$\textbf{(r 6.6)}\quad \frac{\begin{array}{l}\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n}) \ \wedge\ \mathcal{U}(l)\\ (\Sigma,\mathcal{U}) \vdash \langle l_i, \dot\rho\rangle \Downarrow^\star \dot v_i \quad 1 \le i \le n\\ (\Sigma,\mathcal{U}) \vdash \langle l_0, \dot\rho\rangle \Downarrow^\star [l_f,\ \dot\rho_0]\\ \Sigma(l_f) = (\texttt{fn}\ x_1 \ldots x_n \Rightarrow t_b^{l_b})\\ \dot\rho' = \dot\rho_0[x_i \mapsto \dot v_i]_1^n\\ (\Sigma,\mathcal{U}) \vdash \langle l_b, \dot\rho'\rangle \Downarrow^\star \dot v\end{array}}{(\Sigma,\mathcal{U}) \vdash \langle l, \dot\rho\rangle \Downarrow^\star \dot v}$$

$$\textbf{(r 6.7)}\quad \frac{\begin{array}{l}\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n}) \ \wedge\ \mathcal{U}(l)\\ (\Sigma,\mathcal{U}) \vdash \langle l_i, \dot\rho\rangle \Downarrow^\star \dot v_i \quad 1 \le i \le n\\ (\Sigma,\mathcal{U}) \vdash \langle l_0, \dot\rho\rangle \Downarrow^\star [l_f,\ \dot\rho_0]\\ \Sigma(l_f) = (\texttt{fun}\ y\ x_1 \ldots x_n \Rightarrow t_b^{l_b})\\ \dot\rho' = \dot\rho_0[y \mapsto \langle l_f, \dot\rho_0\rangle][x_i \mapsto \dot v_i]_1^n\\ (\Sigma,\mathcal{U}) \vdash \langle l_b, \dot\rho'\rangle \Downarrow^\star \dot v\end{array}}{(\Sigma,\mathcal{U}) \vdash \langle l, \dot\rho\rangle \Downarrow^\star \dot v}$$

$$\textbf{(r 6.8)}\quad \frac{\begin{array}{l}\Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2}) \ \wedge\ \mathcal{U}(l)\\ (\Sigma,\mathcal{U}) \vdash \langle l_0, \dot\rho\rangle \Downarrow^\star true\\ (\Sigma,\mathcal{U}) \vdash \langle l_1, \dot\rho\rangle \Downarrow^\star \dot v\end{array}}{(\Sigma,\mathcal{U}) \vdash \langle l, \dot\rho\rangle \Downarrow^\star \dot v}$$

$$\textbf{(r 6.9)}\quad \frac{\begin{array}{l}\Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2}) \ \wedge\ \mathcal{U}(l)\\ (\Sigma,\mathcal{U}) \vdash \langle l_0, \dot\rho\rangle \Downarrow^\star false\\ (\Sigma,\mathcal{U}) \vdash \langle l_2, \dot\rho\rangle \Downarrow^\star \dot v\end{array}}{(\Sigma,\mathcal{U}) \vdash \langle l, \dot\rho\rangle \Downarrow^\star \dot v}$$

$$\textbf{(r 6.10)}\quad \frac{\begin{array}{l}\Sigma(l) = (\texttt{let}\ x = t_1^{l_1}\ \texttt{in}\ t_2^{l_2}) \ \wedge\ \mathcal{U}(l)\\ (\Sigma,\mathcal{U}) \vdash \langle l_1, \dot\rho\rangle \Downarrow^\star \dot v_1\\ (\Sigma,\mathcal{U}) \vdash \langle l_2, \dot\rho[x \mapsto v_1]\rangle \Downarrow^\star \dot v\end{array}}{(\Sigma,\mathcal{U}) \vdash \langle l, \dot\rho\rangle \Downarrow^\star \dot v}$$

Figure 4.2: Slice Operational Semantics

unchanged". If v is a constant then the definition is straightforward. We say that $\mathcal{U}$ is a slice of configuration $\mathcal{C}$ under universe Σ if $\exists c$ such that:

$$\Sigma \vdash \langle l,\rho\rangle \Downarrow c \ \wedge \ (\Sigma,\mathcal{U}) \vdash \langle l,\rho\rangle \Downarrow^{\star} c$$

that is, if the configuration evaluates to the same constant c under the standard and slice semantics[4]. Slices, according to this definition, are only defined for terminating programs that evaluate to constants. We extend the definition to configurations that evaluate to any kind of value in the next section.

4.3.2 Standard Slice

The definition of relevant slices considers only configurations that evaluate to constants. Even if we are only interested in these type of configurations, at some point we will have to reason about the intermediate configurations of these evaluations, which may include closures. In this section, we extend the definition of slices to any kind of terminating configuration. In order to do this, we study first the relationship between the evaluations of $\Sigma \vdash \mathcal{C} \Downarrow v$ and $(\Sigma,\mathcal{U}) \vdash \mathcal{C} \Downarrow^{\star} \dot{v}$.

We shall start by assuming $\Sigma \vdash \mathcal{C} \Downarrow v$ and $(\Sigma,\mathcal{U}) \vdash \mathcal{C} \Downarrow^{\star} \dot{v}$. If we inspect both derivation trees and try to match configurations and values, we will see that, for each node $(\Sigma,\mathcal{U}) \vdash \langle l,\dot{\rho}_0\rangle \Downarrow^{\star} \dot{v}_0$ in $(\Sigma,\mathcal{U}) \vdash \mathcal{C} \Downarrow^{\star} \dot{v}$ there exists a matching node $\Sigma \vdash \langle l,\rho_0\rangle \Downarrow v_0$ in $\Sigma \vdash \mathcal{C} \Downarrow v$, where $\dot{v}_0$ and v_0 are related by $\mathcal{V}_{\star}$. The relation $(\dot{v} \ \mathcal{V}_{\star} \ v)$ is defined as follows:

$$\begin{array}{rcl} \star \ \mathcal{V}_{\star} \ v & \Leftrightarrow & \mathit{true} \\ c \ \mathcal{V}_{\star} \ c' & \Leftrightarrow & c = c' \\ [l,\, \dot{\rho}] \ \mathcal{V}_{\star} \ [l',\, \rho] & \Leftrightarrow & (l = l') \ \wedge \ \dot{\rho} \ \mathcal{R}_{\star} \ \rho \\ \dot{\rho} \ \mathcal{R}_{\star} \ \rho & \Leftrightarrow & \forall x \in \mathit{dom}(\dot{\rho}).\ \dot{\rho}(x) \ \mathcal{V}_{\star} \ \rho(x) \end{array} \tag{4.1}$$

In other words, $(\dot{v}_0 \ \mathcal{V}_{\star} \ v_0)$ is true if $\dot{v}_0$ has the structure as v_0 except for some constants that have been replaced by $\star$. The same can be said about configurations and nodes, which

[4] In [Bis97b] this is called a *relevant slice*

are related by $\mathcal{R}_\star$:

$$\begin{aligned} \langle l, \dot{\rho}\rangle \;\mathcal{R}_\star\; \langle l', \rho\rangle \quad &\Leftrightarrow \quad (l = l') \;\wedge\; \dot{\rho} \;\mathcal{R}_\star\; \rho \\ (\Sigma, \mathcal{U}) \vdash \mathcal{C} \Downarrow^\star \dot{v} \;\mathcal{R}_\star\; \Sigma' \vdash \mathcal{C}' \Downarrow v \quad &\Leftrightarrow \quad (\Sigma' = \Sigma) \;\wedge\; (\mathcal{C} \;\mathcal{R}_\star\; \mathcal{C}') \;\wedge\; (\dot{v} \;\mathcal{V}_\star\; v) \end{aligned} \tag{4.2}$$

Let $\preceq_\star$ be the simulation induced by $\mathcal{R}_\star$. If $\Sigma \vdash \mathcal{C} \Downarrow v$ and there is $\mathcal{U}$ such that $(\Sigma, \mathcal{U}) \vdash \mathcal{C} \Downarrow^\star \dot{v}$, then the derivation trees $[(\Sigma, \mathcal{U}) \vdash \mathcal{C} \Downarrow^\star \dot{v}]$ and $[\Sigma \vdash \mathcal{C} \Downarrow v]$ form a $\mathcal{R}_\star$-simulation:

Theorem 4.3.2

For all l, ρ, $\dot{\rho}$, $\mathcal{U}$: if $\Sigma \vdash \langle l, \rho\rangle \Downarrow v$, $(\Sigma, \mathcal{U}) \vdash \langle l, \dot{\rho}\rangle \Downarrow^\star \dot{v}$ and $(\dot{\rho} \;\mathcal{R}_\star\; \rho)$ then

$$[(\Sigma, \mathcal{U}) \vdash \langle l, \dot{\rho}\rangle \Downarrow^\star \dot{v}] \;\preceq_\star\; [\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$$

Proof: *By rule induction on* $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$.

In other words, if a configuration succeeds in terminating under the standard and slice semantics - for some $\mathcal{U}$ - then the derivation tree of $(\Sigma, \mathcal{U}) \vdash \langle l, \dot{\rho}\rangle \Downarrow^\star \dot{v}$ is the same as the derivation tree of $\Sigma \vdash \langle l, \rho\rangle \Downarrow v$ excluding constants and subtrees that have been replaced by $\star$. In particular, the resulting values are related by $\mathcal{V}_\star$:

Corollary 4.3.3 *If $\Sigma \vdash \langle l, \rho\rangle \Downarrow v$ and $(\Sigma, \mathcal{U}) \vdash \langle l, \rho\rangle \Downarrow^\star \dot{v}$ then $(\dot{v} \;\mathcal{V}_\star\; v)$.*

Proof: *Let $\mathcal{M} = \Sigma \vdash \langle l, \rho\rangle \Downarrow v$ and $\mathcal{N} = (\Sigma, \mathcal{U}) \vdash \langle l, \rho\rangle \Downarrow^\star \dot{v}$. Clearly $(\rho \;\mathcal{R}_\star\; \rho)$ satisfies the conditions in (4.1). By applying theorem 4.3.2 to $\mathcal{N}$ and $\mathcal{M}$ we get $[\mathcal{N}] \;\preceq_\star\; [\mathcal{M}]$. Therefore, $\mathcal{N} \;\mathcal{R}_\star\; \mathcal{M}$ and, by applying (4.2), $(\dot{v} \;\mathcal{V}_\star\; v)$.*

The definitions and theorems presented above show the relations between the standard and slice semantics. In particular they say that if a configuration succeeds in terminating under the standard and slice semantics, unless the second evaluation returns $\star$, both configurations evaluate to the same constant or the same type of closure. This means that in order to define what we mean by a group of expressions "not affecting the final value of a program" we only need to check that the resulting value is different from $\star$. The general notion of slice - the *standard slice* - is defined by:

Definition 4.3.4 (Standard Slice)

$$Slice(\Sigma,\mathcal{U},\langle l,\rho\rangle) \;\stackrel{def}{\equiv}\; \exists v,\dot{v}:\;\; \Sigma \vdash \langle l,\rho\rangle \Downarrow v \;\wedge\; (\Sigma,\mathcal{U}) \vdash \langle l,\rho\rangle \Downarrow^{\star} \dot{v} \;\wedge\; (\dot{v} \neq \star)$$

$\mathcal{U}$ is a standard slice of $\mathcal{C}$ if the slice evaluation of $\mathcal{C}$ under $\mathcal{U}$ returns a value different from $\star$. By applying theorem 4.3.2 to the definition we know that ($\dot{v}\;\;\mathcal{V}_{\star}\;\;v$) must hold. Therefore, both evaluations return the same constant or the same closure, with some of the values replaced by $\star$. The presence of the existential quantifier in the definition is due to the fact that this kind of slice is only defined for terminating configurations. This definition is sound as long as the semantics are deterministic.

4.3.3 Limitations and Similarities

This definition of a program slice only contemplates terminating programs. In the case of programs that do not terminate, a slice would be defined as the set of program labels that when plugged to the slice semantics preserve non-termination. We address this issue later.

Another technique used to define a slice is to compare the evaluation of an expression and its sliced version. A sliced expression is obtained by replacing all the subexpressions outside the slice - $\mathcal{L}(l) - \mathcal{U}$ - with the new program constant $\star$ (see $\mathcal{T}^{\star}_{\mathcal{U}}$ in section 4.2). The sliced version will no longer be part of the language and, consequently, the standard semantics has to be modified to handle the new cases.

This is the approach followed by Biswas in [Bis97b]. The extended semantics handles the new constant expression as a `no-op` and errors caused by its presence are propagated in a lazy fashion. The rest of the semantics is strict. Some of the slices accepted by this definition are rejected by ours. For example, given the expressions:

$$\Sigma(l) = ((\texttt{fn}\ x\ \texttt{=>}\ 5)^{l_0}\ t_1^{l_1})$$
$$\Sigma(l_1) = (\texttt{if}\ t_2^{l_2}\ \texttt{then}\ t_3^{l_3}\ \texttt{else}\ t_4^{l_4})$$

if t_2 is replaced by $\star$ - $(\texttt{if}\ \star^{l_2}\ \texttt{then}\ t_3^{l_3}\ \texttt{else}\ t_4^{l_4})^{l_1}$ - then the new semantics evaluates $\langle l_1, \rho\rangle$ to $\star$. This is ignored by the evaluation of $(\texttt{fn}\ x\ \texttt{=>}\ 5)^{l_0}$ and, therefore, $\langle l, \rho\rangle$ evaluates to 5. Consequently, the set $\mathcal{L}(l) - \{l_2\}$ is considered as a valid slice by the alternative definition. Clearly $\mathcal{U} = \mathcal{L}(l) - \{l_2\}$ is not a valid slice of $(\Sigma, \langle l, \rho\rangle)$ under our definition, *i.e.*, $\nexists \dot{v}.\ (\Sigma, \mathcal{U}) \vdash \langle l, \rho\rangle \Downarrow^\star \dot{v}$. Our definition of slice is more restrictive but this does not represent a problem since the slices rejected by our definition do not define a meaning preserving transformation.

The semantics of our definition of slice is targeted to address the correctness issues of the transformation in section 4.1. The relation between them is shown in the next section.

4.4 Slices and the Transformation

The transformation and the slice semantics are closely related. We show that the evaluation of a configuration under the slice semantics succeeds if and only if the evaluation of the transformed configuration succeeds too. Furthermore, the derivation trees of $(\Sigma, \mathcal{U}) \vdash \mathcal{C} \Downarrow^\star \dot{v}$ and $\mathcal{T}_{\mathcal{U}}^d(\Sigma) \vdash \mathcal{C} \Downarrow v$ form a bisimulation where each instance of $\star$ in the first tree is paired with d in the second tree. We make this precise by first formalizing the relationship between matching values of both derivation trees. $\dot{v}\ \ \mathcal{V}_\star^d\ \ v$ is defined by:

$$\begin{aligned} \star\ \mathcal{V}_\star^d\ d \quad &\Leftrightarrow \quad \mathit{true} \\ c\ \mathcal{V}_\star^d\ c' \quad &\Leftrightarrow \quad c = c' \\ [l,\, \dot{\rho}]\ \mathcal{V}_\star^d\ [l',\, \rho] \quad &\Leftrightarrow \quad (l' = l)\ \wedge\ \dot{\rho}\ \mathcal{R}_\star^d\ \rho \end{aligned} \tag{4.3}$$

We extend the relation to include configurations and nodes. $\mathcal{R}_\star^d$ is defined by:

$$\begin{aligned} \langle l, \dot{\rho}\rangle\ \mathcal{R}_\star^d\ \langle l', \rho\rangle \quad &\Leftrightarrow \quad (l' = l)\ \wedge\ \dot{\rho}\ \mathcal{R}_\star^d\ \rho \\ ((\Sigma, \mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^\star \dot{v})\ \mathcal{R}_\star^d\ (\Sigma' \vdash \mathcal{C} \Downarrow v) \quad &\Leftrightarrow \quad \begin{cases} (\Sigma' = \mathcal{T}_{\mathcal{U}}^d(\Sigma))\ \wedge\ (\dot{\mathcal{C}}\ \mathcal{R}_\star^d\ \mathcal{C}) \\ (\dot{v}\ \mathcal{V}_\star^d\ v) \end{cases} \end{aligned} \tag{4.4}$$

Let's define $\cong_\star^d$ as the bisimulation induced by $\mathcal{R}_\star^d$. We prove the following:

Theorem 4.4.1

$\forall \Sigma, \dot{\mathcal{C}}, \mathcal{C}$: if $(\dot{\mathcal{C}} \ \mathcal{R}^d_\star \ \mathcal{C})$ then the following is true:

- *If $\exists \dot{v}.\ [(\Sigma, \mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^\star \dot{v}]$ then $\exists v.\ [(\Sigma, \mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^\star \dot{v}] \ \cong^d_\star \ [\mathcal{T}^d_\mathcal{U}(\Sigma) \vdash \mathcal{C} \Downarrow v]$*
- *If $\exists v.\ [\mathcal{T}^d_\mathcal{U}(\Sigma) \vdash \mathcal{C} \Downarrow v]$ then $\exists \dot{v}.\ [(\Sigma, \mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^\star \dot{v}] \ \cong^d_\star \ [\mathcal{T}^d_\mathcal{U}(\Sigma) \vdash \mathcal{C} \Downarrow v]$*

Proof: *By induction on the derivation trees of $[(\Sigma, \mathcal{U}) \vdash \mathcal{C}^\star \Downarrow^\star \dot{v}]$ and $[\mathcal{T}^\star_\mathcal{U}(\Sigma) \vdash \mathcal{C} \Downarrow v]$.*

The previous theorem tells us that derivation trees of $(\Sigma, \mathcal{U}) \vdash \mathcal{C}^\star \Downarrow^\star v^\star$ and $\mathcal{T}^d_\mathcal{U}(\Sigma) \vdash \mathcal{C} \Downarrow c$ have the same "shape" and, furthermore, agree on all values except the cases where $\star$ is matched with the chosen constant d. We are ready to establish the relation with slices:

Theorem 4.4.2 (A Slice defines a meaning preserving transformation)

If $\Sigma \vdash \mathcal{C} \Downarrow c$ and $Slice(\Sigma, \mathcal{U}, \mathcal{C})$ then $\mathcal{T}^d_\mathcal{U}(\Sigma) \vdash \mathcal{C} \Downarrow c$.

Proof: *If $\Sigma \vdash \mathcal{C} \Downarrow c$ then, by applying the definition of Slice, we get $(\Sigma, \mathcal{U}) \vdash \mathcal{C} \Downarrow^\star \dot{v}$ and $\dot{v} \neq \star$. We also know, from theorem 4.3.2, that $\dot{v} \ \mathcal{V}_\star \ v$. Therefore, by applying the definition of $\mathcal{V}_\star$, we derive $\dot{v} = c$*

We have $(\Sigma, \mathcal{U}) \vdash \mathcal{C} \Downarrow^\star c$. Then, by applying theorem 4.4.1 to the previous result, we get:

$$\exists v.\ [(\Sigma, \mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^\star c] \ \cong^d_\star \ [\mathcal{T}^d_\mathcal{U}(\Sigma) \vdash \mathcal{C} \Downarrow v]$$

In particular, we know that $(c \ \mathcal{V}^d_\star \ v)$. Therefore, by applying the definition of $\mathcal{V}^d_\star$ (see 4.3), we derive $v = c$, i.e., $\mathcal{T}^d_\mathcal{U}(\Sigma) \vdash \mathcal{C} \Downarrow c$.

[**QED**]

This means that in order to get a meaning preserving transformation of $\Sigma \vdash \mathcal{C} \Downarrow c$ we have to find a valid slice of $(\Sigma, \mathcal{C})$, *i.e.*, $\mathcal{U}$ such that $(\Sigma, \mathcal{U}) \vdash \mathcal{C} \Downarrow^\star \dot{v}$ and $\dot{v} = c$ or, in general, $\mathcal{U}$ such that $\dot{v} \neq \star$. The next section gives clues on how to obtain such $\mathcal{U}$.

4.5 Properties of Slices

A simple inspection of the evaluation rules of the language and its slice semantics can help us deduce invariants of program slices. These invariants, expressed as a set of constraints, will be written as a proof system that when abstracted turns into a specification for a set of program analyses that compute valid program slices.

We will deduce these invariants by inspecting the evaluations used to define a program slice. We want to know what conditions are necessary, given a terminating evaluation $\Sigma \vdash \langle l, \rho \rangle \Downarrow v$, in order for $\mathcal{U}$ to be a valid slice. According to Definition 4.3.4, the following must be true.

$$\Sigma \vdash \langle l, \rho \rangle \Downarrow v \;\Rightarrow\; (\Sigma, \mathcal{U}) \vdash \langle l, \rho \rangle \Downarrow^{\star} \dot{v} \quad \wedge \quad \dot{v} \neq \star \tag{4.5}$$

where $\dot{v}\ \ \mathcal{V}_{\star}\ \ v$.

Our goal is to set up a proof system based on the evaluation of $\Sigma \vdash \langle l, \rho \rangle \Downarrow v$ that can be used to validate program slices. The judgement $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \models \mathcal{U}$ claims that $\mathcal{U}$ is a property of the derivation tree $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v]$, *i.e.*, a property of the execution of the program. This property must be defined such that:

$$[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \models \mathcal{U} \;\Rightarrow\; \mathit{Slice}(\Sigma, \mathcal{U}, \langle l, \rho \rangle)$$

where $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \models \mathcal{U}$ is defined by a set of constraints placed upon subtrees reachable from $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v]$: a constraint system built on the execution trace.

We list a series of conditions necessary for (4.5) to be true. The first condition is obvious but critical. It says that the expression inside the configuration is always part of its slice. The following is trivially true:

- If $(\Sigma, \mathcal{U}) \vdash \langle l, \rho \rangle \Downarrow^{\star} \dot{v} \ \wedge \ \dot{v} \neq \star$ then $\mathcal{U}(l)$ must be true.

This condition is derived from the fact that if $\neg\mathcal{U}(l)$ then $(\Sigma, \mathcal{U}) \vdash \langle l, \rho \rangle \Downarrow^{\star} \star$ always and it is essential because it triggers the other conditions, which are specific to each evaluation rule of the semantics of the program.

Given $\Sigma(l) = t$ and $\mathcal{U}(l)$ the following must be true:

- If t is a primitive application then the operator and all the actual parameters must belong to the slice.
- If t is an `if`-term then the test must belong to the slice. If the conditional expression evaluates to *true* then the true-branch expression must belong to the slice too. The case where the conditional-expression evaluates to *false* is analogous.

Applications and `let`-expressions present special difficulties. Let $t = (\texttt{let}\ x = e_1\ \texttt{in}\ e_2)$. It is clear that its body e_2 is needed and must belong to the slice. What about e_1? The expression e_1 needs to be included in the slice only if the evaluation of the body of the `let`-expression depends on the value of `let`-variable x. We should have the following:

$$\begin{array}{lcl} \Sigma(l) = (\texttt{let}\ x = t_1^{l_1}\ \texttt{in}\ t_2^{l_2}) & & \mathcal{U}(l_2) \\ (\Sigma,\mathcal{U}) \vdash \langle l_1, \rho\rangle \Downarrow^\star v_1 & \Rightarrow & \\ (\Sigma,\mathcal{U}) \vdash \langle l_2, \rho[x \mapsto v_1]\rangle \Downarrow^\star v & & \mathbf{depends}(x, \langle l_2, \rho[x \mapsto v_1]\rangle) \Rightarrow \mathcal{U}(l_1) \end{array} \tag{4.6}$$

Hence, we must reason about variables and its relation with the elements of the program slice. The formula $\mathbf{depends}(x, \langle l, \rho\rangle)$ is used to express the fact that the evaluation of $\langle l, \rho\rangle$ depends on the value of x. We formalize this dependency/need relation in the next section.

Something similar occurs with applications. If $t = (e_0\ e_1 \ldots e_n)$ then it is clear that the expression in the operator position - e_0 - is needed and must belong to the slice. The case of the operands is different. The operand e_i must be included in the slice only if the evaluation of the body of the closure that flows to e_0 depends on its i-th formal parameter.

We have the following:

$$
\begin{array}{l}
\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n}) \\
\Sigma \vdash \langle l_0, \rho \rangle \Downarrow [l_f,\ \rho_0] \\
\Sigma \vdash \langle l_i, \rho \rangle \Downarrow v_i \quad i = 1, \ldots, n \\
\Sigma \vdash \langle l_b, \rho' \rangle \Downarrow v
\end{array}
\Rightarrow
\begin{array}{l}
\mathcal{U}(l_0) \\
\mathcal{U}(l_b) \\
\mathbf{depends}(x_i, \langle l_b, \rho' \rangle) \Rightarrow \mathcal{U}(l_i)
\end{array}
\tag{4.7}
$$

where $\Sigma(l_f) = (\texttt{fn}\ x_1 \ldots x_n \Rightarrow t_b^{l_b})$ and $\rho' = \rho_0[x_1 \mapsto v_1] \ldots [x_n \mapsto v_n]$.

4.5.1 Slices and Variables

A variable that doesn't contribute to the final value of a computation is considered *useless*. Similarly, a variable that does contribute to the final value of a computation is considered as *useful*[5]. We link the concepts of useful variables and program slices by making the following assertion: A useful variable, in a given slice, should always bind to a value produced by the slice of the program. On the other hand, values generated inside the slice may bind to a variable that is not useful but, if the values generated by an argument expression always bind to useful variables, then the expression can safely be considered outside the slice.

We have already started using the concept of useful variables. In 4.6 we used the relation $\mathbf{depends}(x, \langle l_2, \rho[x \mapsto v_1] \rangle)$ to express the need of x in the evaluation of $\langle l_2, \rho[x \mapsto v_1] \rangle)$. In other words, we want to know if x is *useful* in the evaluation of $\langle l_2, \rho[x \mapsto v_1] \rangle)$.

We model the **depends** relation by introducing a mapping $\mathcal{D}$ from configurations and closures to sets of variables:

$$\mathcal{D} \in (\mathit{Config} \cup \mathit{Closure}) \rightarrow \mathcal{P}(\mathit{Var})$$

where $\mathcal{D}(\langle l, \rho \rangle)$ and $\mathcal{D}([l,\ \rho])$ range over the set of free variables of the expression mapped by l.

[5] We can actually think of this as a version of slice for variables.

The dependency set $\mathcal{D}(\mathcal{C})$ keeps track of the variables that *may* be useful during the evaluation of $\mathcal{C}$, *i.e.*, the set of variables that the evaluation of $\mathcal{C}$ may depend on. The dependency set $\mathcal{D}([l,\ \rho])$ keeps track of the variables that *may* be used by any of the applications of the closure during the evaluation of the program.

Dependency sets may change for different applications of the same function. Let's consider the code fragment:

```
let f = fn(x,y,z) = if (x < 0) then y else z
in
  .... (f -1 a b)
  .....(f 5 c d)
```

The two function applications determine different dependency sets and suggest different transformations. In the first application, the third argument can be replaced while in the second application it is the second argument that can be optimized. If we mix both dependency sets into one then all the arguments of `f` will be marked as part of the slice. Therefore, dependency sets must be kept separately, one per binding configuration.

The pair $(\mathcal{U}, \mathcal{D})$ is checked against a given execution trace by validating each reachable subtree $[\Sigma \vdash \langle l', \rho' \rangle \Downarrow v']$ against the local constraints $[\Sigma \vdash \langle l', \rho' \rangle \Downarrow v'] \models_{sl}^{loc} (\mathcal{U}, \mathcal{D})$ defined in Figure 4.3 . These constraints reflect the conditions outlined at the beginning of the section. For example, compare (4.6) with the rule for `let`-expressions:

$$\begin{array}{l} \Sigma(l) = (\texttt{let}\ x = t_1^{l_1}\ \texttt{in}\ t_2^{l_2})\ \wedge\ \mathcal{U}(l) \\ [\Sigma \vdash \langle l_2, \rho' \rangle \Downarrow v_2] \in \mathit{reach}([\Sigma \vdash \langle l, \rho \rangle \Downarrow v]) \end{array} \Rightarrow \begin{cases} \mathcal{U}(l_2)\ \wedge\ \mathcal{D}(\langle l_2, \rho' \rangle) - \{x\} \subseteq \mathcal{D}(\langle l, \rho \rangle) \\ x \in \mathcal{D}(\langle l_2, \rho' \rangle) \Rightarrow \mathcal{U}(l_1) \\ \mathcal{U}(l_1) \Rightarrow \mathcal{D}(\langle l_1, \rho \rangle) \subseteq \mathcal{D}(\langle l, \rho \rangle) \end{cases}$$

where l_1 must be part of the slice if the evaluation of $\langle l_2, \rho' \rangle$ depends on x.

We only use the information of the dependency sets of binding configurations, *i.e.*, evaluation of `let` and function bodies. For example, in Rule (**r 7.5**), the constraint $x_i \in$

$\mathcal{D}(\langle l_b, \rho' \rangle) \Rightarrow \mathcal{U}(l_i)$ is used to make sure that all arguments whose evaluated values bind to a useful variable must be part of the slice. This dependency information is passed to the reachable subtrees inside the slice - in the example above we write $\mathcal{D}(\langle l_2, \rho' \rangle) = \mathcal{D}(\langle l, \rho \rangle)$ - where each subcomputation checks for the validity of the set. Eventually the validation process hits a variable evaluation where the following is checked:

$$\Sigma(l) = x \ \wedge \ \mathcal{U}(l) \ \Rightarrow \ x \in \mathcal{D}(\langle l, \rho \rangle)$$

Hence, a variable instance is useful[6] if it is accessed during the evaluation of the slice.

We only need to check for subtrees inside the slice defined by $\mathcal{U}$. In order to do this we define $\mathit{reach}^{\mathcal{U}}([\Sigma \vdash \mathcal{C} \Downarrow v])$ as the set of subtrees reachable only via nodes $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v']$.

Definition 4.5.1 (Reachability via useful expressions)
$\mathit{reach}^{\mathcal{U}}$ is defined as the reflexive, transitive closure of $\mathit{reach}\mathcal{U}$ where

$$\mathit{reach}\mathcal{U}([\Sigma \vdash \langle l, \rho \rangle \Downarrow v]) = \begin{cases} \mathit{reach}([\Sigma \vdash \langle l, \rho \rangle \Downarrow v]) & \mathit{if}\, \mathcal{U}(l) \\ \{\,\} & \mathit{otherwise} \end{cases}$$

Note that $\mathit{reach}^{\mathcal{U}}([\Sigma \vdash \langle l, \rho \rangle \Downarrow v])$ will also consider some nodes $(\Sigma \vdash \langle l', \rho' \rangle \Downarrow v')$ where $\neg\mathcal{U}(l')$. These are precisely the first nodes in $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v]$, starting bottom-up from the root, that do not satisfy the $\mathcal{U}$ condition, that is, the first nodes outside the slice. The root is always in $\mathit{reach}^{\mathcal{U}}$.

Lemma 4.5.2 *If $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \models_{sl} (\mathcal{U}, \mathcal{D})$ and $\mathcal{U}(l)$ then*

$$\forall\, [\Sigma \vdash \langle l', \rho' \rangle \Downarrow v'] \in \mathit{reach}([\Sigma \vdash \langle l, \rho \rangle \Downarrow v]).\ \ [\Sigma \vdash \langle l', \rho' \rangle \Downarrow v'] \models_{sl} (\mathcal{U}, \mathcal{D})$$

Now we can state the slice specification:

[6] The notion of useful variable is linked to a particular slice.

Definition 4.5.3 (Slice Specification with useful variables)
$[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \models_{sl} (\mathcal{U}, \mathcal{D}) \;\Leftrightarrow$

$$\forall [\Sigma \vdash \langle l', \rho' \rangle \Downarrow v'] \in reach^{\mathcal{U}}([\Sigma \vdash \langle l, \rho \rangle \Downarrow v]).\ [\Sigma \vdash \langle l', \rho' \rangle \Downarrow v'] \models_{sl}^{loc} (\mathcal{U}, \mathcal{D})$$

where $[\Sigma \vdash \langle l', \rho' \rangle \Downarrow v'] \models_{sl}^{loc} (\mathcal{U}, \mathcal{D})$ *is defined by the rules shown in Figure 4.3.*

Examples

The examples are based on the following code and evaluation:

$$\begin{array}{rl} e_0 = (\texttt{let} & f_1 = (\texttt{fn}\ x, y \Rightarrow x^{31})^1 \\ & f_2 = (\texttt{fn}\ a, b \Rightarrow a^{32})^2 \\ & f_3 = (\texttt{fn}\ c, d \Rightarrow (*\ d\ \ d)^{33})^3 \\ \texttt{in} & e_4)^0 \end{array} \quad \text{and} \quad \begin{array}{l} \Sigma \vdash \langle 0, \rho_0 \rangle \Downarrow v \ \wedge\ \rho_0 = [\,] \\ \Sigma \vdash \langle 4, \rho_4 \rangle \Downarrow v \\ \rho_4 = \rho_0[f_i \mapsto [i,\ \rho_0]]_{i=1,2,3} \end{array} \tag{4.8}$$

where `let` has been extended to handle multiple variables. Terms are labelled by numbers but, for clarity, not all terms are labelled. We will also assume that an expression e_n is always labelled by n.

We start by setting $\mathcal{U}(0)$. According to rule (**r 7.9**) defined for evaluation of `let`-expressions, the following must be true:

$$\mathcal{U}(4) \quad \wedge \quad i \in \{1, 2, 3\} : f_i \in \mathcal{D}(\langle 4, \rho_4 \rangle) \Rightarrow \mathcal{U}(i) \tag{4.9}$$

Example 1: Let's define e_4 as:

$$\begin{array}{rll} e_4 = (\texttt{let} & g = (\texttt{if}\ e_6\ \texttt{then}\ f_1^7\ \texttt{else}\ f_2^8)^5 & \Sigma \vdash \langle 6, \rho_4 \rangle \Downarrow true \\ & h = (\texttt{if}\ e_{10}\ \texttt{then}\ {f_1}^{11}\ \texttt{else}\ {f_3}^{12})^9 & \Sigma \vdash \langle 10, \rho_4 \rangle \Downarrow false \\ \texttt{in} & e_{13})^4 & \Sigma \vdash \langle 13, \rho_{13} \rangle \Downarrow v \end{array} \tag{4.10}$$

where $\rho_{13} = \rho_4[g \mapsto [1,\ \rho_0]][h \mapsto [3,\ \rho_0]]$ and, e_6 and e_{10} are closed expressions.

$$[\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \models_{sl}^{loc} (\mathcal{U}, \mathcal{D}) \quad \Leftrightarrow \quad \mathcal{U}(l) \Rightarrow$$

Let $Reach = reach([\Sigma \vdash \langle l, \rho\rangle \Downarrow v])$:

(r 7.1) $\Sigma(l) = c \;\Rightarrow\; true$

(r 7.2) $\Sigma(l) \in Fun \;\Rightarrow\; \mathcal{D}([l,\ \rho]) \subseteq \mathcal{D}(\langle l, \rho\rangle)$

(r 7.3) $\Sigma(l) = x \;\Rightarrow\; x \in \mathcal{D}(\langle l, \rho\rangle)$

(r 7.4) $\Sigma(l) = (p\ t_1^{l_1} \dots t_n^{l_n}) \;\Rightarrow\; \forall i \in \{1, \dots, n\} : \begin{cases} \mathcal{U}(l_i) \\ \mathcal{D}(\langle l_i, \rho\rangle) \subseteq \mathcal{D}(\langle l, \rho\rangle) \end{cases}$

(r 7.5) $\begin{array}{l} \Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \dots t_n^{l_n}) \\ [\Sigma \vdash \langle l_0, \rho\rangle \Downarrow [l_f,\ \rho_0]] \in Reach \\ \Sigma(l_f) = (\texttt{fn}\ x_1 \dots x_n \Rightarrow t_b^{l_b}) \\ [\Sigma \vdash \langle l_b, \rho'\rangle \Downarrow v] \in Reach \end{array} \Rightarrow \begin{cases} \mathcal{U}(l_0) \ \wedge\ \mathcal{U}(l_b) \\ \mathcal{D}(\langle l_0, \rho\rangle) \subseteq \mathcal{D}(\langle l, \rho\rangle) \\ \mathcal{D}(\langle l_b, \rho'\rangle) - \{x_1, \dots, x_n\} \subseteq \mathcal{D}([l_f,\ \rho_0]) \\ \forall i \in \{1, \dots, n\} : \\ \quad \begin{cases} x_i \in \mathcal{D}(\langle l_b, \rho'\rangle) \Rightarrow \mathcal{U}(l_i) \\ \mathcal{U}(l_i) \Rightarrow \mathcal{D}(\langle l_i, \rho\rangle) \subseteq \mathcal{D}(\langle l, \rho\rangle) \end{cases} \end{cases}$

(r 7.6) $\begin{array}{l} \Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \dots t_n^{l_n}) \\ [\Sigma \vdash \langle l_0, \rho\rangle \Downarrow [l_f,\ \rho_0]] \in Reach \\ \Sigma(l_f) = (\texttt{fun}\ y\ x_1 \dots x_n \Rightarrow t_b^{l_b}) \\ [\Sigma \vdash \langle l_b, \rho'\rangle \Downarrow v] \in Reach \end{array} \Rightarrow \begin{cases} \mathcal{U}(l_0) \ \wedge\ \mathcal{U}(l_b) \\ \mathcal{D}(\langle l_0, \rho\rangle) \subseteq \mathcal{D}(\langle l, \rho\rangle) \\ \mathcal{D}(\langle l_b, \rho'\rangle) - \{y, x_1, \dots, x_n\} \subseteq \mathcal{D}([l_f,\ \rho_0]) \\ \forall i \in \{1, \dots, n\} : \\ \quad \begin{cases} x_i \in \mathcal{D}(\langle l_b, \rho'\rangle) \Rightarrow \mathcal{U}(l_i) \\ \mathcal{U}(l_i) \Rightarrow \mathcal{D}(\langle l_i, \rho\rangle) = \mathcal{D}(\langle l, \rho\rangle) \end{cases} \end{cases}$

(r 7.7) $\begin{array}{l} \Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2}) \\ [\Sigma \vdash \langle l_2, \rho\rangle \Downarrow true] \in Reach \end{array} \Rightarrow \begin{cases} \mathcal{U}(l_0) \ \wedge\ \mathcal{U}(l_1) \\ \mathcal{D}(\langle l_i, \rho\rangle) \subseteq \mathcal{D}(\langle l, \rho\rangle) \quad i = 0, 1 \end{cases}$

(r 7.8) $\begin{array}{l} \Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2}) \\ [\Sigma \vdash \langle l_2, \rho\rangle \Downarrow false] \in Reach \end{array} \Rightarrow \begin{cases} \mathcal{U}(l_0) \ \wedge\ \mathcal{U}(l_2) \\ \mathcal{D}(\langle l_i, \rho\rangle) \subseteq \mathcal{D}(\langle l, \rho\rangle) \quad i = 0, 2 \end{cases}$

(r 7.9) $\begin{array}{l} \Sigma(l) = (\texttt{let}\ x = t_1^{l_1}\ \texttt{in}\ t_2^{l_2}) \\ [\langle l_2, \rho'\rangle v_2] \in Reach \end{array} \Rightarrow \begin{cases} \mathcal{U}(l_2) \\ \mathcal{D}(\langle l_2, \rho'\rangle) - \{x\} \subseteq \mathcal{D}(\langle l, \rho\rangle) \\ x \in \mathcal{D}(\langle l_2, \rho'\rangle) \Rightarrow \mathcal{U}(l_1) \\ \mathcal{U}(l_1) \Rightarrow \mathcal{D}(\langle l_1, \rho\rangle) \subseteq \mathcal{D}(\langle l, \rho\rangle) \end{cases}$

Figure 4.3: Slice Specification with useful variables

By applying rule (**r 7.9**) to $\mathcal{U}(4)$ and the evaluation of $\langle 4, \rho_4 \rangle$ we derive the following conditions:

$$\mathcal{U}(13) \quad \wedge \quad \begin{cases} g \in \mathcal{D}(\langle 13, \rho_{13} \rangle) \Rightarrow \mathcal{U}(5) \Rightarrow \mathcal{D}(\langle 5, \rho_4 \rangle) \subseteq \mathcal{D}(\langle 4, \rho_4 \rangle) \\ h \in \mathcal{D}(\langle 13, \rho_{13} \rangle) \Rightarrow \mathcal{U}(9) \Rightarrow \mathcal{D}(\langle 9, \rho_4 \rangle) \subseteq \mathcal{D}(\langle 4, \rho_4 \rangle) \end{cases} \tag{4.11}$$

Let's consider

$$e_{13} = (+ \ (g^{15} \ e_{16} \ \ e_{17})^{14} \ \ (h^{19} \ e_{20} \ \ e_{21})^{18})^{13}$$

and

$$\begin{aligned} &\Sigma \vdash \langle 15, \rho_{13} \rangle \Downarrow [1, \ \rho_0] \ \wedge \ \Sigma \vdash \langle 31, \rho_0' \rangle \Downarrow v_{14} \\ &\Sigma \vdash \langle 19, \rho_{13} \rangle \Downarrow [3, \ \rho_0] \ \wedge \ \Sigma \vdash \langle 33, \rho_0'' \rangle \Downarrow v_{18} \\ &v = v_{14} + v_{18} \end{aligned} \tag{4.12}$$

where e_{16}, e_{17}, e_{20} and e_{21} are closed expressions. Using $\mathcal{U}(13)$ we apply rule (**r 7.4**) to the evaluation of $\langle 13, \rho_{13} \rangle$ and derive

$$\begin{aligned} &\mathcal{U}(14) \quad \wedge \quad \mathcal{D}(\langle 14, \rho_{13} \rangle) \subseteq \mathcal{D}(\langle 13, \rho_{13} \rangle) \\ &\mathcal{U}(18) \quad \wedge \quad \mathcal{D}(\langle 18, \rho_{13} \rangle) \subseteq \mathcal{D}(\langle 13, \rho_{13} \rangle) \end{aligned} \tag{4.13}$$

This triggers the function application rule for expressions 14 and 18. The constraints for expressions 14 are:

$$\begin{aligned} &\mathcal{U}(15) \quad \wedge \quad \mathcal{D}(\langle 15, \rho_{13} \rangle) \subseteq \mathcal{D}(\langle 14, \rho_{13} \rangle) \\ &x \in \mathcal{D}(\langle 31, \rho_0' \rangle) \Rightarrow \mathcal{U}(16) \Rightarrow \mathcal{D}(\langle 16, \rho_{13} \rangle) \subseteq \mathcal{D}(\langle 14, \rho_{13} \rangle) \\ &y \in \mathcal{D}(\langle 31, \rho_0' \rangle) \Rightarrow \mathcal{U}(17) \Rightarrow \mathcal{D}(\langle 17, \rho_{13} \rangle) \subseteq \mathcal{D}(\langle 14, \rho_{13} \rangle) \end{aligned}$$

Clearly $\mathcal{D}(\langle 31, \rho_0' \rangle) = \{x\}$, $\mathcal{D}(\langle 15, \rho_{13} \rangle) = \{g\}$ and $\mathcal{D}(\langle 16, \rho_{13} \rangle) = \{\ \}$ are sound values. Therefore we can derive $\mathcal{U}(16)$ and $\mathcal{D}(\langle 14, \rho_{13} \rangle) = \{g\}$. Similarly $\mathcal{U}(19)$ is true and, by showing $\mathcal{D}(\langle 33, \rho_0'' \rangle) = \{d\}$, we derive $\mathcal{U}(21)$ and $\mathcal{D}(\langle 18, \rho_{13} \rangle) = \{h\}$.

Using (4.13), $\mathcal{D}(\langle 14, \rho_{13} \rangle) = \{g\}$ and, $\mathcal{D}(\langle 18, \rho_{13} \rangle) = \{h\}$ we derive $\mathcal{D}(\langle 13, \rho_{13} \rangle) = \{g, h\}$ which, when applied to (4.11) gives us:

$$\mathcal{U}(5) \ \wedge \ \mathcal{U}(9) \ \wedge \ \mathcal{D}(\langle 5, \rho_4 \rangle) \subseteq \mathcal{D}(\langle 4, \rho_4 \rangle) \ \wedge \ \mathcal{D}(\langle 9, \rho_4 \rangle) \subseteq \mathcal{D}(\langle 4, \rho_4 \rangle) \tag{4.14}$$

A quick inspection of the evaluation of $\langle 5, \rho_4\rangle$ will let us determine $\mathcal{U}(6)$, $\mathcal{U}(7)$ and, assuming e_6 closed, $\mathcal{D}(\langle 5, \rho_4\rangle) = \{f_1\}$. Similarly, by inspecting the evaluation of $\langle 9, \rho_4\rangle$, we derive $\mathcal{U}(10)$, $\mathcal{U}(12)$ and $\mathcal{D}(\langle 9, \rho_4\rangle) = \{f_3\}$.

Therefore, using (4.14), we get $\mathcal{D}(\langle 4, \rho_4\rangle) = \{f_1, f_3\}$ which, when applied to (4.9), gives us $\mathcal{U}(1)$ and $\mathcal{U}(3)$.

We have proved that $\mathcal{U} = \{1, 3, 4, 5, 6, 7, 9, 10, 12, 13, 14, 15, 16, 18, 19, 21\}$ satisfies the conditions defined in figure 4.3 for the evaluation of $\langle 0, \rho_0\rangle$ and $\mathcal{U}(0)$, that is, it satisfies the slice specification $[\Sigma \vdash \langle 0, \rho_0\rangle \Downarrow v] \models_{sl} (\mathcal{U}, \mathcal{D})$. In the next section we'll prove that $\mathcal{U}$ is indeed a valid slice of the evaluation of $\langle 0, \rho_0\rangle$.

The transformation $\mathcal{T}^0_{\mathcal{U}}$ replaces the expressions labeled by $\{2, 8, 11, 17, 20\}$ with 0. The program is transformed to:

$$\begin{aligned}
\mathcal{T}^0_{\mathcal{U}}(e_0) = (\texttt{let}\quad & f_1 = (\texttt{fn}\, x, y \Rightarrow x^{31})^1 \\
& f_2 = 0^2 \\
& f_3 = (\texttt{fn}\, c, d \Rightarrow (*\ d\ \ d)^{33})^3 \\
\texttt{in}\quad & \mathcal{T}^0_{\mathcal{U}}(e_4))^0
\end{aligned}$$

where

$$\begin{aligned}
\mathcal{T}^0_{\mathcal{U}}(e_4) = (\texttt{let}\quad & g = (\texttt{if}\ \mathcal{T}^0_{\mathcal{U}}(e_6)\ \texttt{then}\ f_1^7\ \texttt{else}\ 0^8)^5 \\
& h = (\texttt{if}\ \mathcal{T}^0_{\mathcal{U}}(e_{10})\ \texttt{then}\ 0^{11}\ \texttt{else}\ {f_3}^{12})^9 \\
\texttt{in} & \\
& (+\ (g^{15}\ \mathcal{T}^0_{\mathcal{U}}(e_{16})\ \ 0^{17})^{14}\ \ (h^{19}\ 0^{20}\ \ \mathcal{T}^0_{\mathcal{U}}(e_{21}))^{18})^{13})^4
\end{aligned}$$

Example 2 Our second example considers the same expressions e_0 and e_4 as before but it considers the new expression e_{13}:

$$e_{13} = (g^{14}\ e_{15}\ \ (h^{17}\ e_{18}\ \ e_{19})^{16})^{13}$$

and

$$\Sigma \vdash \langle 14, \rho_{13}\rangle \Downarrow [1,\ \rho_0]\ \wedge\ \Sigma \vdash \langle 31, \rho_0'\rangle \Downarrow v_{13}$$

Using $\mathcal{U}(13)$ we apply the application rule (**r 7.5**) to the evaluation of $\langle 13, \rho_{13}\rangle$ and derive the following constraints:

$$\begin{array}{l} \mathcal{U}(14) \quad \wedge \quad \mathcal{D}(\langle 14, \rho_{13}\rangle) \subseteq \mathcal{D}(\langle 13, \rho_{13}\rangle) \\ x \in \mathcal{D}(\langle 31, \rho'_0\rangle) \;\Rightarrow\; \mathcal{U}(15) \;\Rightarrow\; \mathcal{D}(\langle 15, \rho_{13}\rangle) \subseteq \mathcal{D}(\langle 13, \rho_{13}\rangle) \\ y \in \mathcal{D}(\langle 31, \rho'_0\rangle) \;\Rightarrow\; \mathcal{U}(16) \;\Rightarrow\; \mathcal{D}(\langle 16, \rho_{13}\rangle) \subseteq \mathcal{D}(\langle 13, \rho_{13}\rangle) \end{array} \tag{4.15}$$

Clearly $\mathcal{D}(\langle 31, \rho'_0\rangle) = \{x\}$ and $\mathcal{D}(\langle 14, \rho_{13}\rangle) = \{g\}$ are sound values which, when applied to (4.15), generate the following results:

$$\mathcal{U}(15) \quad \wedge \quad \mathcal{D}(\langle 13, \rho_{13}\rangle) = \{g\}$$

Using the results above and (4.11) we derive $\mathcal{U}(5)$ and $\mathcal{D}(\langle 5, \rho_4\rangle) \subseteq \mathcal{D}(\langle 4, \rho_4\rangle)$. A quick inspection to the evaluation of $\langle 5, \rho_4\rangle$ will let us determine $\mathcal{U}(6)$, $\mathcal{U}(7)$ and $\mathcal{D}(\langle 5, \rho_4\rangle) = \{f_1\}$. Then $\mathcal{D}(\langle 4, \rho_4\rangle = \{f_1\}$ which, when applied to (4.9), generates $\mathcal{U}(1)$.

Therefore $\mathcal{U} = \{1, 4, 5, 6, 7, 13, 14, 15\}$ satisfies the conditions of the slice specification for the evaluation of $\langle 0, \rho_0\rangle$. The program is transformed to:

$$\begin{array}{rl} \mathcal{T}^0_{\mathcal{U}}(e_0) = (\texttt{let} & f_1 = (\texttt{fn}\, x, y \Rightarrow x^{31})^1 \\ & f_2 = 0^2 \\ & f_3 = 0^3 \\ \texttt{in} & \mathcal{T}^0_{\mathcal{U}}(e_4))^0 \end{array}$$

where

$$\begin{array}{rl} \mathcal{T}^0_{\mathcal{U}}(e_4) = (\texttt{let} & g = (\texttt{if}\; \mathcal{T}^0_{\mathcal{U}}(e_6)\; \texttt{then}\; f_1^7\; \texttt{else}\; 0^8)^5 \\ & h = 0^9 \\ \texttt{in} & \\ & (g^{14}\; \mathcal{T}^0_{\mathcal{U}}(e_{15})\;\; 0^{16})^{13})^4 \end{array}$$

Example 3 The third example uses the same definition of e_0 where e_4 is defined by:

$$\begin{array}{rl} e_4 = (\texttt{let} & g = (\texttt{fn}\, gg \Rightarrow (gg^7\; e_8\;\; e_9)^6)^5 \\ & h = (\texttt{fn}\, hh \Rightarrow (hh^{12}\; e_{13}\;\; e_{14})^{11})^{10} \\ \texttt{in} & e_{15})^4 \end{array}$$

and

$$e_{15} = (+\ (g^{17}\ f_1^{18})^{16}\ \ (g^{20}\ f_2^{21})^{19}\ \ (h^{23}\ f_2^{24})^{22}\ \ (h^{26}\ f_3^{27})^{25})^{15}$$

Let's analyze expressions $(g^{17}\ f_1^{18})^{16}$ and $(g^{20}\ f_2^{21})^{19}$ first. Both f_1 and f_2 flow to gg^7 in $(gg^7\ e_8\ \ e_9)^6$. Clearly, the dependency sets of the configurations that evaluate the bodies of the closures $[1,\ \rho_0]$ and $[2,\ \rho_0]$ that flow to gg^7 are $\{x\}$ and $\{a\}$, the first arguments of f_1 and f_2. Therefore $\mathcal{U}(7)$ and $\mathcal{U}(8)$ get marked.

Similarly, $[2,\ \rho_0]$ and $[3,\ \rho_0]$ flow to hh^{12} in $(hh^{12}\ e_{13}\ \ e_{14})^{11}$ when $(h^{23}\ f_2^{24})^{22}$ and $(h^{26}\ f_3^{27})^{25}$ are executed. The dependency sets of the bodies of f_2 and f_3 are $\{a\}$ and $\{d\}$, the first and second parameters respectively. Therefore, $\mathcal{U}(12)$ and both $\mathcal{U}(13)$ and $\mathcal{U}(14)$ get marked. Only e_9 remains outside of the slice. The transformation yields:

$$\begin{aligned} e_4 = (\texttt{let}\quad & g = (\texttt{fn}\ gg \Rightarrow (gg^7\ \mathcal{T}^0_{\mathcal{U}}(e_8)\ \ 0^9)^6)^5 \\ & h = (\texttt{fn}\ hh \Rightarrow (hh^{12}\ \mathcal{T}^0_{\mathcal{U}}(e_{13})\ \ \mathcal{T}^0_{\mathcal{U}}(e_{14}))^{11})^{10} \\ \texttt{in}\quad & \mathcal{T}^0_{\mathcal{U}}(e_{15}))^4 \end{aligned}$$

4.5.2 Correctness of the Specification

We prove that the specification generates valid slices, that is, if $[\Sigma \vdash \mathcal{C} \Downarrow v] \models_{sl} (\mathcal{U}, \mathcal{D})$ then $\mathcal{U}$ is a valid slice of $\mathcal{C}$. In order to do this we show that $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$ and the derivation tree of the execution of the configuration under the slice semantics form a simulation parameterized by $\mathcal{U}$ and $\mathcal{D}$.

We make this explicit by first relating the values of both derivation trees. The relation $\dot{v}\ \ \mathcal{V}^{sl}_{\mathcal{D}}\ \ v$ between values is parameterized by a mapping $\mathcal{D}$ and is defined as the smallest relation that satisfies:

$$
\begin{aligned}
[l',\, \dot\rho]\ \mathcal{V}^{sl}_{\mathcal{D}}\ [l,\, \rho] \quad &\Leftrightarrow \quad \begin{cases} l = l' \ \wedge\ (\dot\rho\ \mathcal{R}_\star\ \rho) \\ \forall x \in \mathcal{D}([l,\, \rho]).\ x \in dom(\rho) \cap dom(\dot\rho) \ \ \wedge\ \ \dot\rho(x)\ \mathcal{V}^{sl}_{\mathcal{D}}\ \rho(x) \end{cases} \\
c\ \mathcal{V}^{sl}_{\mathcal{D}}\ c' \quad &\Leftrightarrow \quad c = c' \\
\langle l', \dot\rho\rangle\ \mathcal{R}^{sl}_{\mathcal{D}}\ \langle l, \rho\rangle \quad &\Leftrightarrow \quad \begin{cases} l = l' \ \wedge\ (\dot\rho\ \mathcal{R}_\star\ \rho) \\ \forall x \in \mathcal{D}(\langle l, \rho\rangle).\ \dot\rho(x)\ \mathcal{V}^{sl}_{\mathcal{D}}\ \rho(x) \end{cases}
\end{aligned}
\tag{4.16}
$$

where only values mapped to useful variables are guaranteed to agree.

The relation is then extended to nodes of the derivation trees $[(\Sigma,\mathcal{U}) \vdash \langle l, \dot\rho\rangle \Downarrow^\star \dot v]$ and $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$. Given $\mathcal{U}$ and $\mathcal{D}$, $\mathcal{R}^{sl}_{\mathcal{U},\mathcal{D}}$ is defined by:

$$
\mathcal{N}\ \mathcal{R}^{sl}_{\mathcal{U},\mathcal{D}}\ \mathcal{M} \quad \Leftrightarrow \quad \begin{cases} \mathcal{N} = (\Sigma,\mathcal{U}) \vdash \langle l, \dot\rho\rangle \Downarrow^\star \dot v \\ \mathcal{M} = \Sigma \vdash \langle l, \rho\rangle \Downarrow v \\ \langle l, \dot\rho\rangle\ \mathcal{R}^{sl}_{\mathcal{D}}\ \langle l, \rho\rangle \\ \mathcal{U}(l) \Rightarrow \dot v\ \mathcal{V}^{sl}_{\mathcal{D}}\ v \end{cases}
\tag{4.17}
$$

where only values produced by configurations marked by $\mathcal{U}$ are guaranteed to agree.

Let's define $\preceq^{sl}_{\mathcal{U},\mathcal{D}}$ as the simulation induced by $\mathcal{R}^{sl}_{\mathcal{D}}$. The next lemma claims that if we can show $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \models_{sl} (\mathcal{U}, \mathcal{D})$ then the corresponding evaluation under the slice semantics succeeds, and their derivation trees form a $\mathcal{R}^{sl}_{\mathcal{D}}$-simulation:

Lemma 4.5.4 *If $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \models_{sl} (\mathcal{U}, \mathcal{D})$ and $\langle l, \dot\rho\rangle\ \mathcal{R}^{sl}_{\mathcal{D}}\ \langle l, \rho\rangle$ then $\exists \dot v$:*

$$[(\Sigma,\mathcal{U}) \vdash \langle l, \dot\rho\rangle \Downarrow^\star \dot v]\ \preceq^{sl}_{\mathcal{U},\mathcal{D}}\ [\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$$

Proof: *We first consider the case where $\neg\mathcal{U}(l)$. By applying rule* (**r 6.1**) *we derive $(\Sigma,\mathcal{U}) \vdash \langle l, \dot\rho\rangle \Downarrow^\star \dot v$ where $\dot v = \star$. Since rule* (**r 6.1**) *generates a derivation tree with no subtrees, according to the definition of simulation (eq. 2.9) we only need to show*

$$((\Sigma,\mathcal{U}) \vdash \langle l, \dot\rho\rangle \Downarrow^\star \dot v)\ \mathcal{R}^{sl}_{\mathcal{U},\mathcal{D}}\ (\Sigma \vdash \langle l, \rho\rangle \Downarrow v)$$

This is proved by applying equation (4.17) to $(\langle l,\dot\rho\rangle\ \mathcal{R}^{sl}_{\mathcal{D}}\ \langle l,\rho\rangle)$ *and* $\neg\mathcal{U}(l)$.

For $\mathcal{U}(l)$ *we proceed by rule induction on* $[\Sigma \vdash \langle l,\rho\rangle \Downarrow v]$. *The first three rules (axioms) generate derivation trees with no subtrees. According to equation 2.9, it is only necessary to show* $((\Sigma,\mathcal{U}) \vdash \langle l,\dot\rho\rangle \Downarrow^\star \dot v)\ \mathcal{R}^{sl}_{\mathcal{D}}\ (\Sigma \vdash \langle l,\rho\rangle \Downarrow v)$. *For these cases, it is enough to show that* $\exists \dot v$ *s.t.*

$$[(\Sigma,\mathcal{U}) \vdash \langle l,\dot\rho\rangle \Downarrow^\star \dot v] \quad \wedge \quad (\dot v\ \mathcal{V}^{sl}_{\mathcal{D}}\ v)$$

The first three rules are:

(r 1.1) $\Sigma(l) = c$ *and* $v = c$.

By applying rule (**r 6.2**) *to* $\Sigma(l) = c$ *and* $\mathcal{U}(l)$ *we derive* $[(\Sigma,\mathcal{U}) \vdash \langle l,\dot\rho\rangle \Downarrow^\star \dot v]$ *and* $\dot v = c$. *Using equation (4.16) we get* $(c\ \mathcal{V}^{sl}_{\mathcal{D}}\ c)$, *i.e.,* $(\dot v\ \mathcal{V}^{sl}_{\mathcal{D}}\ \dot v)$.

(r 1.2) $\Sigma(l) = x$, $x \in dom(\rho)$ *and* $v = \rho(x)$.

By applying the definition of $[\Sigma \vdash \langle l,\rho\rangle \Downarrow v] \models^{loc}_{sl} (\mathcal{U},\mathcal{D})$ *to* $\Sigma(l) = x$ *and* $\mathcal{U}(l)$ *we get* $x \in \mathcal{D}(\langle l,\rho\rangle)$ *which, together with* $(\langle l,\dot\rho\rangle\ \mathcal{R}^{sl}_{\mathcal{D}}\ \langle l,\rho\rangle)$, *is used to derive* $x \in dom(\dot\rho)$ *and* $(\dot\rho(x)\ \mathcal{V}^{sl}_{\mathcal{D}}\ \rho(x))$. *We apply rule* (**r 6.3**) *to* $\Sigma(l) = x$ *and* $x \in dom(\dot\rho)$ *and derive* $[(\Sigma,\mathcal{U}) \vdash \langle l,\dot\rho\rangle \Downarrow^\star \dot v]$ *where* $\dot v = \dot\rho(x)$. *Therefore* $(\dot v\ \mathcal{V}^{sl}_{\mathcal{D}}\ v)$

(r 1.3) $\Sigma(l) \in Fun$ *and* $v = [l,\ \rho]$.

We apply rule (**r 6.4**) *to* $\Sigma(l) \in Fun$ *and derive* $[(\Sigma,\mathcal{U}) \vdash \langle l,\dot\rho\rangle \Downarrow^\star \dot v]$ *where* $\dot v = [l,\ \dot\rho]$. *We need to prove* $([l,\ \dot\rho]\ \mathcal{V}^{sl}_{\mathcal{D}}\ [l,\ \rho])$.

Using $[\Sigma \vdash \langle l,\rho\rangle \Downarrow v] \models^{loc}_{sl} (\mathcal{U},\mathcal{D})$ *and* $\mathcal{U}(l)$, *we apply rule* (**r 7.2**) *to* $\Sigma(l) \in Fun$ *and derive* $\mathcal{D}([l,\ \rho]) \subseteq \mathcal{D}(\langle l,\rho\rangle$. *Using equation (4.16) and* $(\langle l,\dot\rho\rangle\ \mathcal{V}^{sl}_{\mathcal{D}}\ \langle l,\rho\rangle)$ *we get*

$$\forall x \in \mathcal{D}(\langle l,\rho\rangle).\ (\dot\rho(x)\ \mathcal{V}^{sl}_{\mathcal{D}}\ \rho(x))$$

We have $\mathcal{D}([l,\ \rho]) \subseteq \mathcal{D}(\langle l,\rho\rangle$. *This means that* $\forall x \in \mathcal{D}([l,\rho])$, $x \in \mathcal{D}(\langle l,\rho\rangle)$ *and, according* *to the result above,* $(\dot{\rho}(x)\ \ \mathcal{V}^{sl}_{\mathcal{D}}\ \ \rho(x))$. *This is enough to show, using equation (4.16),* $([l,\ \dot{\rho}]\ \ \mathcal{V}^{sl}_{\mathcal{D}}\ \ [l,\ \rho])$*, i.e.,* $(\dot{v}\ \ \mathcal{V}^{sl}_{\mathcal{D}}\ \ v)$.

The rest of the rules correspond to derivation trees of more than one node. Given $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$*, we need to find* $\dot{v}$ *such that* $[(\Sigma,\mathcal{U}) \vdash \langle l, \dot{\rho}\rangle \Downarrow^{\star} \dot{v}]$ *and*

$$[(\Sigma,\mathcal{U}) \vdash \langle l, \dot{\rho}\rangle \Downarrow^{\star} \dot{v}]\ \ \preceq^{sl}_{\mathcal{U},\mathcal{D}}\ \ [\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \quad [\mathbf{f\ 6.1}]$$

In order to do this we use $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \models_{sl} (\mathcal{U},\mathcal{D})$ *and* $\langle l, \dot{\rho}\rangle\ \ \mathcal{R}^{sl}_{\mathcal{D}}\ \ \langle l, \rho\rangle$. *By applying definition 4.5.3 to* $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \models_{sl} (\mathcal{U},\mathcal{D})$ *and lemma 4.5.2 to* $\mathcal{U}(l)$ *we get:*

- $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \models^{loc}_{sl} (\mathcal{U},\mathcal{D}) \qquad [\mathbf{f\ 6.2}]$
- $\forall [\Sigma \vdash \langle l', \rho'\rangle \Downarrow v'] \in \mathit{reach}([\Sigma \vdash \langle l, \rho\rangle \Downarrow v])$.
  $[\Sigma \vdash \langle l', \rho'\rangle \Downarrow v'] \models_{sl} (\mathcal{U},\mathcal{D}) \qquad [\mathbf{f\ 6.3}]$

where $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \models^{loc}_{sl} (\mathcal{U},\mathcal{D})$ *is defined by the rules in figure 4.3. The definition of* $\langle l, \dot{\rho}\rangle\ \ \mathcal{R}^{sl}_{\mathcal{D}}\ \ \langle l, \rho\rangle$ *is expanded to:*

$$\forall x \in \mathcal{D}(\langle l, \rho\rangle).\ \dot{\rho}(x)\ \ \mathcal{V}^{sl}_{\mathcal{D}}\ \ \rho(x)$$

From the above it is easy to prove the following useful result. Given $(\langle l, \dot{\rho}\rangle\ \ \mathcal{R}^{sl}_{\mathcal{D}}\ \ \langle l, \rho\rangle)$,

$$\forall l'.\ \mathcal{D}(\langle l', \rho\rangle) \subseteq \mathcal{D}(\langle l, \rho\rangle) \Rightarrow \langle l', \dot{\rho}\rangle\ \ \mathcal{R}^{sl}_{\mathcal{D}}\ \ \langle l', \rho\rangle \quad [\mathbf{f\ 6.4}]$$

According to def. (2.9), in order to prove $[\mathbf{f\ 6.1}]$ *we need to show*

$$((\Sigma,\mathcal{U}) \vdash \langle l, \dot{\rho}\rangle \Downarrow^{\star} \dot{v})\ \ \mathcal{R}\ \ (\Sigma \vdash \langle l, \rho\rangle \Downarrow v) \quad [\mathbf{f\ 6.5}]$$

and that for all subtrees $[(\Sigma,\mathcal{U}) \vdash \langle l', \dot{\rho}'\rangle \Downarrow^{\star} \dot{v}']$ *of* $[(\Sigma,\mathcal{U}) \vdash \langle l, \dot{\rho}\rangle \Downarrow^{\star} \dot{v}]$ *there is a subtree* $[\Sigma \vdash \langle l', \rho'\rangle \Downarrow v']$ *of* $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$ *such that*

$$[(\Sigma,\mathcal{U}) \vdash \langle l', \dot{\rho}'\rangle \Downarrow^{\star} \dot{v}']\ \ \preceq^{sl}_{\mathcal{U},\mathcal{D}}\ \ [\Sigma \vdash \langle l', \rho'\rangle \Downarrow v'] \quad [\mathbf{f\ 6.6}]$$

This last result is usually obtained by applying the inductive hypothesis to $[\Sigma \vdash \langle l', \rho' \rangle \Downarrow v']$. *In order to do this we need to make sure that* $[\Sigma \vdash \langle l', \rho' \rangle \Downarrow v'] \models_{sl} (\mathcal{U}, \mathcal{D})$ *is true and find a suitable* $\dot{\rho}'$ *such that* $\langle l', \dot{\rho}' \rangle \ \mathcal{R} \ \langle l', \rho \rangle$.

The first condition - $[\Sigma \vdash \langle l', \rho' \rangle \Downarrow v'] \models_{sl} (\mathcal{U}, \mathcal{D})$ *- is already true by* [**f 6.3**]. *The second condition -* $\langle l', \dot{\rho}' \rangle \ \mathcal{R} \ \langle l', \rho \rangle$ *- requires more work, the simplest case being when* $\dot{\rho}' = \dot{\rho}$ *and* $\mathcal{D}(\langle l', \rho \rangle) \subseteq \mathcal{D}(\langle l, \rho \rangle)$ *(from* $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \models_{sl}^{loc} (\mathcal{U}, \mathcal{D})$*). In this case, we use* [**f 6.4**].

At this point of the proof we have enough information to find $\dot{v}$ *such that* $[(\Sigma,\mathcal{U}) \vdash \langle l, \dot{\rho} \rangle \Downarrow^{\star} \dot{v}]$. *We do this by applying one of the rules of figure 4.2 to the* $[(\Sigma,\mathcal{U}) \vdash \langle l', \dot{\rho}' \rangle \Downarrow^{\star} \dot{v}']$ *subtrees generated in* [**f 6.6**]. *Finally, in order to prove* [**f 6.5**] *we only need to show that* $(\dot{v} \ \mathcal{V}_{\mathcal{D}}^{sl} \ v)$ *is true. This completes the proof.*

We work through each of the remaining rules following the steps sketched above:

(**r 1.4**) $\Sigma(l) = (p \ t_1^{l_1} \dots t_n^{l_n})$ *and*

$$i \in \{1, \dots, \} : [\Sigma \vdash \langle l_i, \rho \rangle \Downarrow c_i] \quad \wedge \quad v = \bar{p}(c_1 \dots c_n) \in \mathit{Const}$$

By applying the definition of $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \models_{sl}^{loc} (\mathcal{U}, \mathcal{D})$ *to* $\Sigma(l) = (p \ t_1^{l_1} \dots t_n^{l_n})$ *and* $\mathcal{U}(l)$ *we get:*

$$\forall i = 1, \dots, n : \begin{cases} \mathcal{U}(l_i) \\ \mathcal{D}(\langle l_i, \rho \rangle) \subseteq \mathcal{D}(\langle l, \rho \rangle) \end{cases}$$

Using $\langle l, \dot{\rho} \rangle \ \mathcal{R}_{\mathcal{D}}^{sl} \ \langle l, \rho \rangle$, $\mathcal{D}(\langle l_i, \rho \rangle) \subseteq \mathcal{D}(\langle l, \rho \rangle)$ *and* [**f 6.4**] *we derive* $\langle l_i, \dot{\rho} \rangle \ \mathcal{R}_{\mathcal{D}}^{sl} \ \langle l_i, \rho \rangle$, *for* $i \in \{1, \dots, n\}$. *Therefore, we can apply the inductive hypothesis to* $[\Sigma \vdash \langle l_i, \rho \rangle \Downarrow c_i]$ *and derive*

$$[(\Sigma,\mathcal{U}) \vdash \langle l_i, \dot{\rho} \rangle \Downarrow^{\star} \dot{v}_i] \ \preceq_{\mathcal{U},\mathcal{D}}^{sl} \ [\Sigma \vdash \langle l_i, \rho \rangle \Downarrow c_i] \quad i \in \{1, \dots, n\}$$

In particular, for each $1 \leq i \leq n$, *we know* $((\Sigma,\mathcal{U}) \vdash \langle l_i, \dot{\rho} \rangle \Downarrow^{\star} \dot{v}_i) \ \mathcal{R}_{\mathcal{D}}^{sl} \ (\Sigma \vdash \langle l_i, \rho \rangle \Downarrow c_i)$. *By applying the definition of* $\mathcal{R}_{\mathcal{D}}^{sl}$ *in equation (4.17) and using* $\mathcal{U}(l_i)$ *we derive* $(\dot{v}_i \ \mathcal{V}_{\mathcal{D}}^{sl} \ c_i)$ *which, according to the definition of* $\mathcal{V}_{\mathcal{D}}^{sl}$, *implies* $\dot{v}_i = c_i$. *Therefore, the result above is* *re-written as:*

$$[(\Sigma,\mathcal{U}) \vdash \langle l_i, \dot\rho\rangle \Downarrow^\star c_i] \ \preceq^{sl}_{\mathcal{U},\mathcal{D}} \ [\Sigma \vdash \langle l_i, \rho\rangle \Downarrow c_i] \quad i \in \{1,\ldots,n\}$$

By applying rule (**r 6.5**) *to the above and* $\Sigma(l) = (p\ t_1^{l_1} \ldots t_n^{l_n})$ *we derive* $[(\Sigma,\mathcal{U}) \vdash \langle l, \dot\rho\rangle \Downarrow^\star \dot v]$ *where* $\dot v = \bar p(c_1 \ldots c_n)$, *that is,* $\dot v = v$. *Therefore* $(\dot v\ \ \mathcal{V}^{sl}_{\mathcal{D}}\ \ v)$ *(eq. 2.13).*

(**r 1.5**) $\quad \Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n})$

$$\begin{aligned} i \in \{1,\ldots,n\}: \quad & [\Sigma \vdash \langle l_i, \rho\rangle \Downarrow v_i] \\ & [\Sigma \vdash \langle l_0, \rho\rangle \Downarrow [l_f,\ \rho_0]] \\ & \Sigma(l_f) = (\mathtt{fn}\ x_1 \ldots x_n \Rightarrow t_b^{l_b}) \\ & [\Sigma \vdash \langle l_b, \rho_0'\rangle \Downarrow v] \end{aligned}$$

where $\rho_0' = \rho_0[x_1 \mapsto v_1]\ldots[x_n \mapsto v_n]$.

By applying the definition of $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \models^{loc}_{sl} (\mathcal{U},\mathcal{D})$ *to* $\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n})$ *we get*

$$\begin{aligned} & \mathcal{U}(l_0)\ \wedge\ \mathcal{U}(l_b)\ \wedge\ \mathcal{D}(\langle l_0, \rho\rangle) \subseteq \mathcal{D}(\langle l, \rho\rangle) \\ & \mathcal{D}(\langle l_b, \rho_0'\rangle) - \{x_1,\ldots,x_n\} \subseteq \mathcal{D}([l_f,\ \rho_0]) \\ & \forall i = 1,\ldots,n: \begin{cases} x_i \in \mathcal{D}(\langle l_b, \rho_0'\rangle) \Rightarrow \mathcal{U}(l_i) & [\mathbf{f\ 6.7}] \\ \mathcal{U}(l_i) \Rightarrow \mathcal{D}(\langle l_i, \rho\rangle) \subseteq \mathcal{D}(\langle l, \rho\rangle) \end{cases} \end{aligned}$$

For $i \in \{1,\ldots,n\}$, *we have:*

$$\mathcal{U}(l_i) \Rightarrow \begin{cases} \mathcal{D}(\langle l_i, \rho\rangle) \subseteq \mathcal{D}(\langle l, \rho\rangle) \text{ which implies } \langle l_i, \dot\rho\rangle\ \ \mathcal{R}^{sl}_{\mathcal{D}}\ \ \langle l_i, \rho\rangle \text{ using } [\mathbf{f\ 6.4}] \\ [\Sigma \vdash \langle l_i, \rho\rangle \Downarrow v_i] \models_{sl} (\mathcal{U},\mathcal{D}) \end{cases}$$

Therefore, if $\mathcal{U}(l_i)$ *then we can apply the inductive hypothesis to* $[\Sigma \vdash \langle l_i, \rho\rangle \Downarrow v_i]$ *and derive:*

$$\exists \dot v_i.\ [(\Sigma,\mathcal{U}) \vdash \langle l_i, \dot\rho\rangle \Downarrow^\star \dot v_i] \ \preceq^{sl}_{\mathcal{U},\mathcal{D}} \ [\Sigma \vdash \langle l_i, \rho\rangle \Downarrow v_i]$$

If instead we have $\neg\mathcal{U}(l_i)$ *then* $(\Sigma,\mathcal{U}) \vdash \langle l_i, \dot\rho\rangle \Downarrow^\star \dot v_i$ *and* $\dot v = \star$ *(rule* (**r 6.1**)*). In this case,* $[(\Sigma,\mathcal{U}) \vdash \langle l_i, \dot\rho\rangle \Downarrow^\star \dot v_i] \ \preceq^{sl}_{\mathcal{U},\mathcal{D}} \ [\Sigma \vdash \langle l_i, \rho\rangle \Downarrow v_i]$ *trivially holds. Therefore,*

$$\exists \dot v_i.\ [(\Sigma,\mathcal{U}) \vdash \langle l_i, \dot\rho\rangle \Downarrow^\star \dot v_i] \ \preceq^{sl}_{\mathcal{U},\mathcal{D}} \ [\Sigma \vdash \langle l_i, \rho\rangle \Downarrow v_i] \quad i \in \{1,\ldots,n\} \quad [\mathbf{f\ 6.8}]$$

We use $\mathcal{D}(\langle l_0, \rho\rangle) \subseteq \mathcal{D}(\langle l, \rho\rangle)$ *and* [**f 6.4**] *to derive* $\langle l_0, \dot\rho\rangle\ \mathcal{R}^{sl}_{\mathcal{D}}\ \langle l_0, \rho\rangle$. *Therefore we can also apply the inductive hypothesis to* $[\Sigma \vdash \langle l_0, \rho\rangle \Downarrow [l_f,\ \rho_0]]$ *and find that there exists* $\dot v_0$ *such that*

$$[(\Sigma,\mathcal{U}) \vdash \langle l_0, \dot\rho\rangle \Downarrow^\star \dot v_0]\ \preceq^{sl}_{\mathcal{U},\mathcal{D}}\ [\Sigma \vdash \langle l_0, \rho\rangle \Downarrow [l_f,\ \rho_0]]$$

By applying the definition of $\mathcal{R}^{sl}_{\mathcal{D}}$ *(eq. 4.17) and using* $\mathcal{U}(l_0)$ *we derive* $(\dot v_0\ \ \mathcal{V}^{sl}_{\mathcal{D}}\ \ [l_f,\ \rho_0])$. *This means that* $\dot v_0$ *must be of the form* $[l_f,\ \dot\rho_0]$ *where* $\dot\rho_0$ *must respect the conditions in (4.16).*

Let $\dot\rho'_0 = \dot\rho_0[x_1 \mapsto \dot v_1]\ldots[x_n \mapsto \dot v_n]$. *We need to prove* $\langle l_b, \dot\rho'_0\rangle\ \mathcal{R}^{sl}_{\mathcal{D}}\ \langle l_b, \rho'\rangle$*:*

$$\forall x \in \mathcal{D}(\langle l_b, \rho'\rangle).\ (\dot\rho'_0(x)\ \ \mathcal{V}^{sl}_{\mathcal{D}}\ \ \rho'_0(x))$$

Given $x \in \mathcal{D}(\langle l_b, \rho'\rangle)$, *we have two cases:*

- $x \neq x_i$, *for all* $i \in \{i, \ldots, n\}$. *Then* $\dot\rho'_0(x) = \dot\rho_0(x)$ *and* $\rho'_0(x) = \rho_0(x)$. *Using*

  $$\mathcal{D}(\langle l_b, \rho'\rangle) - \{x_1, \ldots, x_n\} \subseteq \mathcal{D}([l_f,\ \rho_0])$$

  we derive $x \in \mathcal{D}([l_f,\ \rho_0])$. *By applying the definition of* $\mathcal{R}^{sl}_{\mathcal{D}}$ *to* $([l_f,\ \dot\rho_0]\ \ \mathcal{V}^{sl}_{\mathcal{D}}\ \ [l_f,\ \rho_0])$ *we get* $(\dot\rho_0(x)\ \ \mathcal{V}^{sl}_{\mathcal{D}}\ \ \rho_0(x))$. *Therefore* $(\dot\rho'_0(x)\ \ \mathcal{V}^{sl}_{\mathcal{D}}\ \ \rho'_0(x))$.

- $x = x_i$, *for some* $i \in \{i, \ldots, n\}$. *Then* $\dot\rho'_0(x) = \dot v_i$ *and* $\rho'_0(x) = v_i$. *Using* [**f 6.7**] *and* $x_i \in \mathcal{D}(\langle l_b, \rho'\rangle)$ *we derive* $\mathcal{U}(l_i)$. *By applying the definitions of* $\preceq^{sl}_{\mathcal{U},\mathcal{D}}$ *and* $\mathcal{R}^{sl}_{\mathcal{D}}$ *to* [**f 6.8**] *and* $\mathcal{U}(l_i)$ *we get* $(\dot v_i\ \ \mathcal{V}^{sl}_{\mathcal{D}}\ \ v_i)$, *i.e.,* $(\dot\rho'_0(x)\ \ \mathcal{V}^{sl}_{\mathcal{D}}\ \ \rho'_0(x))$. *Therefore* $x_i \in \mathcal{D}(\langle l_b, \rho'\rangle)$ *implies* $(\dot\rho'_0(x)\ \ \mathcal{V}^{sl}_{\mathcal{D}}\ \ \rho'_0(x))$, *which is what we were looking for.*

We have proved $\langle l_b, \dot\rho'_0\rangle\ \mathcal{R}^{sl}_{\mathcal{D}}\ \langle l_b, \rho'\rangle$. *Now we can apply the inductive hypothesis to* $[\Sigma \vdash \langle l_b, \rho'_0\rangle \Downarrow v]$ *and derive*

$$\exists \dot v.\ [(\Sigma,\mathcal{U}) \vdash \langle l_b, \dot\rho'_0\rangle \Downarrow^\star \dot v]\ \preceq^{sl}_{\mathcal{U},\mathcal{D}}\ [\Sigma \vdash \langle l_b, \rho'_0\rangle \Downarrow v]$$

But $\mathcal{U}(l)$ *implies* $(\dot v\ \ \mathcal{V}^{sl}_{\mathcal{D}}\ \ v)$ *(eq. (4.17)). Therefore, by applying rule* (**r 6.6**) *to*

$$\begin{array}{rll} i \in \{1, \ldots, n\}: & [\Sigma \vdash \langle l_i, \dot\rho\rangle \Downarrow \dot v_i] & \wedge\ [\Sigma \vdash \langle l_0, \dot\rho\rangle \Downarrow [l_f,\ \dot\rho_0]] \\ & \Sigma(l_f) = (\texttt{fn}\ x_1 \ldots x_n \Rightarrow t_b^{l_b}) & \wedge\ [\Sigma \vdash \langle l_b, \dot\rho'_0\rangle \Downarrow \dot v] \end{array}$$

we finally get $[\Sigma \vdash \langle l, \dot{\rho} \rangle \Downarrow \dot{v}]$ *where* $(\dot{v}\ \mathcal{V}^{sl}_{\mathcal{D}}\ v)$.

(r 1.6) $\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n})$ *and:*

$$\begin{array}{l} [\Sigma \vdash \langle l_i, \rho \rangle \Downarrow v_i] \quad i = 1, \ldots, n\ \wedge \\ [\Sigma \vdash \langle l_0, \rho \rangle \Downarrow [l_f,\ \rho_0]]\ \wedge \\ \Sigma(l_f) = (\texttt{fun}\ y\ x_1 \ldots x_n \Rightarrow t_b^{l_b})\ \wedge \\ [\Sigma \vdash \langle l_b, \rho_0' \rangle \Downarrow v] \end{array}$$

where $\rho_0' = \rho_0[y \mapsto \langle l_f, \rho_0 \rangle][x_1 \mapsto v_1] \ldots [x_n \mapsto v_n]$.

The proof is similar to the one shown for rule (**r 1.5**).

(r 1.7) $\Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})$ *and:*

$$[\Sigma \vdash \langle l_0, \rho \rangle \Downarrow \mathit{true}]\ \wedge\ [\Sigma \vdash \langle l_1, \rho \rangle \Downarrow v]$$

$[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \models^{loc}_{sl} (\mathcal{U}, \mathcal{D})$ *corresponds to rule* (**r 7.7**) *where*

$$\begin{array}{rl} & \mathcal{U}(l_0)\ \wedge\ \mathcal{U}(l_1) \\ i = 0, 1: & \mathcal{D}(\langle l_i, \rho \rangle) \subseteq \mathcal{D}(\langle l, \rho \rangle) \quad [\mathbf{f\ 6.9}] \end{array}$$

By applying [**f 6.4**] *to* [**f 6.9**] *we get* $i \in \{0, 1\}.\ \langle l_i, \dot{\rho} \rangle\ \mathcal{R}^{sl}_{\mathcal{D}}\ \langle l_i, \rho \rangle$. *Therefore we can apply the inductive hypothesis to* $[\Sigma \vdash \langle l_0, \rho \rangle \Downarrow \mathit{true}]$ *and* $[\Sigma \vdash \langle l_1, \rho \rangle \Downarrow v]$ *and derive*

$$\begin{array}{l} \exists \dot{v}_0.\ [(\Sigma, \mathcal{U}) \vdash \langle l_0, \dot{\rho} \rangle \Downarrow^{\star} \dot{v}_0]\ \preceq^{sl}_{\mathcal{U},\mathcal{D}}\ [\Sigma \vdash \langle l_0, \rho \rangle \Downarrow \mathit{true}] \\ \exists \dot{v}_1.\ [(\Sigma, \mathcal{U}) \vdash \langle l_1, \dot{\rho} \rangle \Downarrow^{\star} \dot{v}]\ \preceq^{sl}_{\mathcal{U},\mathcal{D}}\ [\Sigma \vdash \langle l_0, \rho \rangle \Downarrow v] \end{array}$$

Using the definitions of $\preceq^{sl}_{\mathcal{U},\mathcal{D}}$ *and* $\mathcal{R}^{sl}_{\mathcal{D}}$ *together with* $\mathcal{U}(l_0)$ *and* $\mathcal{U}(l_1)$ *we get*

$$\begin{array}{l} (\dot{v}_0\ \mathcal{V}^{sl}_{\mathcal{D}}\ \mathit{true}) \quad \textit{which implies} \quad \dot{v}_0 = \mathit{true} \quad [\textit{using}\ \mathbf{eq.\ (4.16)}] \\ (\dot{v}\ \mathcal{V}^{sl}_{\mathcal{D}}\ v) \end{array}$$

By applying rule (**r 6.8**) *to* $\Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})$, $[(\Sigma, \mathcal{U}) \vdash \langle l_0, \dot{\rho} \rangle \Downarrow^{\star} \mathit{true}]$ *and* $[(\Sigma, \mathcal{U}) \vdash \langle l_1, \dot{\rho} \rangle \Downarrow^{\star} \dot{v}_1]$ *we derive* $[(\Sigma, \mathcal{U}) \vdash \langle l, \dot{\rho} \rangle \Downarrow^{\star} \dot{v}]$ *where* $(\dot{v}\ \mathcal{V}^{sl}_{\mathcal{D}}\ v)$.

(r 1.8) $\Sigma(l) = (\texttt{if } t_0^{l_0} \texttt{ then } t_1^{l_1} \texttt{ else } t_2^{l_2})$ *and:*

$$\Sigma \vdash \langle l_0, \rho \rangle \Downarrow \mathit{false} \ \wedge \ \Sigma \vdash \langle l_2, \rho \rangle \Downarrow v$$

Similar to the **(r 1.7)** *case using* $i = 0, 2$.

(r 1.9) $\Sigma(l) = (\texttt{let } x = t_1^{l_1} \texttt{ in } t_2^{l_2})$ *and:*

$$[\Sigma \vdash \langle l_1, \rho \rangle \Downarrow v_1] \ \wedge \ [\Sigma \vdash \langle l_2, \rho' \rangle \Downarrow v]$$

where $\rho' = \rho[x \mapsto v_1]$ $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \models_{sl}^{loc} (\mathcal{U}, \mathcal{D})$ *corresponds to rule* **(r 7.9)** *where*

$$\begin{array}{ll} \mathcal{U}(l_2) & \\ \mathcal{D}(\langle l_2, \rho' \rangle) - \{x\} \subseteq \mathcal{D}(\langle l, \rho \rangle) & [\mathbf{f\ 6.10}] \\ x \in \mathcal{D}(\langle l_2, \rho' \rangle) \Rightarrow \mathcal{U}(l_1) & [\mathbf{f\ 6.11}] \\ \mathcal{U}(l_1) \Rightarrow \mathcal{D}(\langle l_1, \rho \rangle) \subseteq \mathcal{D}(\langle l, \rho \rangle) & [\mathbf{f\ 6.12}] \end{array}$$

If $\mathcal{U}(l_1)$ *then we can apply* **[f 6.4]** *to* **[f 6.12]** *and derive* $\langle l_1, \dot{\rho} \rangle \ \mathcal{R}_{\mathcal{D}}^{sl} \ \langle l_1, \rho \rangle$. *Therefore we could apply the inductive hypothesis to* $[\Sigma \vdash \langle l_1, \rho \rangle \Downarrow v_1]$ *and derive:*

$$\mathcal{U}(l_1) \Rightarrow \exists \dot{v}_1.\ [(\Sigma, \mathcal{U}) \vdash \langle l_1, \dot{\rho} \rangle \Downarrow^\star \dot{v}_1] \ \preceq_{\mathcal{U},\mathcal{D}}^{sl} \ [\Sigma \vdash \langle l_1, \rho \rangle \Downarrow v_1]$$

On the other hand, if $\neg\mathcal{U}(l_1)$ *then, by applying* **(r 6.1)**, *we get* $[(\Sigma, \mathcal{U}) \vdash \langle l_1, \dot{\rho} \rangle \Downarrow^\star \dot{v}_1]$ *where* $\dot{v}_1 = \star$. *Therefore* $[(\Sigma, \mathcal{U}) \vdash \langle l_1, \dot{\rho} \rangle \Downarrow^\star \dot{v}_1] \ \preceq_{\mathcal{U},\mathcal{D}}^{sl} \ [\Sigma \vdash \langle l_1, \rho \rangle \Downarrow v_1]$ *trivially holds. We have:*

$$\exists \dot{v}_1.\ [(\Sigma, \mathcal{U}) \vdash \langle l_1, \dot{\rho} \rangle \Downarrow^\star \dot{v}_1] \ \preceq_{\mathcal{U},\mathcal{D}}^{sl} \ [\Sigma \vdash \langle l_1, \rho \rangle \Downarrow v_1] \quad [\mathbf{f\ 6.13}]$$

Let $\dot{\rho}' = \dot{\rho}[x \mapsto \dot{v}_1]$. *We want to prove* $\langle l_2, \dot{\rho}' \rangle \ \mathcal{R}_{\mathcal{D}}^{sl} \ \langle l_2, \rho' \rangle$, *that is,*

$$\forall y \in \mathcal{D}(\langle l_2, \rho' \rangle).\ \dot{\rho}'(y) \ \mathcal{V}_{\mathcal{D}}^{sl} \ \rho'(y)$$

Given $y \in \mathcal{D}(\langle l_2, \rho' \rangle)$ *we have two cases:*

- $y \neq x$. *Then* $\dot{\rho}'(y) = \dot{\rho}(y)$ *and* $\rho'(y) = \rho(y)$. *Using* $y \in \mathcal{D}(\langle l_2, \rho' \rangle)$, $y \neq x$ *and* **[f 6.10]** *we derive* $y \in \mathcal{D}(\langle l, \rho \rangle)$. *By applying the definition of* $\langle l, \dot{\rho} \rangle \ \mathcal{R}_{\mathcal{D}}^{sl} \ \langle l, \rho \rangle$ *to the latter we derive* $\dot{\rho}(y) \ \mathcal{V}_{\mathcal{D}}^{sl} \ \rho(y)$, *i.e.,* $\dot{\rho}'(y) \ \mathcal{V}_{\mathcal{D}}^{sl} \ \rho'(y)$.

- $y = x$. *Then* $\dot{\rho}'(y) = \dot{v}_1$, $\rho'(y) = v_1$ *and* $x \in \mathcal{D}(\langle l_2, \rho' \rangle)$. *Using* [**f 6.11**] *and* $x \in \mathcal{D}(\langle l_2, \rho' \rangle)$ *we derive* $\mathcal{U}(l_1)$. *By applying the definitions of* $\preceq^{sl}_{\mathcal{U},\mathcal{D}}$ *and* $\mathcal{R}^{sl}_{\mathcal{D}}$ *to* [**f 6.13**] *and* $\mathcal{U}(l_1)$ *we derive* $(\dot{v}_1 \ \mathcal{V}^{sl}_{\mathcal{D}} \ v_1)$, *i.e.,* $\dot{\rho}'(y) \ \mathcal{V}^{sl}_{\mathcal{D}} \ \rho'(y)$.

We have proved $\langle l_2, \dot{\rho}' \rangle \ \mathcal{R}^{sl}_{\mathcal{D}} \ \langle l_2, \rho' \rangle$. *Now we can apply the inductive hypothesis to* $[\Sigma \vdash \langle l_2, \rho' \rangle \Downarrow v]$ *and derive:*

$$\exists \dot{v}.\ [(\Sigma, \mathcal{U}) \vdash \langle l_2, \dot{\rho}' \rangle \Downarrow^{\star} \dot{v}] \ \preceq^{sl}_{\mathcal{U},\mathcal{D}} \ [\Sigma \vdash \langle l_2, \rho' \rangle \Downarrow v]$$

But $\mathcal{U}(l_2)$ *implies* $(\dot{v} \ \mathcal{V}^{sl}_{\mathcal{D}} \ v)$ *(eq. (4.17)). Therefore, by applying rule* (**r 6.10**) *to* $[(\Sigma, \mathcal{U}) \vdash \langle l_1, \dot{\rho} \rangle \Downarrow^{\star} \dot{v}_1]$ *and* $[(\Sigma, \mathcal{U}) \vdash \langle l_2, \dot{\rho}' \rangle \Downarrow^{\star} \dot{v}]$ *we derive* $[(\Sigma, \mathcal{U}) \vdash \langle l, \dot{\rho} \rangle \Downarrow^{\star} \dot{v}]$ *where* $(\dot{v} \ \mathcal{V}^{sl}_{\mathcal{D}} \ v)$.

[**QED**]

The $\mathcal{R}^{sl}_{\mathcal{D}}$ relation is designed to make sure that all the variables considered as useful map to non-$\star$ values. The constraints that link expressions ($\mathcal{U}$) and variables ($\mathcal{D}$) make sure that the evaluations marked as part of the slice always produce non-$\star$ values. The latter is verified by the following corollary:

Corollary 4.5.5
If $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \models_{sl} (\mathcal{U}, \mathcal{D})$, $\langle l, \dot{\rho} \rangle \ \mathcal{R}^{sl}_{\mathcal{D}} \ \langle l, \rho \rangle$ *and* $\mathcal{U}(l)$ *then* $(\Sigma, \mathcal{U}) \vdash \langle l, \dot{\rho} \rangle \Downarrow^{\star} \dot{v}$ *and* $(\dot{v} \ \mathcal{V}^{sl}_{\mathcal{D}} \ v)$.

Proof: *By applying lemma 4.5.4 and unfolding the definition of* $\preceq^{sl}_{\mathcal{U},\mathcal{D}}$.

Finally, we prove that if $\mathcal{U}$ has correct information of $\Sigma \vdash \langle l, \rho \rangle \Downarrow v$ as defined by $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \models_{sl} (\mathcal{U}, \mathcal{D})$ then $\mathcal{U}$ is a valid slice of $\langle l, \rho \rangle$.

Theorem 4.5.6 *If* $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \models_{sl} (\mathcal{U}, \mathcal{D})$ *and* $\mathcal{U}(l)$ *then* $Slice(\Sigma, \mathcal{U}, \langle l, \rho \rangle)$.

Proof: *Using* $\rho \ \mathcal{R}_{\star} \ \rho$, *we can easily check* $\langle l, \rho \rangle \ \mathcal{R}^{sl}_{\mathcal{D}} \ \langle l, \rho \rangle$. *Then we can apply corollary 4.5.5 and derive* $(\Sigma, \mathcal{U}) \vdash \langle l, \rho \rangle \Downarrow^{\star} \dot{v}$ *and* $(\dot{v} \ \mathcal{V}^{sl}_{\mathcal{D}} \ v)$. *The latter is defined such that*

$\dot{v} \neq \star$ *(def 4.16). Putting together* $\Sigma \vdash \langle l, \rho \rangle \Downarrow v$, $(\Sigma,\mathcal{U}) \vdash \langle l, \rho \rangle \Downarrow^{\star} \dot{v}$ *and* $\dot{v} \neq \star$, *we get* $\mathit{Slice}(\Sigma,\mathcal{U},\langle l,\rho\rangle)$.

4.6 Slice CFA

The slice and useful variable specification $[\Sigma \vdash \mathcal{C} \Downarrow v] \models_{sl} (\mathcal{U},\mathcal{D})$ defines a constraint system that theoretically could be used to compute a valid slice $\mathcal{U}$ with dependency sets $\mathcal{D}$. Given the derivation tree produced by the evaluation of a program, the constraint system is built by making sure that the initial configuration (root) satisfies the specification and that the label in question is marked as part of the slice. The initial constraints will then trigger new constraints on the preceding configurations (subtrees) which, in turn, will do the same if the specification says so. The constraints are spread all over the nodes of the tree. Therefore, in the same way that the rules in Figure 2.2 were used to find v such that $\Sigma \vdash \mathcal{C} \Downarrow v$, the rules in 4.3 could be used to find $\mathcal{U}$ given $[\Sigma \vdash \mathcal{C} \Downarrow v]$.

The system is theoretically sound but impractical. The whole program has to be evaluated first before the constraint system is set up. Even worse, for semantics that allow non-determinism or for non-terminating programs, the system is non-computable.

The slice CFA is obtained by performing abstract interpretation on the semantics of the slice specification $[\Sigma \vdash \langle l,\rho\rangle \Downarrow v] \models_{sl} (\mathcal{U},\mathcal{D})$. The abstractions used by the slice CFA are based on the abstractions performed on binding contexts that were applied for the specification of the context CFA defined in section 3.2. Intuitively, the slice CFA is the same constraint system used for $[\Sigma \vdash \langle l,\rho\rangle \Downarrow v] \models_{sl} (\mathcal{U},\mathcal{D})$ but this time built on top of the derivation tree of $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma,\widehat{\Phi})$. This can be an infinite tree and, therefore, a co-inductive definition is required.

The slice CFA is defined by the judgment $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma,\widehat{\Phi},\mathcal{U},\widehat{\mathcal{D}})$, where $\widehat{\Phi}$ contains data and control-flow information of the evaluation of abstract interpretation of configuration $\langle l, \widehat{me}\rangle$ under abstract context $\widehat{m}$. The semantic domains related to data and control flow information are the same domains defined for $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})$ in section 3.2. The mapping $\mathcal{U}$ keeps information about the expressions that belong to the program slice while the mapping $\mathcal{D}$ keeps track of the variable instances that may have been used during its evaluation. The domains of $\mathcal{U}$ and $\widehat{\mathcal{D}}$ are:

$$\begin{array}{lcl} \mathcal{U} & \in & Lab \to Bool \\ \widehat{\mathcal{D}} & \in & (Var \times \widehat{Context}) \to Bool \end{array}$$

Definition 4.6.1 (Slice CFA) *The judgement $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$ is defined as the greatest relation that satisfies the rules in Figure 4.4.*

The constraints that define the slice CFA can be seen as the constraints defined by $[\Sigma \vdash \mathcal{C} \Downarrow v] \models^{loc}_{sl} (\mathcal{U}, \mathcal{D})$ in Figure 4.3 but this time applied to the derivation tree generated by the context CFA defined in Figure 3.2. The analysis takes advantage of its explicit manipulation of binding contexts and simplifies the representation of the dependency sets mapped by $\mathcal{D}$. The rules in Figure 4.3 deal with dependency sets associated with each binding configuration that denote the set of variables that may be used by the evaluation of the configuration. Dependency is checked by the variable rule such that variables accessed inside the slice must be part of the dependency set of the immediate binding configuration.

In other words, we are checking if a particular variable instance is used by the slice. In the analysis, information about useful variables is collected by $\widehat{\mathcal{D}}$ such that information of variable x in context $\widehat{m}$ is kept in $\widehat{\mathcal{D}}(x, \widehat{m})$, in the same way as values are kept in $\widehat{\Phi}(x, \widehat{m})$.

Thus, the predicate $x \in \mathcal{D}(\langle l, \rho\rangle)$ in $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \models_{sl} (\mathcal{U}, \mathcal{D})$ is replaced by $\widehat{\mathcal{D}}(x, \widehat{me}(x))$ in $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$. The constraints placed upon arguments in function applications (rule (**r 8.5**)) are now written as $\widehat{\mathcal{D}}(x, \widehat{m}') \Rightarrow \mathcal{U}(l_i)$, where $\widehat{m}'$ is the fresh context assigned to the evaluation of one of the function bodies flowing to the application site. The relation between the slice specification and the slice CFA is studied in more detail in the next section.

$$\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \quad \Leftrightarrow \quad \mathcal{U}(l) \Rightarrow \ldots$$

(r 8.1) $\Sigma(l) = c \ \Rightarrow\ \beta_{\mathsf{c}}(c) \sqsubseteq \Pi_{\mathsf{c}}(\widehat{\Phi}(l, \widehat{m}))$

(r 8.2) $\Sigma(l) = x \ \Rightarrow\ \widehat{\Phi}(x, \widehat{me}(x)) \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \ \wedge\ \boxed{\mathcal{D}(x, \widehat{me}(x))}$

(r 8.3) $\Sigma(l) \in \mathit{Fun} \ \Rightarrow\ \{[l,\ \widehat{me}]\} \subseteq \Pi_{\lambda}(\widehat{\Phi}(l, \widehat{m}))$

(r 8.4) $\Sigma(l) = (p\ t_1^{l_1} \ldots t_n^{l_n}) \ \Rightarrow\ \begin{cases} i = 1, \ldots, n:\ \langle l_i, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \ \wedge\ \boxed{\mathcal{U}(l_i)} \\ \hat{p}(\widehat{\Phi}(l_1, \widehat{m}), \ldots, \widehat{\Phi}(l_n, \widehat{m})) \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \end{cases}$

(r 8.5) $\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n}) \ \Rightarrow$

$$\begin{cases} \boxed{\mathcal{U}(l_0)} \ \wedge\ i = 0, \ldots, n:\ \langle l_i, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \\ \forall [l_f,\ \widehat{me}_0] \in \Pi_{\lambda}(\widehat{\Phi}(l_0, \widehat{m})): \\ \begin{cases} (\Sigma(l_f) = (\texttt{fn}\, x_1 \ldots x_n \Rightarrow t_b^{l_b}) \ \wedge\ \widehat{me}'_0 = \widehat{me}_0[x_i \mapsto \widehat{m}']) \vee \\ (\Sigma(l_f) = (\texttt{fun}\, y\ x_1 \ldots x_n \Rightarrow t_b^{l_b})) \ \wedge\ \widehat{me}'_0 = \widehat{me}_0[y \mapsto \widehat{m}'][x_i \mapsto \widehat{m}']) \Rightarrow \\ \begin{cases} \widehat{m}' = \widehat{\mathcal{I}}(\Sigma, l, \widehat{m}, l_f) \\ \langle l_b, \widehat{me}'_0\rangle \models^{\widehat{m}'} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \ \wedge\ \boxed{\mathcal{U}(l_b)} \\ i = 1, \ldots, n:\ \widehat{\Phi}(l_i, \widehat{m}) \sqsubseteq \widehat{\Phi}(x_i, \widehat{m}') \ \wedge\ \boxed{\mathcal{D}(x_i, \widehat{m}') \Rightarrow \mathcal{U}(l_i)} \\ \widehat{\Phi}(l_b, \widehat{m}') \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \end{cases} \\ \Sigma(l_f) = (\texttt{fun}\, y\ x_1 \ldots x_n \Rightarrow t_b^{l_b}) \Rightarrow \{[l_f,\ \widehat{me}_0]\} \subseteq \Pi_{\lambda}(\widehat{\Phi}(y, \widehat{m}')) \end{cases} \end{cases}$$

(r 8.6) $\Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2}) \Rightarrow$

$$\begin{cases} \langle l_i, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \quad i \in \{0, 1, 2\} \\ \beta_{\mathsf{c}}(\mathit{true}) \sqsubseteq \Pi_{\mathsf{c}}(\widehat{\Phi}(l_0, \widehat{m})) \ \Rightarrow\ (\widehat{\Phi}(l_1, \widehat{m}) \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \ \wedge\ \boxed{\mathcal{U}(l_1)}) \\ \beta_{\mathsf{c}}(\mathit{false}) \sqsubseteq \Pi_{\mathsf{c}}(\widehat{\Phi}(l_0, \widehat{m})) \ \Rightarrow\ (\widehat{\Phi}(l_2, \widehat{m}) \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \ \wedge\ \boxed{\mathcal{U}(l_2)}) \end{cases}$$

(r 8.7) $\Sigma(l) = (\texttt{let}\ x = t_1^{l_1}\ \texttt{in}\ t_2^{l_2}) \Rightarrow \begin{cases} \langle l_1, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \ \wedge\ \boxed{\mathcal{U}(l_2)} \\ \langle l_2, \widehat{me}[x \mapsto \widehat{m}']\rangle \models^{\widehat{m}'} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \\ \widehat{\Phi}(l_1, \widehat{m}) \sqsubseteq \widehat{\Phi}(x, \widehat{m}') \\ \boxed{\mathcal{D}(x, \widehat{m}') \Rightarrow \mathcal{U}(l_1)} \qquad \widehat{m}' = \widehat{\mathcal{I}}(\Sigma, l, \widehat{m}) \\ \widehat{\Phi}(l_2, \widehat{m}') \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \end{cases}$

Figure 4.4: Slice CFA

The values evaluated by expressions that do not belong to the program slice do not affect the evaluation of the program. This is modeled by adding a rule that forces these evaluations to return $\star$ and by checking that the final evaluation does not return $\star$. The analysis takes advantage of this and does not generate data and flow conditions for expressions not included in the slice. This is in anticipation of the fact that these evaluations "will not be needed". Thus, the slice CFA is a demand-driven CFA, as in [Bis97a], in the same spirit of analyses that only analyze reachable code [NN97]: the slice CFA does not analyze expressions outside the computed program slice.

4.6.1 Correctness of the Slice CFA

In this section we prove that slice CFA is a safe simulation of the standard semantics and that it generates valid slices according to the slice specification. We prove the first part by relating the nodes of the standard and slice CFA derivation trees and then proving that the trees form a simulation that respects these relations. We start by relating values and environments. Given abstract cache $\widehat{\Phi}$ and abstract dependency mapping $\widehat{\mathcal{D}}$, $\mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}$ and $\mathcal{R}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}$ are defined as the smallest relations that satisfy:

$$\begin{array}{rcl} c \ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}} \ \hat{v} & \Leftrightarrow & \beta_{\mathsf{c}}(c) \sqsubseteq \Pi_{\mathsf{c}}(\hat{v}) \\ {[l,\ \rho]} \ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}} \ \hat{v} & \Leftrightarrow & \exists [l,\ \widehat{me}] \in \Pi_{\lambda}(\hat{v}),\ \forall x \in dom(\rho) \\ & & \widehat{\mathcal{D}}(x, \widehat{me}(x)) \Rightarrow \rho(x) \ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}} \ \widehat{\Phi}(x, \widehat{me}(x)) \end{array} \tag{4.18}$$

Lemma 4.6.2 *For all v, $\hat{v}$ and $\hat{v}'$, if $(v \ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}} \ \hat{v})$ and $\hat{v} \sqsubseteq \hat{v}'$ then $(v \ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}} \ \hat{v}')$*

Our decision of only analyzing expressions that are part of the slice affects the relation between the values captured inside closures. The analysis only has relevant information of values mapped to useful variables. This is also the case for variables in configurations:

$$\begin{array}{rcl} \langle l, \rho\rangle \ \mathcal{R}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}} \ \langle l, \widehat{me}\rangle & \Leftrightarrow & \forall x \in dom(\rho). \\ & & \widehat{\mathcal{D}}(x, \widehat{me}(x)) \Rightarrow \rho(x) \ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}} \ (\widehat{\Phi}(x, \widehat{me}(x))) \end{array} \tag{4.19}$$

The same can be said for nodes. We only care about values generated by configurations that are part of the slice:

$$(\Sigma \vdash \langle l, \rho \rangle \Downarrow v) \;\; \mathcal{R}_{sl}^{cfa} \;\; \langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \quad \Leftrightarrow$$

$$\mathcal{U}(l) \Rightarrow \begin{cases} \langle l, \rho \rangle \;\; \mathcal{R}_{\widehat{\Phi}, \widehat{\mathcal{D}}}^{cfa} \;\; \langle l, \widehat{me} \rangle \\ v \;\; \mathcal{V}_{\widehat{\Phi}, \widehat{\mathcal{D}}}^{cfa} \;\; \widehat{\Phi}(l, \widehat{m}) \end{cases} \tag{4.20}$$

As usual, we extend the analysis definition to configurations and associate an abstract context environment $\widehat{me}$ to a specific environment ρ by writing $\langle l, \rho \rangle \models_{\widehat{me}}^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$, defined as follows:

$$\langle l, \rho \rangle \models_{\widehat{me}}^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \quad \Leftrightarrow \quad \langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \;\wedge\; \langle l, \rho \rangle \;\; \mathcal{R}_{\widehat{\Phi}, \widehat{\mathcal{D}}}^{cfa} \;\; \langle l, \widehat{me} \rangle \tag{4.21}$$

Let $\preceq_{sl}^{cfa}$ be the simulation relation between derivation trees of the slice specification and the slice CFA defined by:

Definition 4.6.3 $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \;\; \preceq_{sl}^{cfa} \;\; [\langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})] \;\Leftrightarrow$

- $(\Sigma \vdash \langle l, \rho \rangle \Downarrow v) \;\; \mathcal{R}_{sl}^{cfa} \;\; (\langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}))$ *and*
- *If* $\mathcal{U}(l)$ *then* $\forall [\mathcal{N}] \in \mathit{reach}([\Sigma \vdash \langle l, \rho \rangle \Downarrow v])$, $\exists [\mathcal{M}] \in \mathit{reach}[\langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})]$ *such that* $[\mathcal{N}] \;\; \preceq_{sl}^{cfa} \;\; [\mathcal{M}]$

This definition is almost identical to the one defined in equation (2.9). The only difference is that the comparison between immediate subtrees is conditioned by $\mathcal{U}(l)$. We prove the following:

Lemma 4.6.4
If $[\langle l, \rho \rangle \models_{\widehat{me}}^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})]$ *and* $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v]$ *then*

$$[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \;\; \preceq_{sl}^{cfa} \;\; [\langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})]$$

Proof: *By rule induction on* $[\Sigma \vdash \langle l,\rho\rangle \Downarrow v]$. *The case where* $\neg\mathcal{U}(l)$ *trivially holds. For* $\mathcal{U}(l)$ *each rule has to be treated separately. We work though the three axioms and show two of the more important cases.*

The first three rules (axioms) generate derivation trees with no subtrees. According to definition 4.6.3, we have to show $(\Sigma \vdash \langle l,\rho\rangle \Downarrow v)\ \mathcal{R}^{cfa}_{sl}\ (\langle l,\widehat{me}\rangle \models^{\widehat{m}} (\Sigma,\widehat{\Phi},\mathcal{U},\widehat{\mathcal{D}}))$. *Since* $\mathcal{U}(l)$ *and* $\langle l,\rho\rangle\ \mathcal{R}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \langle l,\widehat{me}\rangle$ *are part of the initial conditions, it is only left to show* $(v\ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \widehat{\Phi}(l,\widehat{m}))$.

The axioms are:

(r 1.1) $\Sigma(l) = c$ *and* $v = c$.

By applying the definition of $\langle l,\widehat{me}\rangle \models^{\widehat{m}} (\Sigma,\widehat{\Phi},\mathcal{U},\widehat{\mathcal{D}})$ *to* $\Sigma(l) = c$ *and* $\mathcal{U}(l)$ *we derive* $\beta_{\mathsf{c}}(c) \sqsubseteq \widehat{\Phi}(l,\widehat{m})$ *which is precisely the condition for* $(c\ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \widehat{\Phi}(l,\widehat{m}))$ *(eq. 4.18).*

(r 1.2) $\Sigma(l) = x$, $x \in dom(\rho)$ *and* $v = \rho(x)$.

By applying the definition of $\langle l,\widehat{me}\rangle \models^{\widehat{m}} (\Sigma,\widehat{\Phi},\mathcal{U},\widehat{\mathcal{D}})$ *to* $\Sigma(l) = x$ *and* $\mathcal{U}(l)$ *we get*

$$\widehat{\Phi}(x,\widehat{me}(x)) \sqsubseteq \widehat{\Phi}(l,\widehat{m}) \quad [\mathbf{f\ 7.1}] \qquad \mathcal{D}(x,\widehat{me}(x)) \quad [\mathbf{f\ 7.2}]$$

We have:

$$\begin{array}{llll} \textit{using} & \langle l,\rho\rangle\ \mathcal{R}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \langle l,\widehat{me}\rangle \wedge [\mathbf{f\ 7.2}] & \textit{derive} & \rho(x)\ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ (\widehat{\Phi}(x,\widehat{me}(x)) \quad [\mathbf{f\ 7.3}] \\ \textit{using} & \textit{lemma 4.6.2}, [\mathbf{f\ 7.3}] \wedge [\mathbf{f\ 7.1}] & \textit{derive} & (\rho(x)\ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \widehat{\Phi}(l,\widehat{m})) \quad [\mathbf{f\ 7.4}] \end{array}$$

where $v = \rho(x)$. *Therefore* $(v\ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \widehat{\Phi}(l,\widehat{m}))$.

(r 1.3) $\Sigma(l) \in$ *Fun and* $v = [l,\ \rho]$.

By applying the definition of $\langle l,\widehat{me}\rangle \models^{\widehat{m}} (\Sigma,\widehat{\Phi},\mathcal{U},\widehat{\mathcal{D}})$ *to* $\Sigma(l) \in$ *Fun and* $\mathcal{U}(l)$ *we derive* $\{[l,\ \widehat{me}]\} \subseteq \Pi_\lambda(\widehat{\Phi}(l,\widehat{m}))$. *We need to prove* $([l,\ \rho]\ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \widehat{\Phi}(l,\widehat{m}))$, *that is,*

$$\exists [l,\ \widehat{me}'] \in \Pi_\lambda(\hat{v}),\ \forall x \in dom(\rho).\ \widehat{\mathcal{D}}(x,\widehat{me}'(x)) \Rightarrow \rho(x)\ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \widehat{\Phi}(x,\widehat{me}'(x))$$

Let $\widehat{me}$ *be the* $\widehat{me}'$ *we are looking for. Then we need to prove*

$$\forall x \in dom(\rho).\ \widehat{\mathcal{D}}(x,\widehat{me}(x)) \Rightarrow \rho(x)\ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \widehat{\Phi}(x,\widehat{me}(x))$$

which is precisely the definition of $\langle l, \rho\rangle \;\; \mathcal{R}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}} \;\; \langle l, \widehat{me}\rangle$ *in equation (4.19). Therefore,* $([l,\ \rho] \;\; \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}} \;\; \widehat{\Phi}(l, \widehat{m}))$.

The rest of the rules correspond to derivation trees of more than one node. The proof of each case is divided into two parts. In the first part we must show that for all immediate subtrees $[\Sigma \vdash \langle l', \rho'\rangle \Downarrow v']$ *of* $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$*, there is an immediate subtree* $[\langle l', \widehat{me}'\rangle \models^{\widehat{m}'} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})]$ *of* $[\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})]$ *such that*

$$[\Sigma \vdash \langle l', \rho'\rangle \Downarrow v'] \;\; \preceq^{cfa}_{sl} \;\; [\langle l', \widehat{me}'\rangle \models^{\widehat{m}'} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})]$$

These results are obtained by applying the inductive hypothesis to each $[\Sigma \vdash \langle l', \rho'\rangle \Downarrow v']$*, given a suitable* $[\langle l', \widehat{me}'\rangle \models^{\widehat{m}'} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})]$*. In order to do this we need to show* $\langle l', \rho'\rangle \;\; \mathcal{R}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}} \;\; \langle l', \widehat{me}'\rangle$.

In the second part we have to make make sure that the roots $(\Sigma \vdash \langle l, \rho\rangle \Downarrow v)$ *and* $(\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}))$ *are* $\mathcal{R}^{cfa}_{sl}$*-related. In order to do this we just need to show* $(v \;\; \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}} \;\; \widehat{\Phi}(l, \widehat{m}))$*, which completes the proof.*

All cases follow the same proof skectch. We show the `fn`*-application and one of the conditional rules:*

(r 1.7) $\quad \Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})$ *and:*

$$[\Sigma \vdash \langle l_0, \rho\rangle \Downarrow \mathit{true}] \;\wedge\; [\Sigma \vdash \langle l_1, \rho\rangle \Downarrow v]$$

By applying the definition of $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$ *to* $\Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})$ *and* $\mathcal{U}(l)$ *we derive:*

$$\begin{aligned}
&[\langle l_i, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})] \quad i = 1, 2, 3 \\
&\beta_{\mathsf{c}}(\mathit{true}) \sqsubseteq \Pi_{\mathsf{c}}(\widehat{\Phi}(l_0, \widehat{m})) \;\Rightarrow\; (\widehat{\Phi}(l_1, \widehat{m}) \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \;\wedge\; \mathcal{U}(l_1)) \quad [\mathbf{f\ 7.5}] \\
&\beta_{\mathsf{c}}(\mathit{false}) \sqsubseteq \Pi_{\mathsf{c}}(\widehat{\Phi}(l_0, \widehat{m})) \;\Rightarrow\; (\widehat{\Phi}(l_2, \widehat{m}) \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \;\wedge\; \mathcal{U}(l_2))
\end{aligned}$$

Using $\langle l_i, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$, *for* $i \in \{1,2\}$, *we apply the inductive hypothesis to* $[\Sigma \vdash \langle l_0, \rho\rangle \Downarrow \mathit{true}]$ *and* $[\Sigma \vdash \langle l_1, \rho\rangle \Downarrow v]$ *and derive*

$$\begin{array}{lll} [\Sigma \vdash \langle l_0, \rho\rangle \Downarrow \mathit{true}] \preceq^{cfa}_{sl} [\langle l_0, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})] & [\mathbf{f\ 7.6}] \\ [\Sigma \vdash \langle l_1, \rho\rangle \Downarrow v] \preceq^{cfa}_{sl} [\langle l_1, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})] & [\mathbf{f\ 7.7}] \end{array}$$

This takes care of the first part of the proof. Now we have to show $(v\ \ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \ \widehat{\Phi}(l, \widehat{m}))$. *By applying definition 4.6.3 and equation (4.20) to* [**f 7.6**] *and* $\mathcal{U}(l_0)$ *we derive*

$$(\mathit{true}\ \ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \ \widehat{\Phi}(l_0, \widehat{m}))\ \ \text{that is}\ \ \beta_{\mathsf{c}}(\mathit{true}) \sqsubseteq \Pi_{\mathsf{c}}(\widehat{\Phi}(l_0, \widehat{m}))\quad [\text{using } \mathbf{eq.\ (4.18)}]$$

Then, we can use [**f 7.5**] *to derive*

$$\mathcal{U}(l_1) \quad \wedge \quad \widehat{\Phi}(l_1, \widehat{m}) \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \quad [\mathbf{f\ 7.8}]$$

By applying definition 4.6.3 and equation (4.20) to [**f 7.7**] *and* $\mathcal{U}(l_1)$ *we derive* $(v\ \ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \ \widehat{\Phi}(l_1, \widehat{m}))$. *By applying lemma 4.6.2 to* [**f 7.8**] *and* $(v\ \ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \ \widehat{\Phi}(l_1, \widehat{m}))$ *we derive* $(v\ \ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \ \widehat{\Phi}(l, \widehat{m}))$, *which completes the proof.*

(**r 1.5**) $\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \dots t_n^{l_n})$

$$\begin{array}{rl} i \in \{1, \dots, n\}: & [\Sigma \vdash \langle l_i, \rho\rangle \Downarrow v_i] \\ & [\Sigma \vdash \langle l_0, \rho\rangle \Downarrow [l_f,\ \rho_0]] \\ & \Sigma(l_f) = (\mathtt{fn}\ x_1 \dots x_n \Rightarrow t_b^{l_b}) \\ & [\Sigma \vdash \langle l_b, \rho_0'\rangle \Downarrow v] \end{array}$$

where $\rho_0' = \rho_0[x_1 \mapsto v_1] \dots [x_n \mapsto v_n]$.

By applying the definition of $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$ *to* $\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \dots t_n^{l_n})$ *and* $\mathcal{U}(l)$ *we derive:*

$$\mathcal{U}(l_0) \quad \wedge \quad i = 0, \dots, n:\ \langle l_i, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$$

which corresponds to part of rule (**r 8.5**).

Using $\langle l_i, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$, *for* $i \in \{0, n\}$, *we apply the inductive hypothesis to* $[\Sigma \vdash \langle l_0, \rho\rangle \Downarrow [l_f,\ \rho_0]]$ *and* $[\Sigma \vdash \langle l_i, \rho\rangle \Downarrow v_i]$, $i \in \{1, \ldots, n\}$, *and derive:*

$$[\Sigma \vdash \langle l_0, \rho\rangle \Downarrow [l_f,\ \rho_0]] \ \preceq^{cfa}_{sl}\ [\langle l_0, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})] \quad [\mathbf{f\ 7.9}]$$

$$i \in \{1, \ldots, n\} \quad [\Sigma \vdash \langle l_i, \rho\rangle \Downarrow v_i] \ \preceq^{cfa}_{sl}\ [\langle l_i, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})] \quad [\mathbf{f\ 7.10}]$$

By applying definition 4.6.3 and equation (4.20) to [**f 7.9**] *and* $\mathcal{U}(l_0)$ *we derive* $([l_f,\ \rho_0]\ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \widehat{\Phi}(l_0, \widehat{m}))$ *which, according to equation (4.18), implies:*

$$\begin{aligned}&\exists [l_f,\ \widehat{me}_0] \in \Pi_\lambda(\widehat{\Phi}(l_0, \widehat{m})),\ \forall x \in dom(\rho_0).\\ &\quad \widehat{\mathcal{D}}(x, \widehat{me}_0(x)) \Rightarrow \rho_0(x)\ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \widehat{\Phi}(x, \widehat{me}_0(x)) \quad [\mathbf{f\ 7.11}]\end{aligned}$$

Using this $[l_f,\ \widehat{me}_0] \in \Pi_\lambda(\widehat{\Phi}(l_0, \widehat{m}))$ *and the definition of* $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$ *for* $\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n})$ *and* $\mathcal{U}(l)$ *- the rest of* (**r 8.5**) *- we get:*

$$\begin{aligned}&\langle l_b, \widehat{me}'_0\rangle \models^{\widehat{m}'} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \ \wedge\ \mathcal{U}(l_b)\\ &i = 1, \ldots, n : \begin{cases}\widehat{\Phi}(l_i, \widehat{m}) \sqsubseteq \widehat{\Phi}(x_i, \widehat{m}') & [\mathbf{f\ 7.12}]\\ \mathcal{D}(x_i, \widehat{m}') \Rightarrow \mathcal{U}(l_i) & [\mathbf{f\ 7.13}]\end{cases}\\ &\widehat{\Phi}(l_b, \widehat{m}') \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \quad [\mathbf{f\ 7.14}]\end{aligned}$$

where $\widehat{m}' = \widehat{\mathcal{I}}(\Sigma, l, \widehat{m}, l_f)$ *and* $\widehat{me}'_0 = \widehat{me}_0[x_i \mapsto \widehat{m}']$.

We want to apply the inductive hypothesis to $[\Sigma \vdash \langle l_b, \rho'_0\rangle \Downarrow v]$. *In order to do this we need to prove* $\langle l_b, \rho'_0\rangle\ \mathcal{R}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \langle l_b, \widehat{me}'_0\rangle$, *that is:*

$$\forall x \in dom(\rho'_0).\ \widehat{\mathcal{D}}(x, \widehat{me}'_0(x)) \Rightarrow \rho'_0(x)\ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ (\widehat{\Phi}(x, \widehat{me}'_0(x))$$

We have two cases. Given $x \in dom(\rho'_0)$*:*

- $x \neq x_i$, *for all* $i \in \{1, \ldots, n\}$. *Then* $\rho'_0(x) = \rho_0(x)$ *and* $\widehat{me}'_0(x) = \widehat{me}_0(x)$. *Therefore* $\widehat{\mathcal{D}}(x, \widehat{me}'_0(x))$ *can be written as* $\widehat{\mathcal{D}}(x, \widehat{me}_0(x))$ *which, using* [**f 7.11**], *triggers* $\rho_0(x)\ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \widehat{\Phi}(x, \widehat{me}_0(x))$, *i.e.,* $\rho'_0(x)\ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ \widehat{\Phi}(x, \widehat{me}'_0(x))$.
- $x = x_i$, *for some* $i \in \{1, \ldots, n\}$. *Then* $\rho'_0(x) = v_i$ *and* $\widehat{me}'_0(x) = \widehat{m}'$. *We need to prove:*

$$\widehat{\mathcal{D}}(x_i, \widehat{m}') \Rightarrow v_i\ \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}}\ (\widehat{\Phi}(x_i, \widehat{m}')$$

*Using $\widehat{\mathcal{D}}(x_i, \widehat{m}')$ and [**f 7.13**] we get $\mathcal{U}(l_i)$. By applying definition 4.6.3 and equation (4.20) to [**f 7.10**] and $\mathcal{U}(l_i)$ we derive $(v_i \;\; \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}} \;\; \widehat{\Phi}(l_i, \widehat{m}))$.*
*By applying lemma 4.6.2 to $(v_i \;\; \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}} \;\; \widehat{\Phi}(l_i, \widehat{m}))$ and [**f 7.12**] we finally derive $(v_i \;\; \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}} \;\; (\widehat{\Phi}(x_i, \widehat{m}'))$.*

Now we can apply the inductive hypothesis to $[\Sigma \vdash \langle l_b, \rho'_0\rangle \Downarrow v]$ and derive:

$$[\Sigma \vdash \langle l_b, \rho'_0\rangle \Downarrow v] \;\; \preceq^{cfa}_{sl} \;\; [\langle l_b, \widehat{me}'_0\rangle \models^{\widehat{m}'} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})] \quad [\mathbf{f\ 7.15}]$$

This takes care of the first part of the proof: all subtrees of $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$ have been matched. Now we have show $(v \;\; \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}} \;\; \widehat{\Phi}(l, \widehat{m}))$.
*By applying definition 4.6.3 and equation (4.20) to [**f 7.15**] and $\mathcal{U}(l_b)$ we derive $(v \;\; \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}} \;\; \widehat{\Phi}(l_b, \widehat{m}'))$. Finally, using lemma 4.6.2, $(v \;\; \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}} \;\; \widehat{\Phi}(l_b, \widehat{m}'))$ and [**f 7.14**] we derive $(v \;\; \mathcal{V}^{cfa}_{\widehat{\Phi},\widehat{\mathcal{D}}} \;\; \widehat{\Phi}(l, \widehat{m}))$. This completes the proof.*

[**QED**]

Every node in $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$ that corresponds to an expression marked as part of the slice $\mathcal{U}$ is simulated by a node in $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$. This simulation enables us to prove that the results of the analysis satisfy the slice specification:

Lemma 4.6.5

If $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \;\; \preceq^{cfa}_{sl} \;\; [\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})]$ then

$$\exists \mathcal{D}.\ [\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \models_{sl} (\mathcal{U}, \mathcal{D})$$

Proof Sketch:

Given $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$, we can build $\mathcal{D}$ such that for all reachable subtrees $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$ inside the slice where $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \;\; \preceq^{cfa}_{sl} \;\; [\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})]$:

$$x \in \mathcal{D}(\langle l, \rho\rangle) \;\Leftrightarrow\; \widehat{\mathcal{D}}(x, \widehat{me}(x))$$

We must prove that, given the above conditions, $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \models^{loc}_{sl} (\mathcal{U}, \mathcal{D})$ holds.

[**QED**]

Now we are ready to prove that a solution $\mathcal{U}$ to the slice CFA is a valid slice:

Theorem 4.6.6 (Correctness of the Slice CFA)

If $\Sigma \vdash \langle l, \rho \rangle \Downarrow v$, $\langle l, \rho \rangle \models^{\widehat{m}}_{\widehat{me}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$ *and* $\mathcal{U}(l)$ *then* $Slice(\Sigma, \mathcal{U}, \langle l, \rho \rangle)$.

Proof:

By applying lemma 4.6.4 to the initial conditions and then immediately applying lemma 4.6.5 to the result, we derive $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \models_{sl} (\mathcal{U}, \mathcal{D})$ *for some* $\mathcal{D}$*. By applying theorem 4.5.6 to* $\mathcal{U}(l)$ *and* $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \models_{sl} (\mathcal{U}, \mathcal{D})$ *we derive* $Slice(\Sigma, \mathcal{U}, \langle l, \rho \rangle)$

[**QED**]

4.7 Correctness of the Transformation with respect to the Analysis

We prove that the analysis provides correct information to the transformation. The correctness of the transformation is stated for programs that evaluate to constants. If an initial configuration of a closed program evaluates to a constant, then the transformed program evaluates to the same constant.

Theorem 4.7.1 (Correctness of the Transformation)

Given closed program $\Sigma(l_p) = P$*, if* $\langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$, $\Sigma \vdash \langle l_p, [\,] \rangle \Downarrow c$ *and* $\mathcal{U}(l_p)$ *then* $\mathcal{T}^d_{\mathcal{U}}(\Sigma) \vdash \langle l_p, [\,] \rangle \Downarrow c$.

Proof:

$\langle l, [\,] \rangle \models^{\widehat{m}}_{\widehat{me}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$	*using* $\langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$ *and equation (4.21).*
$Slice(\Sigma, \widehat{\Phi}, \langle l, [\,] \rangle)$	*theorem 4.6.6 and* $\mathcal{U}(l_p)$.
$\mathcal{T}^d_{\mathcal{U}}(\Sigma) \vdash \langle l_p, [\,] \rangle \Downarrow c$	*theorem 4.4.2 and* $\Sigma \vdash \langle l_p, [\,] \rangle \Downarrow c$.

[**QED**]

Furthermore, the information computed by the analysis is still valid - as expected - for the transformed program:

Theorem 4.7.2 *If* $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$ *then* $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\mathcal{T}^d_{\mathcal{U}}(\Sigma), \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$.

Proof: *By co-induction on* $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\mathcal{T}^d_{\mathcal{U}}(\Sigma), \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$.

4.8 Slice 0CFA

The judgement $\langle l, \widehat{me}\rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$ together with the rules in Figure 4.4 provide a framework for the definition of a series of control and data flow analyses that compute valid program slices. A new analysis is determined by the choice of abstract contexts $\widehat{Context}$, its instantiators and the domain of abstract constants $\widehat{Const}$. In this section we apply the abstractions used for the traditional 0CFA and derive the Slice 0CFA .

A Slice 0CFA is generated by abstracting all binding contexts to a single context $\bullet$ and all constants to a single value $\bullet_c$. As it was shown in section 3.3, the control and data flow elements of the analysis can be simplified to the following domains:

$$\begin{array}{rcl} \widehat{\Phi} \in \widehat{Cache} & = & (Lab \cup Var) \rightarrow \widehat{Val} \\ \widehat{Val} & = & \widehat{Closure} \\ \widehat{Closure} & = & \mathcal{P}(Lab) \end{array}$$

Furthermore, the mapping $\widehat{\mathcal{D}}$ - initially defined for pairs of abstract contexts and variables - can be simplified to the following definition:

$$\widehat{\mathcal{D}} \in Var \rightarrow Bool$$

Thus, instead of writing $\widehat{\mathcal{D}}(x, \bullet)$ we write $\widehat{\mathcal{D}}(x)$ to indicate that x - all bindings of x - may have been used by the abstract evaluation of the slice determined by $\mathcal{U}$.

$$l \models^{\bullet} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \quad \Leftrightarrow \quad \mathcal{U}(l) \Rightarrow \ldots$$

$$(\mathbf{r\ 9.1}) \quad \Sigma(l) = c \Rightarrow \mathit{true}$$

$$(\mathbf{r\ 9.2}) \quad \Sigma(l) = x \Rightarrow \widehat{\Phi}(x) \subseteq \widehat{\Phi}(l) \ \wedge \ \widehat{\mathcal{D}}(x)$$

$$(\mathbf{r\ 9.3}) \quad \Sigma(l) \in \mathit{Fun} \Rightarrow l \in \widehat{\Phi}(l)$$

$$(\mathbf{r\ 9.4}) \quad \Sigma(l) = (p\ t_1^{l_1} \ldots t_n^{l_n}) \Rightarrow i = 1, \ldots, n : \begin{cases} l_i \models^{\bullet} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \\ \mathcal{U}(l_i) \end{cases}$$

$$(\mathbf{r\ 9.5}) \quad \Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n}) \Rightarrow \begin{cases} \mathcal{U}(l_0) \\ l_i \models^{\bullet} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \quad i = 0, \ldots, n \\ \forall l_f \in \widehat{\Phi}(l_0) : \\ \quad \begin{cases} (\Sigma(l_f) = (\mathtt{fn}\, x_1 \ldots x_n \Rightarrow t_b^{l_b}) \vee \\ \ \Sigma(l_f) = (\mathtt{fun}\, y\, x_1 \ldots x_n \Rightarrow t_b^{l_b})) \Rightarrow \\ \quad \begin{cases} l_b \models^{\bullet} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \ \wedge \ \mathcal{U}(l_b) \\ i = 1, \ldots, n : \begin{cases} \widehat{\Phi}(l_i) \subseteq \widehat{\Phi}(x_i) \\ \widehat{\mathcal{D}}(x_i) \Rightarrow \mathcal{U}(l_i) \end{cases} \\ \widehat{\Phi}(l_f) \subseteq \widehat{\Phi}(l) \end{cases} \\ \Sigma(l_f) = (\mathtt{fun}\, y\, x_1 \ldots x_n \Rightarrow t_f^{l_f}) \Rightarrow l_f \in \widehat{\Phi}(y) \end{cases} \end{cases}$$

$$(\mathbf{r\ 9.6}) \quad \Sigma(l) = (\mathtt{if}\ t_0^{l_0}\ \mathtt{then}\ t_1^{l_1}\ \mathtt{else}\ t_2^{l_2}) \Rightarrow \begin{cases} l_i \models^{\bullet} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \ \wedge \ \mathcal{U}(l_i) \quad i \in \{0,1,2\} \\ \widehat{\Phi}(l_1) \subseteq \widehat{\Phi}(l) \\ \widehat{\Phi}(l_2) \subseteq \widehat{\Phi}(l) \end{cases}$$

$$(\mathbf{r\ 9.7}) \quad \Sigma(l) = (\mathtt{let}\ x = t_1^{l_1}\ \mathtt{in}\ t_2^{l_2}) \Rightarrow \begin{cases} l_i \models^{\bullet} (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \quad i \in \{1,2\} \\ \widehat{\Phi}(l_1) \subseteq \widehat{\Phi}(x) \\ \widehat{\Phi}(l_2) \subseteq \widehat{\Phi}(l) \\ \mathcal{U}(l_2) \ \wedge \ \widehat{\mathcal{D}}(x) \Rightarrow \mathcal{U}(l_1) \end{cases}$$

Figure 4.5: 0CFA Slice Analysis

A Slice 0CFA is defined as the greatest relation that satisfies the rules of Figure 4.5. The judgement $l \models^\bullet (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$ claims that $\widehat{\Phi}$ represents a consistent description of the control flow during the evaluation of l under universe Σ - and environments consistent with $\widehat{\Phi}$ - but only circumscribed to the slice defined by $\mathcal{U}$. In other words, $\widehat{\Phi}$ contains correct flow information of the evaluation of the slice defined by $\mathcal{U}$.

The $\langle l, \rho \rangle \models^\bullet (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$ relation relates a configuration to a particular instance of the analysis. The relation makes sure that $\widehat{\Phi}$ provides a consistent description of the values mapped to the variables considered as useful by the analysis. The consistency relation is defined as follows:

$$\begin{array}{rcl}
\langle l, \rho \rangle \models^\bullet (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) & \Leftrightarrow & l \models^\bullet (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \ \wedge \ \rho \models^\bullet (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) \\
\rho \models^\bullet (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}}) & \Leftrightarrow & \forall x \in dom(\rho) \ \wedge \ \widehat{\mathcal{D}}(x). \\
& & \qquad \qquad \rho(x) = c \qquad \Rightarrow \quad true \\
& & \qquad \qquad \rho(x) = [l, \ \rho'] \ \ \Rightarrow \quad (l \in \widehat{\Phi}(x)) \ \wedge \ \rho' \models^\bullet (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})
\end{array} \tag{4.22}$$

Since the slice 0CFA is an instantiation of the standard slice CFA, the same theorems proved in section 4.6 apply. In particular,

Theorem 4.8.1 (Correctness of the Slice 0CFA)
If $\Sigma \vdash \langle l, \rho \rangle \Downarrow v$, $\langle l, \rho \rangle \models^\bullet (\Sigma, \widehat{\Phi}, \mathcal{U}, \widehat{\mathcal{D}})$ *and* $\mathcal{U}(l)$ *then* $Slice(\Sigma, \mathcal{U}, \langle l, \rho \rangle)$.

4.8.1 Relation with [WS99]

In [WS99] we defined a constraint system that identified the useless expressions of the program. This was achieved by:

- Computing the 0CFA of the program.
- Computing the dependency sets of each analyzed expression. A dependency set $\mathcal{D}(l)$ contains the variables on which the evaluation of expression l may depend on.

- The same reasoning to identify useful arguments was used: If the body of an applied body depends on its i-th formal parameter then the i-th operand must be useful (part of the slice).

The constraint system in [WS99] is very similar to our slice 0CFA, the latter being an improved version of the former. The new analysis deals with program slices explicitly by dedicating the mapping $\mathcal{U}$ to keep track of the useful expressions. The information of useful variables compiled by $\mathcal{D}$ is the same information contained in the dependency sets $\widehat{\mathcal{D}}$ of program bodies. The slice 0CFA has the advantage of not having to compute the dependency sets for each the expressions since it is only the information of the body that we are looking for. It is also important to point out that the new analysis only analyzes the expressions proved to be part of a valid slice.

4.9 Previous and Related Work

The transformation and analysis defined in this chapter were initially motivated by the specification of useless-variable elimination provided by Shivers in [Shi91] as an application of control flow analysis Our initial results were reported in [WS99]. We later found out that the problem that we were dealing with - the transformation presented in this chapter is a less radical version of our initial transformation - was an application of the concepts of program slicing to the domain of higher-order languages. Hence, most of the terminology used in this chapter comes from the field of program slicing. Closely related concepts are those of program dependency and dead-code.

Program slicing [Wei84, HRB84] deals with the identification of the elements of the program that do not contribute to its final value. Slicing algorithms use information gathered by data and control flow analyses [Wei84], or system dependence graphs [HRB84]. Bodík and Gupta [BG85] use slicing transformations to eliminate partial dead code, *i.e.*, code which is only dead on some program paths [KRS94].

Most program slicing research have been performed in the context of imperative languages. Our treatment of program slices is closely related to the work of Biswas [Bis97b]. We have also found an interesting application of concepts of partial evaluation to the definition of program slices in [Hug88]. We point out the following:

- Biswas [Bis97b] provides a definition of dynamic slices for higher-order languages using a non-standard natural semantics. The evaluation of a term returns a pair (V, L) where L is the set of labels of terms that contribute to the evaluation of V. In an application, the set L_i associated to a formal parameter is only added to the total when the variable it is bound to is accessed during the evaluation of the body. This is similar to the conditions we have for useful variables. The semantics computes the minimum dynamic slice and its relation with called-by-name evaluation is formalized.

  The analysis used to compute approximations of the minimum slice is based on 0CFA. The analysis resembles our slice 0CFA and takes advantage of the same properties: terms that are not part of the dynamic slice need not be analyzed. The results are presented in [Bis97a]. We provide a more general specification of how to validate (and compute) valid program slices, and use it to derive a standard slice CFA. We take these results a step further and present a more radical transformation in Chapter 5.

- Reps and Turnidge [RT96] also develop the notion of slices of functional programs and compare it with specialization operations performed by partial evaluation. A slicing criterion is defined as a composition of projection functions [Hug88] that indicate what parts of the output are of interest. The projection function (slicing criterion), defined by a regular tree grammar, is pushed backwards through the nodes of the syntax tree of the program - as instructed by the analysis - generating the correspondent projection function that, when applied to the program, generates a valid program slice. Slices are also defined by the substitution of expressions of a program with a dummy element. The meaning of the resulting program must be equivalent to the composition of the slicing criterion with the meaning of the original program. The use of projection functions permits the slicing of complex heap-allocated data structures. Unfortunately, the relation between the semantics of the program and its slice, and their interaction with projection functions, is not considered.

Our transformation falls into the category of optimizations known as dead-code elimination, useless variable elimination and unreachable-code elimination, which are described in detail for first-order languages by most standard compiler text books [ASU86, Muc97, App98].

Our definition of dead-code (expressions outside the slice) comes closest to the one given by Muchnick where dead-code is defined as code that is executed but that has no effect on the result of the computation being performed [Muc97]. Our transformation also replaces unreachable-code : unused branches or functions that are never applied.

The notion of program dependency is closely related to ideas presented in this chapter. Abadi *et al* [ALL96] define a dependency analysis for caching optimizations in a similar spirit to ours: they are also concerned in what parts of the program *may* affect the final result. Their analysis is based on the labelled λ-calculus. The labelled λ-calculus deals with labelled terms terms and contains extra reduction rules whose main task is to move - and concatenate - labels outwards as little as possible in order to permit β reduction. A term reduces to its normal form labelled with the terms that were involved in the reduction. Terms not present in the list can be replaced by holes (we use $\bullet$) and still reduce to the same normal form. Hence, the set of labels determines a program slice.

Dependency information is used to group equivalent terms in a single cache entry. They do not use data or flow analysis results, but rather rely on previous, actual executions of programs. They also point out the similarity between dependency analysis and strictness analysis [BHA86, GVS95], which is concerned with what parts of the program *must* be evaluated.

Abadi et.al [ABHR99] define the Dependency Core Calculus, a small extension to Moggi's computational lambda calculus. This calculus is used to derive type-based secure information flow, binding-time, slicing, and call-tracking analyses. The notion of dependency is captured in a type-based setting.

Chapter 5

Removing Useless Variables and Expressions

5.1 Motivation: Removing Useless Variables and Expressions

A useless variable is one whose value contributes nothing to the final outcome of a computation [Shi91]. Useless variable elimination is a transformation that eliminates useless parameters from procedures and the corresponding actual parameters from their call sites.

Consider the following example:

```
let fun loop(a, bogus, j) =                 let fun loop (a, j) =
     if (j > 100)                                if (j > 100)
         then a                        ==>           then a
     else loop(f(a,j), bogus+2, j+1)             else loop (f(a,j), j+1)
in loop (a, 3, 1)                           in loop (a, 1)
end                                         end
```

It is clear in the program on the left-hand side that the second argument does not contribute to the value of the computation. Our transformation replaces `loop` by the evident two-argument procedure, as shown in the right-hand side.

Our analysis also identifies useless variables in higher-order procedures. For example, consider:

```
let fun f1 (x, y) = x                         let fun f1 (x) = x
    fun f2 (x, y) = x+x             ==>           fun f2 (x) = x+x
    fun f3 (x, y) = y                             val g = if p then f1 else f2
    val g = if p then f1 else f2              in g(x)
    val h = if q then f1 else f3
in g(x,h)
```

`f1` and `f2` are the only functions that flow to the application site `g(x,h)`. These functions do not use their second argument and they do not flow to any other calling points. Hence, the second actual parameter can be removed from the application site and both functions can be replaced by their one-argument versions. We observe that `h` and `f3` do not contribute to the value of the computation. We can remove them from the initial program, getting the second program.

Shivers has presented such a transformation as an example of the optimizations that can be performed using control-flow analysis information. In [WS99] we reformulate the transformation and prove its correctness. In this chapter we present the transformation as an instance of program slicing; we call it a removable program slice. We provide a specification for validating removable slices and use it to derive a family of control and data flow analysis that compute valid removable slices. The correctness of the transformation is ensured by proving that 1) the information computed by the analyses satisfies the conditions set up by the specification, and 2) the information is correct with respect to the specification.

$$\mathcal{T}_{\mathcal{U}}(c^l) = c^l$$

$$\mathcal{T}_{\mathcal{U}}(x^l) = x^l$$

$$\mathcal{T}_{\mathcal{U}}((\texttt{fn}\, x_1 \ldots x_n \Rightarrow e_b)^l) = (\texttt{fn}\, x_{i_1} \ldots x_{i_m} \Rightarrow \mathcal{T}_{\mathcal{U}}(e_b))^l, \text{ where } x_{i_1}{:}\ldots{:}x_{i_m} = (x_1{:}\ldots{:}x_n)\backslash\mathcal{U}$$

$$\mathcal{T}_{\mathcal{U}}((\texttt{fun}\, y\ x_1 \ldots x_n \Rightarrow e_b)^l) = \begin{cases} (\texttt{fn}\, x_{i_1} \ldots x_{i_m} \Rightarrow \mathcal{T}_{\mathcal{U}}(e_b))^l & \text{if } \mathcal{U}(y) \\ (\texttt{fun}\, y\ x_{i_1} \ldots x_{i_m} \Rightarrow \mathcal{T}_{\mathcal{U}}(e_b))^l & \text{otherwise} \end{cases} \text{ where } x_{i_1}{:}\ldots{:}x_{i_m} = (x_1{:}\ldots{:}x_n)\backslash\mathcal{U}$$

$$\mathcal{T}_{\mathcal{U}}((p\ e_1 \ldots e_n)^l) = (p\ \ \mathcal{T}_{\mathcal{U}}(e_1)\ \ \ldots\ \ \mathcal{T}_{\mathcal{U}}(e_n))^l$$

$$\mathcal{T}_{\mathcal{U}}((t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n})^l) = (\mathcal{T}_{\mathcal{U}}(t_0^{l_0})\ \ \mathcal{T}_{\mathcal{U}}(t_{i_1}^{l_{i_1}})\ \ \ldots\ \ \mathcal{T}_{\mathcal{U}}(t_{i_m}^{l_{i_m}}))^l, \text{ where } \mathcal{U}(l_{i_j})$$

$$\mathcal{T}_{\mathcal{U}}((\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})^l) =$$

$$\begin{cases} (\texttt{if}\ \mathcal{T}_{\mathcal{U}}(t_0^{l_0})\ \texttt{then}\ \mathcal{T}_{\mathcal{U}}(t_1^{l_1})\ \texttt{else}\ \mathcal{T}_{\mathcal{U}}(t_2^{l_2}))^l & \text{if } \mathcal{U}(l_1) \wedge \mathcal{U}(l_2) \\ \mathcal{T}_{\mathcal{U}}(t_i^{l_i}) & \text{if } \mathcal{U}(l_i) \quad i = 1, 2 \end{cases}$$

$$\mathcal{T}_{\mathcal{U}}((\texttt{let}\ x = t_1^{l_1}\ \texttt{in}\ t_2^{l_2})^l) =$$

$$\begin{cases} (\texttt{let}\ x = \mathcal{T}_{\mathcal{U}}(t_1^{l_1})\ \texttt{in}\ \mathcal{T}_{\mathcal{U}}(t_2^{l_2}))^l & \text{if } \mathcal{U}(l_1) \\ \mathcal{T}_{\mathcal{U}}(t_2^{l_2}) & \text{otherwise} \end{cases}$$

Figure 5.1: Transformation: Removing Useless Variables and Expressions

5.2 The Transformation

Let's consider the mapping $\mathcal{U}$ from program labels and variables to booleans. The transformation removes from a program the expressions and variables marked as useless by $\mathcal{U}$. The transformation takes a term e and returns a new term $\mathcal{T}_{\mathcal{U}}(e)$ following the rules in Figure 5.1. According to the rules, a transformed expression may have a label different from the original.

The transformation removes all expressions and variables not marked by $\mathcal{U}$. In the case of applications, useless argument expressions are removed. Similarly, useless formal parameters are removed from the respective function definitions. If a recursive function `fun` is not used as such then it is transformed to its non-recursive version `fn`. In the case of `let`-expressions, the removal of the left-hand side of the assignment implies the removal of the binding variable. `if`-expressions with a useless branch are replaced by their useful branch.

The transformation of a universe Σ given $\mathcal{U}$ is defined by:

$$\mathcal{T}_{\mathcal{U}}(\Sigma) = \{\ (l, t^l) \mid \exists l' \in dom(\Sigma) \cap \mathcal{U}.\ t^l = \mathcal{T}_{\mathcal{U}}(\Sigma(l'))\ \} \tag{5.1}$$

The transformation keeps a subset of the labels of the original program. Labels not marked by $\mathcal{U}$ are always removed. In some cases, labels of useful expressions are removed too. For example, in the case of a `let`-expression labeled with l, the transformation may replace the whole expression by the transformed version of its body and, consequently, the label l is lost.

Let's define[1] $\mathcal{T}_{\Sigma,\mathcal{U}}(l) = l'$ where $\mathcal{T}_{\mathcal{U}}(\Sigma(l)) = t^{l'}$. Now we extend the transformation to configurations and values:

$$\begin{array}{rcl} \mathcal{T}_{\mathcal{U}}(\langle l, \rho\rangle) & = & \langle \mathcal{T}_{\mathcal{U}}(l), \mathcal{T}_{\mathcal{U}}(\rho)\rangle \\ \mathcal{T}_{\mathcal{U}}(c) & = & c \\ \mathcal{T}_{\mathcal{U}}([l,\ \rho]) & = & [\mathcal{T}_{\mathcal{U}}(l),\ \mathcal{T}_{\mathcal{U}}(\rho)] \end{array} \tag{5.2}$$

where useless variables are removed from environments:

$$\mathcal{T}_{\mathcal{U}}(\rho) = \{(x, \mathcal{T}_{\mathcal{U}}(\rho(x))) \mid x \in dom(\rho) \cap \mathcal{U}\}$$

[1] We'll write $\mathcal{T}_{\mathcal{U}}$ when Σ is clear from the context.

5.3 The Transformation and Program Slices

The concept of useless variable, as defined in section 5.1, is closely related to the notions of dependency and program slices developed in Chapter 4. Expressions outside the program slice do not contribute to the final value of the program and are deemed as useless. Similarly, variables that never bind to values generated by the slice do not contribute to the final value of the program and, therefore, are useless variables.

Our specification of program slices deals with the notion of useful variables: it reasons about the variables that are useful in a particular configuration. Can we use this information and say that variables that are never used by the program slice are useless and, therefore, can be removed? Or, given an application term, can we remove the formal parameters not included in the program slice, as long as we remove the variables from the respective functions as indicated in the previous statement?

The answer to these questions is: not always. Consider the following example:

```
let fun f1 (x, y) = x    (* y is useless *)
    fun f2 (x, y) = x+x  (* y is useless *)
    fun f3 (x, y) = y    (* y is useful  *)
    fun g (gg) = gg(e1,e2)
    fun h (hh) = hh(e3,e4)
in (g(f1) + g(f2) + h(f2) + h(f3))
```

Both `f1` and `f2` depend only on their first argument. Given that `gg` can only evaluate to `f1` or `f2`, we can conclude that `gg(e1,e2)` also depends only on its first argument. It may seem plausible then to remove the second argument from `f1` and `f2`, and to remove the second actual parameter from `gg(e1,e2)`.

Unfortunately this can't be done. Observe that `f2` and `f3` flow to `hh(e3,e4)` and, since `f3` depends on its second argument, neither the second actual parameter nor the second argument of `f2` and `f3` can be removed. This means that `f2` depends on its second argument too and the transformation suggested above cannot be performed. Dependency information should be contagious.

5.3.1 Removable Program Slices

A program slice, as defined in chapter 4, does not ensure that the removal of useless arguments and expressions will succeed. The definition of program slice only ensures that expressions outside a slice can be replaced by any other terminating expression without affecting the final value of the program. The removal of these expressions, and the matching formal parameters from the respective function's parameter lists, introduces new problems. These complications motivated the re-formulation of the concept of slices used in imperative languages - where the removal of statements is a simpler operation - to the one used in chapter 4 tailored to functional programming languages.

We reconsider the initial formulation of slicing. We define a *removable* slice as a set of program labels and variables that can be extracted from the initial program[2] without changing its original meaning. In other words, if the expressions and variables outside the removable slice are removed, the resulting program - the slice - evaluates to the same value.

A removable slice is a mapping (or set) $\mathcal{U}$ of program labels and variables to booleans defined by:

Definition 5.3.1 (Removable Program Slices)

$$\mathit{Slice}_{rm}(\Sigma, \mathcal{U}, \mathcal{C}) \quad \Leftrightarrow \quad \exists\, v, v'.\ \Sigma \vdash \mathcal{C} \Downarrow v \ \wedge\ \mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \mathcal{T}_{\mathcal{U}}(\mathcal{C}) \Downarrow v' \ \wedge\ v' = \mathcal{T}_{\mathcal{U}}(v)$$

[2]This notion seems to fit more accurately the meaning of the word slice.

5.3.2 Relating *Slice* and $Slice_{rm}$

A removable slice is a standard slice with extra constraints that make sure that the removal of formal parameters and arguments is consistent. Let's consider the same example shown before:

```
let fun f1 (x, y) = x    (* y is useless *)
    fun f2 (x, y) = x+x  (* y is useless *)
    fun f3 (x, y) = y    (* y is useful  *)
    fun g (gg) = gg(e1,e2)
    fun h (hh) = hh(e3,e4)
in (g(f1) + g(f2) + h(f2) + h(f3))
```

This code corresponds to example 3 in section 4.5.1 where `e2` was deemed as useless. The slice conditions fail to ensure a correct transformation. As we have seen, `e2` cannot be removed and therefore is part of every slice. Even though the second argument of `f1` and `f2` is useless (in their respective bodies), it cannot be removed from the parameter list. This is due to the fact that `f2` also flows to `hh(e3,e4)` where the second argument is needed by `f3`. At the moment, information about useful variables is passed in one direction: from functions to application sites. The problem is solved by making this flow bi-directional so information about useful parameters is shared across call sites. In this particular example, `f2` must inform `hh(e3,e4)` that the first argument is useful, which in turn must inform the same to `f3`. Similarly, `f3` must inform `hh(e3,e4)` that the second argument is useful, which in turn must inform the same to `f2`. As a result, both formal parameters of `f2` are useful and cannot be removed.

Something similar occurs with:

```
let f = fn(x,y,z) = if (x < 0) then y else z
in
```

```
.... (f -1 a b)
.....(f 5 c d)
```

where, even though `b` and `c` do not belong to the standard slice, they cannot be removed from the application.

Therefore, all functions flowing to the same call site must have the same information about useful parameters (position-wise, *i.e.*, first, second, etc). We want this to be true for all reachable application sites in $[(\Sigma,\mathcal{U}) \vdash \mathcal{C} \Downarrow^\star \dot{v}]$. These conditions are enforced by $\mathit{Params}([(\Sigma,\mathcal{U}) \vdash \mathcal{C} \Downarrow^\star \dot{v}])$ defined below:

$$\begin{array}{l} \mathit{Params}([(\Sigma,\mathcal{U}) \vdash \mathcal{C} \Downarrow^\star \dot{v}]) \;\Leftrightarrow \\ \quad \forall \langle l,\rho\rangle \in \mathit{reach}^*([(\Sigma,\mathcal{U}) \vdash \mathcal{C} \Downarrow^\star \dot{v}]) : \\ \qquad \left\{ \begin{array}{l} \Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n}) \wedge \mathcal{U}(l) \\ \Sigma \vdash \langle l_0,\rho\rangle \Downarrow [l_f,\ \dot{\rho}_0] \\ \mathit{formals}(\Sigma, l_f) = x_1{:}\ldots{:}x_n \end{array} \right\} \Rightarrow \begin{array}{c} \forall i = 1,\ldots,n : \\ \mathcal{U}(x_i) \;\Leftrightarrow\; \mathcal{U}(l_i) \end{array} \end{array} \tag{5.3}$$

Put together with the slice conditions we want to prove the following:

$$\Sigma \vdash \mathcal{C} \Downarrow v \;\wedge\; \left\{ \begin{array}{l} (\Sigma,\mathcal{U}) \vdash \mathcal{C} \Downarrow^\star \dot{v} \;\wedge\; (\dot{v} \neq \star) \\ \mathit{Params}([(\Sigma,\mathcal{U}) \vdash \mathcal{C} \Downarrow^\star \dot{v}]) \end{array} \right\} \quad \Rightarrow \quad \mathit{Slice}_{rm}(\Sigma,\mathcal{U},\mathcal{C}) \tag{5.4}$$

that is

$$\mathit{Slice}(\mathcal{C},\Sigma,\mathcal{U}) \quad \wedge \quad \mathit{Params}([(\Sigma,\mathcal{U}) \vdash \mathcal{C} \Downarrow^\star \dot{v}]) \quad \Rightarrow \quad \mathit{Slice}_{rm}(\mathcal{C},\Sigma,\mathcal{U})$$

We use the same techniques applied so far: we build a simulation that relates the derivation trees of $(\Sigma,\mathcal{U}) \vdash \mathcal{C} \Downarrow^\star \dot{v}$ and $\mathcal{T}_\mathcal{U}(\Sigma) \vdash \mathcal{T}_\mathcal{U}(\mathcal{C}) \Downarrow \mathcal{T}_\mathcal{U}(\dot{v})$. The simulation is based on the following relation:

$$(\Sigma' \vdash \mathcal{C} \Downarrow v) \;\; \mathcal{R}^{tr}_{rm} \;\; ((\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^\star \dot{v}) \quad \Leftrightarrow \quad \begin{cases} \Sigma' = \mathcal{T}_\mathcal{U}(\Sigma) \\ \mathcal{C} = \mathcal{T}_\mathcal{U}(\dot{\mathcal{C}}) \\ v = \mathcal{T}_\mathcal{U}(\dot{v}) \end{cases} \tag{5.5}$$

Let's define $\preceq^{tr}_{rm}$ as the simulation induced by $\mathcal{R}^{tr}_{rm}$. The following theorem says that the derivation trees of $\mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \mathcal{T}_{\mathcal{U}}(\dot{\mathcal{C}}) \Downarrow v$ and $(\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^{\star} \dot{v}$ form a $\mathcal{R}^{tr}_{rm}$-simulation, as long as $\mathit{Params}([(\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^{\star} \dot{v}])$ is true:

Lemma 5.3.2

$$\mathit{If} \left\{ \begin{array}{l} \mathcal{T}_{\mathcal{U}}(\dot{\mathcal{C}}) \in \mathit{Config} \\ [(\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^{\star} \dot{v}] \ \wedge \ (\dot{v} \neq \star) \\ \mathit{Params}([(\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^{\star} \dot{v}]) \end{array} \right\} \ \Rightarrow \ \exists v.\ [\mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \mathcal{T}_{\mathcal{U}}(\dot{\mathcal{C}}) \Downarrow v] \ \preceq^{tr}_{rm} \ [(\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^{\star} \dot{v}]$$

Proof: *By rule induction on the definition of* $[(\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^{\star} \dot{v}]$.

We need to find v *such that* $[\mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \mathcal{T}_{\mathcal{U}}(\dot{\mathcal{C}}) \Downarrow v]$ *and*

$$[\mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \mathcal{T}_{\mathcal{U}}(\dot{\mathcal{C}}) \Downarrow v] \ \preceq^{tr}_{rm} \ [(\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^{\star} \dot{v}]$$

According to the definition of simulation (eq. 2.9), we must show that the roots of both derivation trees are $\mathcal{R}^{tr}_{rm}$*-related:*

$$(\mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \mathcal{T}_{\mathcal{U}}(\dot{\mathcal{C}}) \Downarrow v) \ \mathcal{R}^{tr}_{rm} \ ((\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^{\star} \dot{v}) \quad [\mathbf{f\ 10.1}],$$

and that their subtrees are related too. This last condition can be satisfied in two ways: for each immediate subtree of $[\mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \mathcal{T}_{\mathcal{U}}(\dot{\mathcal{C}}) \Downarrow v]$ *we can find an immediate subtree of* $[(\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^{\star} \dot{v}]$ *such that both form an* $\mathcal{R}^{tr}_{rm}$*-simulation:*

$$\begin{array}{l} \forall [\mathcal{N}'] \in \mathit{reach}([\mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \mathcal{T}_{\mathcal{U}}(\dot{\mathcal{C}}) \Downarrow v]), \\ \qquad \exists [\mathcal{N}'_s] \in \mathit{reach}([(\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^{\star} \dot{v}]) : [\mathcal{N}'] \preceq^{\mathcal{R}} [\mathcal{N}'_s] \qquad [\mathbf{f\ 10.2}] \end{array}$$

or, there exists an immediate subtree of $[(\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^{\star} \dot{v}]$ *that* $\mathcal{R}^{tr}_{rm}$*-simulates* $[\mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \mathcal{T}_{\mathcal{U}}(\dot{\mathcal{C}}) \Downarrow v]$*:*

$$\exists [\mathcal{N}'_s] \in \mathit{reach}([(\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^{\star} \dot{v}]) : [\mathcal{N}] \preceq^{\mathcal{R}} [\mathcal{N}'_s] \qquad [\mathbf{f\ 10.3}]$$

The first condition, $[\mathbf{f\ 10.1}]$*, is satisfied by showing* $v = \mathcal{T}_{\mathcal{U}}(\dot{v})$*. The second condition requires the use of induction, that is, the application of the inductive hypothesis to each subtree of* $[(\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^{\star} \dot{v}]$*.*

This is the first proof where condition [**f 10.3**] *is required. This is necessary because in some cases the transformation replaces the label of the expression with the label of one of its subexpressions. This is the case for the rule that deals with conditionals:*

(**r 6.8**) $\Sigma(l) = (\texttt{if } t_0^{l_0} \texttt{ then } t_1^{l_1} \texttt{ else } t_2^{l_2})$ *and:*

$$(\Sigma, \mathcal{U}) \vdash \langle l_0, \dot{\rho} \rangle \Downarrow^{\star} \mathit{true} \qquad \wedge \qquad (\Sigma, \mathcal{U}) \vdash \langle l_1, \dot{\rho} \rangle \Downarrow^{\star} \dot{v}$$

Since $\mathit{true} \neq \star$ *and* $\dot{v} \neq \star$ *(initial conditions), we can apply the inductive hypothesis to* $[(\Sigma, \mathcal{U}) \vdash \langle l_0, \dot{\rho} \rangle \Downarrow^{\star} \mathit{true}]$ *and* $[(\Sigma, \mathcal{U}) \vdash \langle l_1, \dot{\rho} \rangle \Downarrow^{\star} \dot{v}]$ *and derive:*

$$[\mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \langle \mathcal{T}_{\mathcal{U}}(l_0), \mathcal{T}_{\mathcal{U}}(\dot{\rho}) \rangle \Downarrow \mathit{true}] \ \preceq_{rm}^{tr} \ [(\Sigma, \mathcal{U}) \vdash \langle l_0, \dot{\rho} \rangle \Downarrow^{\star} \mathit{true}] \quad [\mathbf{f\ 10.4}]$$
$$[\mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \langle \mathcal{T}_{\mathcal{U}}(l_1), \mathcal{T}_{\mathcal{U}}(\dot{\rho}) \rangle \Downarrow v] \ \preceq_{rm}^{tr} \ [(\Sigma, \mathcal{U}) \vdash \langle l_1, \dot{\rho} \rangle \Downarrow^{\star} \dot{v}] \quad [\mathbf{f\ 10.5}]$$

where $v = \mathcal{T}_{\mathcal{U}}(\dot{v})$.

Since $\mathcal{U}(l_1)$ *must be true, we only have two cases for* $\mathcal{T}_{\mathcal{U}}((\texttt{if } t_0^{l_0} \texttt{ then } t_1^{l_1} \texttt{ else } t_2^{l_2})^l)$.

- $\mathcal{U}(l_1)$ and $\mathcal{U}(l_2)$. *Then:*

  $$\mathcal{T}_{\mathcal{U}}((\texttt{if } t_0^{l_0} \texttt{ then } t_1^{l_1} \texttt{ else } t_2^{l_2})^l) = (\texttt{if } \mathcal{T}_{\mathcal{U}}(t_0^{l_0}) \texttt{ then } \mathcal{T}_{\mathcal{U}}(t_1^{l_1}) \texttt{ else } \mathcal{T}_{\mathcal{U}}(t_2^{l_2}))^l$$

  where $\mathcal{T}_{\mathcal{U}}(l_0)$ *and* $\mathcal{T}_{\mathcal{U}}(l_1)$ *are the labels of* $\mathcal{T}_{\mathcal{U}}(t_0^{l_0})$ *and* $\mathcal{T}_{\mathcal{U}}(t_1^{l_1})$ *under* $\mathcal{T}_{\mathcal{U}}(\Sigma)$*, respectively. Then we can apply rule* (**r 1.7**) *to* $(\texttt{if } \mathcal{T}_{\mathcal{U}}(t_0^{l_0}) \texttt{ then } \mathcal{T}_{\mathcal{U}}(t_1^{l_1}) \texttt{ else } \mathcal{T}_{\mathcal{U}}(t_2^{l_2}))^l$, [**f 10.4**] *and* [**f 10.5**] *and derive:*

  $$[\mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \langle \mathcal{T}_{\mathcal{U}}(l), \mathcal{T}_{\mathcal{U}}(\dot{\rho}) \rangle \Downarrow v]$$

  The first condition of the simulation is satisfied by $v = \mathcal{T}_{\mathcal{U}}(\dot{v})$*. The second condition is satisfied by* [**f 10.4**] *and* [**f 10.5**], *which imply* [**f 10.2**].

- $\mathcal{U}(l_1)$ *and* $\neg\mathcal{U}(l_2)$. *Then:*

  $$\mathcal{T}_{\mathcal{U}}((\texttt{if } t_0^{l_0} \texttt{ then } t_1^{l_1} \texttt{ else } t_2^{l_2})^l) = \mathcal{T}_{\mathcal{U}}(t_1^{l_1})$$

Therefore, $\mathcal{T}_{\mathcal{U}}(l) = \mathcal{T}_{\mathcal{U}}(l_1)$*. Using* [**f 10.5**] *we get:*

$$[\mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \langle \mathcal{T}_{\mathcal{U}}(l), \mathcal{T}_{\mathcal{U}}(\dot{\rho})\rangle \Downarrow v]$$

where $v = \mathcal{T}_{\mathcal{U}}(\dot{v})$*. This satisfies condition* [**f 10.1**] *of the simulation. The second condition is satisfied by showing* [**f 10.3**]*. This is done by replacing* $\mathcal{T}_{\mathcal{U}}(l) = \mathcal{T}_{\mathcal{U}}(l_1)$ *in* [**f 10.5**]*. We get:*

$$[\mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \langle \mathcal{T}_{\mathcal{U}}(l), \mathcal{T}_{\mathcal{U}}(\dot{\rho})\rangle \Downarrow v] \;\preceq^{tr}_{rm}\; [(\Sigma,\mathcal{U}) \vdash \langle l_1, \dot{\rho}\rangle \Downarrow^{\star} \dot{v}]$$

which completes the proof.

[**QED**]

The conditions of the theorem not only ensure that $\Sigma' \vdash \mathcal{T}_{\mathcal{U}}(\dot{\mathcal{C}}) \Downarrow v$ successfully terminates but that each subtree (or node) of its derivation tree is simulated by a subtree (or node) of $[(\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^{\star} \dot{v}]$. The safety relation $\mathcal{R}^{tr}_{rm}$ indicates that matching nodes are related by $\mathcal{T}_{\mathcal{U}}$. In particular, the transformed configuration evaluates to the transformed version of $\dot{v}$:

Corollary 5.3.3

If $(\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^{\star} \dot{v}$, $\dot{v} \neq \star$ *and* $Params([(\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^{\star} \dot{v}])$ *then* $\mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \mathcal{T}_{\mathcal{U}}(\dot{\mathcal{C}}) \Downarrow \mathcal{T}_{\mathcal{U}}(\dot{v})$*.*

This corollary shows a very important property of $\mathcal{T}_{\mathcal{U}}(\dot{v})$. By removing all useless variables from $\dot{v}$, $\mathcal{T}_{\mathcal{U}}(\dot{v})$ removes at the same time all $\star$ values from environments and closures. This fact is exploited by the following lemma:

Lemma 5.3.4

If $(v\ \ \mathcal{V}_{\star}\ \ \dot{v})$*, and* $\mathcal{T}_{\mathcal{U}}(\dot{v}) \in Val$ *then* $\mathcal{T}_{\mathcal{U}}(v) = \mathcal{T}_{\mathcal{U}}(\dot{v})$*.*

Proof: *By rule induction on* $(v\ \ \mathcal{V}_{\star}\ \ \dot{v})$*.*

The previous lemma gives us the missing link between the evaluation of $\Sigma \vdash \mathcal{C} \Downarrow v$ and the evaluation of the transformed program $\Sigma' \vdash \mathcal{T}_{\mathcal{U}}(\mathcal{C}) \Downarrow v'$. We are ready to prove our initial claim, that a removable slice is a specialized version of a standard slice:

Theorem 5.3.5

If $\mathit{Slice}(\mathcal{C}, \Sigma, \mathcal{U})$ *and* $\mathit{Params}([(\Sigma, \mathcal{U}) \vdash \mathcal{C} \Downarrow^{\star} \dot{v}])$ *then* $\mathit{Slice}_{rm}(\mathcal{C}, \Sigma, \mathcal{U})$*.*

Proof: *Using definition 4.3.4 we know:*

$$\Sigma \vdash \mathcal{C} \Downarrow v \quad \wedge \quad (\Sigma, \mathcal{U}) \vdash \mathcal{C} \Downarrow^{\star} \dot{v} \quad \wedge \quad (\dot{v} \neq \star)$$

Using $\mathit{Params}([(\Sigma, \mathcal{U}) \vdash \mathcal{C} \Downarrow^{\star} \dot{v}])$ *and corollary 5.3.3 we derive:*

$$\mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \mathcal{T}_{\mathcal{U}}(\mathcal{C}) \Downarrow \mathcal{T}_{\mathcal{U}}(\dot{v})$$

The evaluation relation is only defined for values $v \in \mathit{Val}$*. Therefore* $\mathcal{T}_{\mathcal{U}}(\dot{v}) \in \mathit{Val}$*. Using* $(v \;\; \mathcal{V}_{\star} \;\; \dot{v})$ *we apply lemma 5.3.4 and derive* $\mathcal{T}_{\mathcal{U}}(\dot{v}) = \mathcal{T}_{\mathcal{U}}(v)$*. Putting everything together we have:*

$$\Sigma \vdash \langle l, \rho \rangle \Downarrow v \quad \wedge \quad \mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \mathcal{T}_{\mathcal{U}}(\mathcal{C}) \Downarrow \mathcal{T}_{\mathcal{U}}(v)$$

that is, $\mathit{Slice}_{rm}(\langle l, \rho \rangle, \Sigma, \mathcal{U})$ *(definition 5.3.1).*

[**QED**]

5.4 A Removable Slice Specification

We define a removable slice specification as a validation system that checks a mapping $\mathcal{U}$ against derivation trees of the standard operational semantics. Given $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v]$, we want to define $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \models_{rm} \mathcal{U}$ such that $\mathit{Slice}_{rm}(\langle l, \rho \rangle, \Sigma, \mathcal{U})$ is true.

Theorem 5.3.5 showed us that a removable slice is a specialized standard slice, *i.e.*, a slice with extra constraints defined by the predicate $\models^{par}$. It makes sense then to start our definition of $[\Sigma \vdash \mathcal{C} \Downarrow v] \models_{rm} \mathcal{U}$ by combining the rules of $[\Sigma \vdash \mathcal{C} \Downarrow v] \models_{sl} (\mathcal{U}, \mathcal{D})$ - the standard slice specification - with the constraints defined by $\mathit{Params}([\Sigma \vdash \mathcal{C} \Downarrow v])$. In the case of an application $\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n})$, $\mathcal{C} = \langle l, \rho \rangle$ and $\mathcal{U}(l)$ we would have the following:

$$\begin{array}{l} \Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n}) \\ [\Sigma \vdash \langle l_0, \rho \rangle \Downarrow [l_f,\ \rho_0]] \in \mathit{reach}([\mathcal{N}]) \\ \Sigma(l_f) = (\mathtt{fn}\, x_1 \ldots x_n \Rightarrow t_b^{l_b}) \\ [\Sigma \vdash \langle l_b, \rho' \rangle \Downarrow v] \in \mathit{reach}([\mathcal{N}]) \end{array} \Rightarrow \begin{cases} \mathcal{U}(l_0) \\ \mathcal{U}(l_i) \Rightarrow \mathcal{D}(\langle l_i, \rho \rangle) \subseteq \mathcal{D}(\langle l, \rho \rangle) \\ \mathcal{D}(\langle l_b, \rho' \rangle) - \{x_1, \ldots, x_n\} \subseteq \mathcal{D}([l_f,\ \rho_0]) \\ x_i \in \mathcal{D}(\langle l_b, \rho' \rangle) \Rightarrow \mathcal{U}(l_i) \\ \boxed{\mathcal{U}(x_i) \ \Leftrightarrow\ \mathcal{U}(l_i)} \end{cases}$$

where $\mathcal{U}$ collects information from all the u mappings generated for all the applications of a function during the execution of the program. All local information of variables ($\mathcal{D}$) is eventually gathered by $\mathcal{U}(x)$ at the application rule, which is then used to determine which arguments belong to the slice. The precision gained by assigning individual sets of useful variables to each configuration is lost. Therefore, we can remove them altogether from the specification.

The same rules for detecting useful variables applies. A variable is useful if it is used by the slice:

$$\Sigma(l) = x \ \wedge\ \mathcal{U}(l) \quad \Rightarrow \quad \mathcal{U}(x)$$

The application rule is rewritten as:

$$\begin{array}{l} \Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n}) \ \wedge\ \mathcal{U}(l) \\ [\Sigma \vdash \langle l_0, \rho \rangle \Downarrow [l_f,\ \rho_0]] \in \mathit{reach}([\Sigma \vdash \langle l, \rho \rangle \Downarrow v]) \\ \Sigma(l_f) = (\mathtt{fn}\, x_1 \ldots x_n \Rightarrow t_b^{l_b}) \\ [\Sigma \vdash \langle l_b, \rho' \rangle \Downarrow v] \in \mathit{reach}([\Sigma \vdash \langle l, \rho \rangle \Downarrow v]) \end{array} \Rightarrow \begin{cases} \mathcal{U}(l_0) \ \wedge\ \mathcal{U}(l_b) \\ \forall i = 1, \ldots, n : \mathcal{U}(x_i) \ \Leftrightarrow\ \mathcal{U}(l_i) \end{cases}$$

The rest of the rules are updated accordingly. The new set of rules is defined by $\mathcal{N} \models_{rm}^{loc} \mathcal{U}$ and must be validated against all reachable subtrees inside the slice. The removable slice specification is defined as:

$$[\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \models^{loc}_{rm} \mathcal{U} \quad \Leftrightarrow \quad \mathcal{U}(l) \Rightarrow$$

(r 10.1) $\Sigma(l) = c \;\Rightarrow\; \mathit{true}$

(r 10.2) $\Sigma(l) \in \mathit{Fun} \;\Rightarrow\; \mathit{true}$

(r 10.3) $\Sigma(l) = x \;\Rightarrow\; \mathcal{U}(x)$

(r 10.4) $\Sigma(l) = (p\ t_1^{l_1} \ldots t_n^{l_n}) \;\Rightarrow\; \forall i \in \{1, \ldots, n\} : \mathcal{U}(l_i)$

(r 10.5) $\begin{array}{l} \Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n}) \;\wedge\; \exists \rho' : \\ [\Sigma \vdash \langle l_0, \rho\rangle \Downarrow [l_f,\ \rho_0]] \in \mathit{reach}([\Sigma \vdash \langle l, \rho\rangle \Downarrow v]) \\ \Sigma(l_f) = (\texttt{fn}\, x_1 \ldots x_n \Rightarrow t_b^{l_b}) \\ [\Sigma \vdash \langle l_b, \rho'\rangle \Downarrow v] \in \mathit{reach}([\Sigma \vdash \langle l, \rho\rangle \Downarrow v]) \end{array} \;\Rightarrow\; \begin{cases} \mathcal{U}(l_0) \;\wedge\; \mathcal{U}(l_b) \\ \forall i \in \{1, \ldots, n\} : \\ \quad \mathcal{U}(x_i) \;\Leftrightarrow\; \mathcal{U}(l_i) \end{cases}$

(r 10.6) $\begin{array}{l} \Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n}) \;\wedge\; \exists \rho' : \\ [\Sigma \vdash \langle l_0, \rho\rangle \Downarrow [l_f,\ \rho_0]] \in \mathit{reach}([\Sigma \vdash \langle l, \rho\rangle \Downarrow v]) \\ \Sigma(l_f) = (\texttt{fun}\, y\ x_1 \ldots x_n \Rightarrow t_b^{l_b}) \\ [\Sigma \vdash \langle l_b, \rho'\rangle \Downarrow v] \in \mathit{reach}([\Sigma \vdash \langle l, \rho\rangle \Downarrow v]) \end{array} \;\Rightarrow\; \begin{cases} \mathcal{U}(l_0) \;\wedge\; \mathcal{U}(l_b) \\ \forall i \in \{1, \ldots, n\} : \\ \quad \mathcal{U}(x_i) \;\Leftrightarrow\; \mathcal{U}(l_i) \end{cases}$

(r 10.7) $\begin{array}{l} \Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2}) \\ [\Sigma \vdash \langle l_2, \rho\rangle \Downarrow \mathit{true}] \in \mathit{reach}([\Sigma \vdash \langle l, \rho\rangle \Downarrow v]) \end{array} \;\Rightarrow\; \mathcal{U}(l_0) \;\wedge\; \mathcal{U}(l_1)$

(r 10.8) $\begin{array}{l} \Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2}) \\ [\Sigma \vdash \langle l_2, \rho\rangle \Downarrow \mathit{false}] \in \mathit{reach}([\Sigma \vdash \langle l, \rho\rangle \Downarrow v]) \end{array} \;\Rightarrow\; \mathcal{U}(l_0) \;\wedge\; \mathcal{U}(l_2)$

(r 10.9) $\Sigma(l) = (\texttt{let}\ x = t_1^{l_1}\ \texttt{in}\ t_2^{l_2}) \;\Rightarrow\; \begin{cases} \mathcal{U}(l_2) \\ \mathcal{U}(x) \Rightarrow \mathcal{U}(l_1) \end{cases}$

Figure 5.2: Removable Slice Specification

Definition 5.4.1 (Removable Slice Specification)

$$[\Sigma \vdash \mathcal{C} \Downarrow v] \models_{rm} \mathcal{U} \quad \Leftrightarrow \quad \forall [\mathcal{N}'] \in \mathit{reach}^{\mathcal{U}}([\Sigma \vdash \mathcal{C} \Downarrow v]).\ \ [\mathcal{N}'] \models^{loc}_{rm} \mathcal{U}$$

where $[\mathcal{N}'] \models^{loc}_{rm} \mathcal{U}$ *is defined by the rules in Figure 5.2 and* $\mathit{reach}^{\mathcal{U}}$ *returns the set of subtrees reachable via useful nodes, i.e., inside the slice.*

Examples

We will consider the same set of examples used in chapter 4. The program e_0 is defined by:

$$\begin{array}{lll} e_0 = (\texttt{let} & f_1 = (\texttt{fn}\, x, y \Rightarrow x^{31})^1 & \\ & f_2 = (\texttt{fn}\, a, b \Rightarrow a^{32})^2 & \\ & f_3 = (\texttt{fn}\, c, d \Rightarrow (*\ d\ \ d)^{33})^3 & \\ \texttt{in} & e_4)^0 & \end{array} \quad \text{and} \quad \begin{array}{l} \Sigma \vdash \langle 0, \rho_0 \rangle \Downarrow v \ \wedge \ \rho_0 = [\,] \\ \Sigma \vdash \langle 4, \rho_4 \rangle \Downarrow v \\ \rho_4 = \rho_0[f_i \mapsto [i,\ \rho_0]]_{i=1,2,3} \end{array} \tag{5.6}$$

We start by setting $\mathcal{U}(0)$. According to rule (**r 10.9**) defined for evaluation of `let`-expressions, the following must be true:

$$\mathcal{U}(4) \quad \wedge \quad i \in \{1,2,3\} : \mathcal{U}(f_i) \Rightarrow \mathcal{U}(i) \tag{5.7}$$

Example 1: Let's define e_4 as:

$$\begin{array}{lll} e_4 = (\texttt{let} & g = (\texttt{if}\ e_6\ \texttt{then}\ f_1^7\ \texttt{else}\ f_2^8)^5 & \Sigma \vdash \langle 6, \rho_4 \rangle \Downarrow \mathit{true} \\ & h = (\texttt{if}\ e_{10}\ \texttt{then}\ {f_1}^{11}\ \texttt{else}\ {f_3}^{12})^9 & \Sigma \vdash \langle 10, \rho_4 \rangle \Downarrow \mathit{false} \\ \texttt{in} & e_{13})^4 & \Sigma \vdash \langle 13, \rho_{13} \rangle \Downarrow v \end{array} \tag{5.8}$$

where $\rho_{13} = \rho_4[g \mapsto [1,\ \rho_0]][h \mapsto [3,\ \rho_0]]$ and, e_6 and e_10 closed expressions.

Again, by applying rule (**r 10.9**) to $\mathcal{U}(4)$ and the evaluation of $\langle 4, \rho_4 \rangle$ we derive the following conditions:

$$\mathcal{U}(13) \quad \wedge \quad \begin{cases} \mathcal{U}(g) \ \Rightarrow \ \mathcal{U}(5) \\ \mathcal{U}(h) \ \Rightarrow \ \mathcal{U}(9) \end{cases} \tag{5.9}$$

Let's consider

$$e_{13} = (+\ (g^{15}\ e_{16}\ \ e_{17})^{14}\ \ (h^{19}\ e_{20}\ \ e_{21})^{18})^{13}$$

and

$$\begin{aligned} &\Sigma \vdash \langle 15, \rho_{13}\rangle \Downarrow [1,\ \rho_0]\ \wedge\ \Sigma \vdash \langle 31, \rho_0'\rangle \Downarrow v_{14} \\ &\Sigma \vdash \langle 19, \rho_{13}\rangle \Downarrow [3,\ \rho_0]\ \wedge\ \Sigma \vdash \langle 33, \rho_0''\rangle \Downarrow v_{18} \\ &v = v_{14} + v_{18} \end{aligned} \tag{5.10}$$

where e_{16}, e_{17}, e_{20} and e_{21} are closed expressions. Using $\mathcal{U}(13)$ we apply rule (**r 10.4**) to the evaluation of $\langle 13, \rho_{13}\rangle$ and derive $\mathcal{U}(14)$ and $\mathcal{U}(18)$, which triggers the constraints for the evaluation of expressions 14 and 18. The constraints generated for expression 14 are:

$$\begin{aligned} &\mathcal{U}(15) \quad \text{which implies} \quad \mathcal{U}(g) \quad \text{using} \quad (\textbf{r 10.3}) \\ &\mathcal{U}(x)\ \Rightarrow\ \mathcal{U}(16) \qquad \wedge \qquad \mathcal{U}(y)\ \Rightarrow\ \mathcal{U}(17) \\ &\mathcal{U}(31) \end{aligned} \tag{5.11}$$

$\mathcal{U}(18)$ triggers the following constraints:

$$\begin{aligned} &\mathcal{U}(19) \quad \text{which implies} \quad \mathcal{U}(h) \quad \text{using} \quad (\textbf{r 10.3}) \\ &\mathcal{U}(c)\ \Rightarrow\ \mathcal{U}(20) \qquad \wedge \qquad \mathcal{U}(d)\ \Rightarrow\ \mathcal{U}(21) \\ &\mathcal{U}(33) \end{aligned} \tag{5.12}$$

Now we check the evaluations of bodies 31 and 33. By applying rule (**r 10.3**) to $\Sigma \vdash \langle 31, \rho_0'\rangle \Downarrow v_{14}$ we derive $\mathcal{U}(x)$. Similarly, by applying (**r 10.4**) to $\Sigma \vdash \langle 33, \rho_0''\rangle \Downarrow v_{18}$ first and then (**r 10.3**), we derive $\mathcal{U}(d)$.

Using $\mathcal{U}(x)$ and $\mathcal{U}(d)$, we go back to (5.11) and (5.12) and derive $\mathcal{U}(16)$ and $\mathcal{U}(21)$. Similarly we use $\mathcal{U}(g)$ and $\mathcal{U}(h)$ in (5.9) and derive $\mathcal{U}(5)$ and $\mathcal{U}(9)$.

We apply rules (**r 10.7**) and (**r 10.8**) to:

$$\begin{aligned} &\langle 5, \rho_4\rangle \quad \text{derive} \quad \mathcal{U}(6)\ \wedge\ \mathcal{U}(7) \qquad \text{implies} \quad \mathcal{U}(f_1) \quad \text{using}(\textbf{r 10.3}) \\ &\langle 9, \rho_4\rangle \quad \text{derive} \quad \mathcal{U}(10)\ \wedge\ \mathcal{U}(12) \quad \text{implies} \quad \mathcal{U}(f_3) \quad \text{using}(\textbf{r 10.3}) \end{aligned} \tag{5.13}$$

Finally, using $\mathcal{U}(f_1)$ and $\mathcal{U}(f_3)$ in (5.7), we derive $\mathcal{U}(1)$ and $\mathcal{U}(3)$.

Let $\mathcal{U}_l = \{1, 3, 4, 5, 6, 7, 9, 10, 12, 13, 14, 15, 16, 18, 19, 21\}$, $\mathcal{U}_x = \{f_1, f_3, g, h, c, d\}$ and $\mathcal{U} = \mathcal{U}_x \cup \mathcal{U}_l$. We have shown that $\mathcal{U}$ satisfies the conditions of the removable slice specification for the evaluation of $\langle 0, \rho_0 \rangle$ and that $\mathcal{U}_x$ is the set of variables used by the evaluation of the expressions labeled by $\mathcal{U}_l$. The program is transformed to:

$$\begin{aligned} \mathcal{T}_{\mathcal{U}}(e_0) = (\texttt{let} \quad & f_1 = (\texttt{fn}\, x \Rightarrow x^{31})^1 \\ & f_3 = (\texttt{fn}\, d \Rightarrow (*\ d\ \ d)^{33})^3 \\ \texttt{in} \quad & \mathcal{T}_{\mathcal{U}}(e_4))^0 \end{aligned}$$

where

$$\begin{aligned} \mathcal{T}_{\mathcal{U}}(e_4) = (\texttt{let} \quad & g = f_1^7 \\ & h = f_3^{12} \\ \texttt{in} & \\ & (+\ (g^{15}\ \mathcal{T}_{\mathcal{U}}(e_{16}))^{14}\ \ (h^{19}\ \mathcal{T}_{\mathcal{U}}(e_{21}))^{18})^{13})^4 \end{aligned}$$

Example 2 We build on the results of the first example. The new e_{13} is defined by:

$$e_{13} = (g^{14}\ e_{15}\ \ (h^{17}\ e_{18}\ \ e_{19})^{16})^{13}$$

and

$$\Sigma \vdash \langle 14, \rho_{13} \rangle \Downarrow [1,\ \rho_0]\ \wedge\ \Sigma \vdash \langle 31, \rho'_0 \rangle \Downarrow v_{13}$$

Using $\mathcal{U}(13)$ we apply the application rule (**r 10.5**) to the evaluation of $\langle 13, \rho_{13} \rangle$ and derive the following constraints:

$$\mathcal{U}(14) \quad \wedge \quad \begin{cases} \mathcal{U}(x) \ \Rightarrow\ \mathcal{U}(15) \\ \mathcal{U}(y) \ \Rightarrow\ \mathcal{U}(16) \end{cases} \quad \wedge \quad \mathcal{U}(31) \tag{5.14}$$

We apply $use(31)$ and rule (**r 10.3**) to $\Sigma \vdash \langle 31, \rho'_0 \rangle \Downarrow v_{13}$ and derive $\mathcal{U}(x)$. We do the same with $\mathcal{U}(15)$ and derive $\mathcal{U}(g)$.

We use $\mathcal{U}(x)$ in (5.14) and derive $\mathcal{U}(15)$. Similarly we use $\mathcal{U}(g)$ in (5.9) and derive $\mathcal{U}(5)$. We have already seen in (5.13) that $\mathcal{U}(5)$ generates $\mathcal{U}(6)$, $\mathcal{U}(7)$ and $\mathcal{U}(f_1)$. Finally, using $\mathcal{U}(f_1)$ in (5.7), we derive $\mathcal{U}(1)$.

Therefore $\mathcal{U} = \mathcal{U}_l \cup \mathcal{U}_x$, where $\mathcal{U}_l = \{1, 4, 5, 6, 7, 13, 14, 15\}$ and $\mathcal{U}_x = \{f_1, x, g\}$, satisfies the conditions of the removable slice specification for the evaluation of $\langle 0, \rho_0 \rangle$. $\mathcal{U}_x$ contains the variables used by the evaluation of elements in $\mathcal{U}_l$.

The program is transformed to:

$$\begin{aligned}\mathcal{T}_{\mathcal{U}}(e_0) = (\texttt{let}\quad & f_1 = (\texttt{fn}\, x \Rightarrow x^{31})^1 \\ \texttt{in}\quad & \mathcal{T}_{\mathcal{U}}(e_4))^0\end{aligned}$$

where

$$\begin{aligned}\mathcal{T}_{\mathcal{U}}(e_4) = (\texttt{let}\quad & g = f_1^7 \\ \texttt{in}\quad & (g^{14}\ \mathcal{T}_{\mathcal{U}}(e_{15}))^{13})^4\end{aligned}$$

Example 3 The third example uses the same definition of e_0. e_4 is defined by:

$$\begin{aligned}e_4 = (\texttt{let}\quad & g = (\texttt{fn}\, gg \Rightarrow (gg^7\ e_8\ \ e_9)^6)^5 \\ & h = (\texttt{fn}\, hh \Rightarrow (hh^{12}\ e_{13}\ \ e_{14})^{11})^{10} \\ \texttt{in}\quad & e_{15})^4\end{aligned}$$

By applying rule (**r 10.9**) to $\mathcal{U}(4)$ and the evaluation of $\langle 4, \rho_4 \rangle$ we derive the following conditions:

$$\mathcal{U}(15) \quad \wedge \quad \begin{cases}\mathcal{U}(g) \ \Rightarrow\ \mathcal{U}(5) \\ \mathcal{U}(h) \ \Rightarrow\ \mathcal{U}(11)\end{cases} \tag{5.15}$$

Let's consider

$$e_{15} = (+\ (g^{17}\ f_1^{18})^{16}\ \ (g^{20}\ f_2^{21})^{19}\ \ (h^{23}\ f_2^{24})^{22}\ \ (h^{26}\ f_3^{27})^{25})^{15}$$

Let's analyze expressions $(g^{17}\ f_1^{18})^{16}$ and $(g^{20}\ f_2^{21})^{19}$ first. Both f_1 and f_2 flow to gg^7 in $(gg^7\ e_8\ \ e_9)^6$. The conditions triggered by these applications are:

$$\begin{aligned}&\mathcal{U}(7) \quad \wedge \quad \mathcal{U}(x) \ \Leftrightarrow\ \mathcal{U}(8) \quad \wedge \quad \mathcal{U}(y) \ \Leftrightarrow\ \mathcal{U}(9) \\ &\mathcal{U}(7) \quad \wedge \quad \mathcal{U}(a) \ \Leftrightarrow\ \mathcal{U}(8) \quad \wedge \quad \mathcal{U}(b) \ \Leftrightarrow\ \mathcal{U}(9)\end{aligned} \tag{5.16}$$

Clearly, the evaluation of the bodies of the closures $[1,\, \rho_0]$ and $[2,\, \rho_0]$ that flow to gg^7 mark $\{x\}$ and $\{a\}$, the first arguments of f_1 and f_2. Therefore $\mathcal{U}(7)$ and $\mathcal{U}(8)$ get marked.

Similarly, $[2,\ \rho_0]$ and $[3,\ \rho_0]$ flow to hh^{12} in $(hh^{12}\ e_{13}\ \ e_{14})^{11}$ when $(h^{23}\ f_2^{24})^{22}$ and $(h^{26}\ f_3^{27})^{25}$ are executed. The conditions triggered by these applications are:

$$\begin{array}{ccccccccc} \mathcal{U}(12) & \wedge & \mathcal{U}(a) & \Leftrightarrow & \mathcal{U}(13) & \wedge & \mathcal{U}(b) & \Leftrightarrow & \mathcal{U}(14) \\ \mathcal{U}(12) & \wedge & \mathcal{U}(c) & \Leftrightarrow & \mathcal{U}(13) & \wedge & \mathcal{U}(d) & \Leftrightarrow & \mathcal{U}(14) \end{array} \tag{5.17}$$

Clearly, the evaluation of the bodies of f_2 and f_3 that flow to hh^{12} mark $\{a\}$ and $\{d\}$, the first and second parameters respectively. Therefore, $\mathcal{U}(12)$, $\mathcal{U}(13)$ and $\mathcal{U}(14)$ get marked. This also means that, since $\mathcal{U}(d)\ \Leftrightarrow\ \mathcal{U}(14)$, $\mathcal{U}(b)$ gets marked too. Going back to (5.16), the last result generates $\mathcal{U}(9)$.

Therefore, the whole program is marked!! The transformation does not change the program.

5.4.1 Correctness of the Removable Slice Specification

We prove that the specification generates valid relevant slices. In order to do this we first prove that $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \models_{rm} \mathcal{U}$ implies $[(\Sigma, \mathcal{U}) \vdash \mathcal{C} \Downarrow^{\star} \dot{v}]$, where $\dot{v}$ is different from $\star$, and furthermore, that $[(\Sigma, \mathcal{U}) \vdash \mathcal{C} \Downarrow^{\star} \dot{v}]$ has the property of matching formal and actual parameters.

We start by showing that there exists $\dot{v}$ s.t. $[(\Sigma, \mathcal{U}) \vdash \mathcal{C} \Downarrow^{\star} \dot{v}]$ and that, furthermore, it forms a simulation with $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$. It is easy to check then that useless actual and formal parameters match.

The simulation is defined by first relating the values in both derivation trees. The relation $\mathcal{V}^{\mathcal{U}}_{rm}$ between values is parameterized by $\mathcal{U}$:

$$\begin{array}{rcl} [l',\ \dot{\rho}]\ \mathcal{V}^{\mathcal{U}}_{rm}\ [l,\ \rho] & \Leftrightarrow & \begin{cases} l = l'\ \wedge\ (\dot{\rho}\ \mathcal{R}_{\star}\ \rho) \\ \forall x \in dom(\dot{\rho}), \mathcal{U}(x)\ \Rightarrow\ \dot{\rho}(x)\ \mathcal{V}^{\mathcal{U}}_{rm}\ \rho(x) \end{cases} \\ c\ \mathcal{V}^{\mathcal{U}}_{rm}\ c' & \Leftrightarrow & c = c' \\ \langle l', \dot{\rho}\rangle\ \mathcal{R}^{\mathcal{U}}_{rm}\ \langle l, \rho\rangle & \Leftrightarrow & \begin{cases} l = l'\ \wedge\ (\dot{\rho}\ \mathcal{R}_{\star}\ \rho) \\ \forall x \in dom(\rho), \mathcal{U}(x)\ \Rightarrow\ \dot{\rho}(x)\ \mathcal{R}^{\mathcal{U}}_{rm}\ \rho(x) \end{cases} \end{array} \tag{5.18}$$

The relation is then extended to nodes of the derivation trees $[(\Sigma,\mathcal{U}) \vdash \langle l,\dot{\rho}\rangle \Downarrow^\star \dot{v}]$ and $[\Sigma \vdash \langle l,\rho\rangle \Downarrow v]$:

$$\mathcal{N}\ \mathcal{R}_{rm}\ \mathcal{M} \quad \Leftrightarrow \quad \begin{cases} \mathcal{N} = (\Sigma,\mathcal{U}) \vdash \langle l,\dot{\rho}\rangle \Downarrow^\star \dot{v} \\ \mathcal{M} = \Sigma \vdash \langle l,\dot{\rho}\rangle \Downarrow v \\ \langle l,\dot{\rho}\rangle\ \ \mathcal{R}^{\mathcal{U}}_{rm}\ \ \langle l,\rho\rangle \\ \mathcal{U}(l) \Rightarrow \dot{v}\ \ \mathcal{R}^{\mathcal{U}}_{rm}\ \ v \end{cases}$$

Let's define $\preceq_{rm}$ as the simulation induced by $\mathcal{R}_{rm}$. The next lemma claims that if we can show $[\Sigma \vdash \langle l,\rho\rangle \Downarrow v] \models_{rm} \mathcal{U}$ then the correspondent evaluation under the slice semantics succeeds and its derivation tree forms a $\mathcal{R}^{\mathcal{U}}_{rm}$-simulation with the standard derivation tree:

Lemma 5.4.2

If $[\Sigma \vdash \mathcal{C} \Downarrow v] \models_{rm} \mathcal{U}$ *and* $(\dot{\mathcal{C}}\ \ \mathcal{R}^{\mathcal{U}}_{rm}\ \ \mathcal{C})$ *then* $\exists \dot{v}$:

$$\mathit{Params}([(\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^\star \dot{v}])\ \wedge\ [(\Sigma,\mathcal{U}) \vdash \dot{\mathcal{C}} \Downarrow^\star \dot{v}]\ \preceq_{rm}\ [\Sigma \vdash \mathcal{C} \Downarrow v]$$

Proof: *By rule induction on the definition of* $[\Sigma \vdash \langle l,\rho\rangle \Downarrow v]$.

Theorem 5.4.3 (Correctness of the Removable Slice Specification)

If $[\Sigma \vdash \langle l,\rho\rangle \Downarrow v] \models_{rm} \mathcal{U}$ *and* $\mathcal{U}(l)$ *then* $\mathit{Slice}_{rm}(\mathcal{C},\Sigma,\mathcal{U})$.

Proof: $\langle l,\rho\rangle\ \ \mathcal{R}^{\mathcal{U}}_{rm}\ \ \langle l,\rho\rangle$ *holds trivially. By applying lemma 5.4.2 we derive:*

$$\mathit{Params}([(\Sigma,\mathcal{U}) \vdash \langle l,\rho\rangle \Downarrow^\star \dot{v}])\ \wedge\ [(\Sigma,\mathcal{U}) \vdash \langle l,\rho\rangle \Downarrow^\star \dot{v}]\ \preceq_{rm}\ [\Sigma \vdash \langle l,\rho\rangle \Downarrow v]$$

Using $\mathcal{U}(l)$ *and the definition of* $\mathcal{R}^{\mathcal{U}}_{rm}$ *we derive* $\dot{v}\ \ \mathcal{V}^{\mathcal{U}}_{rm}\ \ v$. *The rules in 5.18 only consider non-$\star$ values. Therefore* $\dot{v} \neq \star$. *By applying definition 4.3.4 to:*

$$\Sigma \vdash \langle l,\rho\rangle \Downarrow v \quad \wedge \quad (\Sigma,\mathcal{U}) \vdash \langle l,\rho\rangle \Downarrow^\star \dot{v} \quad \wedge \quad (\dot{v} \neq \star)$$

we derive $\mathit{Slice}(\langle l,\rho\rangle,\Sigma,\mathcal{U})$ *which together with* $\mathit{Params}([(\Sigma,\mathcal{U}) \vdash \langle l,\rho\rangle \Downarrow^\star \dot{v}])$ *is used to derive* $\mathit{Slice}_{rm}(\langle l,\rho\rangle,\Sigma,\mathcal{U})$ *(theorem 5.3.5).*

[**QED**]

5.5 A Removable Slice CFA

The analysis is derived by performing abstract interpretation on the removable slice specification $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \models_{rm} \mathcal{U}$. Data and control flow information is abstracted using the same framework developed for CFAs in section 3.2 and later applied to the definition of the slice CFA in section 4.6.

Definition 5.5.1 (Removable Slice CFA)
The judgement $\langle l, \widehat{me} \rangle \models^{\widehat{m}}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U})$ is defined as the greatest relation that satisfies the rules in Figure 5.3.

The constraints posed upon elements $\mathcal{U}(l)$ and $\mathcal{U}(x)$ can be directly deduced from the specification. If we consider $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \models_{rm} \mathcal{U}$ as a validation of $\mathcal{U}$ against the derivation tree of $\Sigma \vdash \langle l, \rho \rangle \Downarrow v$ then we should consider $\langle l, \widehat{me} \rangle \models^{\widehat{m}}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U})$ as the validation of $\mathcal{U}$ against the derivation tree of $\langle l, \widehat{me} \rangle \models^{\widehat{m}} (\Sigma, \widehat{\Phi})$, a data and control flow analysis of the standard evaluation. For example, if $\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n})^l$, the constraints defined for $[\Sigma \vdash \langle l, \rho \rangle \Downarrow v] \models_{rm} \mathcal{U}$:

$$\mathcal{U}(l) \quad \Rightarrow \quad \begin{cases} [\Sigma \vdash \langle l_i, \rho \rangle \Downarrow v_i] \models_{rm} \mathcal{U} \quad i = 1, \ldots, n \\ [\Sigma \vdash \langle l_0, \rho \rangle \Downarrow [l_f,\ \rho_0]] \models_{rm} \mathcal{U} \ \wedge \ \boxed{\mathcal{U}(l_0)} \\ \Sigma(l_f) = (\texttt{fn}\ x_1 \ldots x_n \Rightarrow t_b^{l_b}) \\ \rho' = \rho_0[x_i \mapsto v_i]_1^n \\ \boxed{\mathcal{U}(x_i) \ \Leftrightarrow \ \mathcal{U}(l_i)} \\ [\Sigma \vdash \langle l_b, \rho' \rangle \Downarrow v] \models_{rm} \mathcal{U} \ \wedge \ \boxed{\mathcal{U}(l_b)} \end{cases}$$

$$\langle l, \widehat{me} \rangle \models^{\widehat{m}}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U}) \quad \Leftrightarrow \quad \mathcal{U}(l) \Rightarrow \ldots$$

(r 11.1) $\Sigma(l) = c \;\Rightarrow\; \beta_{\mathsf{c}}(c) \sqsubseteq \Pi_{\mathsf{c}}(\widehat{\Phi}(l, \widehat{m}))$

(r 11.2) $\Sigma(l) = x \;\Rightarrow\; \widehat{\Phi}(x, \widehat{me}(x)) \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \;\wedge\; \boxed{\mathcal{U}(x)}$

(r 11.3) $\Sigma(l) \in \mathit{Fun} \;\Rightarrow\; \{[l,\, \widehat{me}]\} \subseteq \Pi_{\lambda}(\widehat{\Phi}(l, \widehat{m}))$

(r 11.4) $\Sigma(l) = (p\ t_1^{l_1} \ldots t_n^{l_n}) \;\Rightarrow\; \begin{cases} i \in \{1, \ldots, n\} :\ \langle l_i, \widehat{me} \rangle \models^{\widehat{m}}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U}) \;\wedge\; \boxed{\mathcal{U}(l_i)} \\ \hat{p}(\widehat{\Phi}(l_1, \widehat{m}), \ldots, \widehat{\Phi}(l_n, \widehat{m})) \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \end{cases}$

(r 11.5) $\Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n}) \;\Rightarrow$

$$\begin{cases} \boxed{\mathcal{U}(l_0)} \;\wedge\; i = 0, \ldots, n :\ \langle l_i, \widehat{me} \rangle \models^{\widehat{m}}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U}) \\ \forall [l_f,\, \widehat{me}_0] \in \Pi_{\lambda}(\widehat{\Phi}(l_0, \widehat{m})) : \\ \begin{cases} (\Sigma(l_f) = (\mathtt{fn}\, x_1 \ldots x_n \Rightarrow t_b^{l_b}) \;\wedge\; \widehat{me}'_0 = \widehat{me}_0[x_i \mapsto \widehat{m}']) \vee \\ (\Sigma(l_f) = (\mathtt{fun}\ y\ x_1 \ldots x_n \Rightarrow t_b^{l_b})) \;\wedge\; \widehat{me}'_0 = \widehat{me}_0[y \mapsto \widehat{m}'][x_i \mapsto \widehat{m}']) \Rightarrow \\ \quad \begin{cases} \widehat{m}' = \widehat{\mathcal{I}}(\Sigma, l, \widehat{m}, l_f) \\ \langle l_b, \widehat{me}'_0 \rangle \models^{\widehat{m}'}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U}) \;\wedge\; \boxed{\mathcal{U}(l_b)} \\ i = 1, \ldots, n :\ \widehat{\Phi}(l_i, \widehat{m}) \sqsubseteq \widehat{\Phi}(x_i, \widehat{m}') \;\wedge\; \boxed{\mathcal{U}(x_i) \;\Leftrightarrow\; \mathcal{U}(l_i)} \\ \widehat{\Phi}(l_b, \widehat{m}') \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \end{cases} \\ \Sigma(l_f) = (\mathtt{fun}\ y\ x_1 \ldots x_n \Rightarrow t_b^{l_b}) \Rightarrow \{[l_f,\, \widehat{me}_0]\} \subseteq \Pi_{\lambda}(\widehat{\Phi}(y, \widehat{m}')) \end{cases} \end{cases}$$

(r 11.6) $\Sigma(l) = (\mathtt{if}\ t_0^{l_0}\ \mathtt{then}\ t_1^{l_1}\ \mathtt{else}\ t_2^{l_2}) \Rightarrow$

$$\begin{cases} \langle l_i, \widehat{me} \rangle \models^{\widehat{m}}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U}) \quad i \in \{1, 2, 3\} \\ \beta_{\mathsf{c}}(\mathit{true}) \sqsubseteq \Pi_{\mathsf{c}}(\widehat{\Phi}(l_0, \widehat{m})) \;\Rightarrow\; \widehat{\Phi}(l_1, \widehat{m}) \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \;\wedge\; \boxed{\mathcal{U}(l_1)} \\ \beta_{\mathsf{c}}(\mathit{false}) \sqsubseteq \Pi_{\mathsf{c}}(\widehat{\Phi}(l_0, \widehat{m})) \;\Rightarrow\; \widehat{\Phi}(l_2, \widehat{m}) \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \;\wedge\; \boxed{\mathcal{U}(l_2)} \end{cases}$$

(r 11.7) $\Sigma(l) = (\mathtt{let}\ x = t_1^{l_1}\ \mathtt{in}\ t_2^{l_2}) \Rightarrow \begin{cases} \langle l_1, \widehat{me} \rangle \models^{\widehat{m}}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U}) \;\wedge\; \boxed{\mathcal{U}(l_2)} \\ \langle l_2, \widehat{me}[x \mapsto \widehat{m}'] \rangle \models^{\widehat{m}'}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U}) \\ \widehat{\Phi}(l_1, \widehat{m}) \sqsubseteq \widehat{\Phi}(x, \widehat{m}') \\ \boxed{\mathcal{U}(x) \;\Leftrightarrow\; \mathcal{U}(l_1)} \qquad \widehat{m}' = \widehat{\mathcal{I}}(\Sigma, l, \widehat{m}, l) \\ \widehat{\Phi}(l_2, \widehat{m}') \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \end{cases}$

Figure 5.3: Removable Slice Context Analysis

are transformed to:

$$\mathcal{U}(l) \quad \Rightarrow \quad \begin{cases} i = 0, \ldots, n: \; \langle l_i, \widehat{me} \rangle \models^{\widehat{m}}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U}) \; \wedge \; \boxed{\mathcal{U}(l_0)} \\ \forall [l_f, \, \widehat{me}_0] \in \Pi_\lambda(\widehat{\Phi}(l_0)), \;\; \Sigma(l_f) = (\texttt{fn}\, x_1 \ldots x_n \Rightarrow t_b^{l_b}): \\ \quad \begin{cases} \widehat{m}' = \widehat{\mathcal{I}}(\Sigma, l, \widehat{m}, l_f) \; \wedge \; \widehat{me}'_0 = \widehat{me}_0[x_i \mapsto \widehat{m}']) \\ \langle l_b, \widehat{me}'_0 \rangle \models^{\widehat{m}'}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U}) \; \wedge \; \boxed{\mathcal{U}(l_b)} \\ i = 1, \ldots, n: \; \widehat{\Phi}(l_i, \widehat{m}) \sqsubseteq \widehat{\Phi}(x_i, \widehat{m}') \; \wedge \; \boxed{\mathcal{U}(x_i) \; \Leftrightarrow \; \mathcal{U}(l_i)} \\ \widehat{\Phi}(l_f, \widehat{m}') \sqsubseteq \widehat{\Phi}(l, \widehat{m}) \end{cases} \end{cases}$$

where the same constraints are placed on elements on a context CFA.

Expressions outside the slice are not needed and, therefore, do not need to be analyzed. This is enforced by the rules in Figure 5.3 where $\neg\mathcal{U}(l) \Rightarrow \langle l, \widehat{me} \rangle \models^{\widehat{m}}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U})$.

5.5.1 Correctness of the Analysis

We prove the correctness of the analysis with respect to the removable slice specification defined in section 5.4. We do this by proving that the analysis is a safe simulation of the specification, according to the safety relation $\mathcal{R}^{cfa}_{\widehat{\Phi},\,\mathcal{U}}$ defined below.

We start by relating values. Given $\widehat{\Phi}$ and $\mathcal{U}$, the $\mathcal{V}^{cfa}_{\widehat{\Phi},\,\mathcal{U}}$ relation is defined by:

$$\begin{aligned} c \;\; \mathcal{V}^{cfa}_{\widehat{\Phi},\,\mathcal{U}} \;\; \hat{v} \quad &\Leftrightarrow \quad \beta_{\mathsf{c}}(c) \sqsubseteq \Pi_{\mathsf{c}}(\hat{v}) \\ [l, \, \rho] \;\; \mathcal{V}^{cfa}_{\widehat{\Phi},\,\mathcal{U}} \;\; \hat{v} \quad &\Leftrightarrow \quad \exists [l, \, \widehat{me}] \in \Pi_\lambda(\hat{v}), \; \forall x \in dom(\rho): \\ & \qquad\qquad \mathcal{U}(x) \Rightarrow (\rho(x) \;\; \mathcal{V}^{cfa}_{\widehat{\Phi},\,\mathcal{U}} \;\; \widehat{\Phi}(x, \widehat{me}(x))) \end{aligned} \tag{5.19}$$

The relation $\mathcal{V}^{cfa}_{\widehat{\Phi},\,\mathcal{U}}$ between closures says that the analysis can only guarantee consistent information of the variables marked by $\mathcal{U}$, *i.e.*, the variables considered as part of the slice. We extend the safety relation to configurations:

$$\langle l, \rho \rangle \;\; \mathcal{R}^{cfa}_{\widehat{\Phi},\,\mathcal{U}} \;\; \langle l, \widehat{me} \rangle \quad \Leftrightarrow \quad \forall x \in dom(\rho). \; \mathcal{U}(x) \Rightarrow \rho(x) \;\; \mathcal{V}^{cfa}_{\widehat{\Phi},\,\mathcal{U}} \;\; \widehat{\Phi}(x, \widehat{me}(x)) \tag{5.20}$$

where again, only values mapped to variables that are part of the slice are related. The same can be said for nodes. We only care about values generated by configurations that are part of the slice:

$$(\Sigma \vdash \langle l, \rho\rangle \Downarrow v) \;\; \mathcal{R}^{cfa}_{rm} \;\; \langle l, \widehat{me}\rangle \models^{\widehat{m}}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U}) \qquad \Leftrightarrow \qquad \mathcal{U}(l) \Rightarrow \begin{cases} \langle l, \rho\rangle \;\; \mathcal{R}^{cfa}_{\widehat{\Phi},\, \mathcal{U}} \;\; \langle l, \widehat{me}\rangle \\ v \;\; \mathcal{V}^{cfa}_{\widehat{\Phi},\, \mathcal{U}} \;\; \widehat{\Phi}(l, \widehat{m}) \end{cases} \tag{5.21}$$

Let $\preceq^{cfa}_{rm}$ be the simulation relation between derivation trees of the slice specification and the slice CFA defined by:

Definition 5.5.2 $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \;\; \preceq^{cfa}_{rm} \;\; [\langle l, \widehat{me}\rangle \models^{\widehat{m}}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U})] \;\; \Leftrightarrow$

- $(\Sigma \vdash \langle l, \rho\rangle \Downarrow v) \;\; \mathcal{R}^{cfa}_{rm} \;\; (\langle l, \widehat{me}\rangle \models^{\widehat{m}}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U}))$ *and*
- *If* $\mathcal{U}(l)$ *then* $\forall[\mathcal{N}] \in reach([\Sigma \vdash \langle l, \rho\rangle \Downarrow v])$, $\exists[\mathcal{M}] \in reach[\langle l, \widehat{me}\rangle \models^{\widehat{m}}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U})]$ *such that* $[\mathcal{N}] \;\; \preceq^{cfa}_{rm} \;\; [\mathcal{M}]$

In this definition, the search for matching immediate subtrees stops when $\neg\mathcal{U}(l)$, as in definition 4.6.3. We prove the following:

Lemma 5.5.3
If $\langle l, \widehat{me}\rangle \models^{\widehat{m}}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U})$, $\Sigma \vdash \langle l, \rho\rangle \Downarrow v$ *and* $(\langle l, \rho\rangle \;\; \mathcal{R}^{cfa}_{\widehat{\Phi},\, \mathcal{U}} \;\; \langle l, \widehat{me}\rangle)$ *then*

$$[\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \models_{rm} \mathcal{U} \quad \wedge \quad [\Sigma \vdash \langle l, \rho\rangle \Downarrow v] \;\; \preceq^{cfa}_{rm} \;\; [\langle l, \widehat{me}\rangle \models^{\widehat{m}}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U})]$$

Proof: *By induction on the derivation tree* $[\Sigma \vdash \langle l, \rho\rangle \Downarrow v]$.

Theorem 5.5.4 (Correctness of the Removable Slice CFA)
If $\langle l, \widehat{me}\rangle \models^{\widehat{m}}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U})$, $\Sigma \vdash \langle l, \rho\rangle \Downarrow v$, $(\langle l, \rho\rangle \;\; \mathcal{R}^{cfa}_{\widehat{\Phi},\, \mathcal{U}} \;\; (\langle l, \widehat{me}\rangle, \widehat{\Phi}, \mathcal{U}))$ *and* $\mathcal{U}(l)$ *then* $Slice_{rm}(\langle l, \rho\rangle, \Sigma, \mathcal{U})$.

5.5.2 Correctness of the Transformation

We prove that if the transformation uses information generated by the removable slice CFA, the transformed program successfully terminates and the evaluated value is the same if the original value was a constant:

Theorem 5.5.5 (Correctness of the Transformation)
Given closed program $\Sigma(l) = P$. *If* $\langle l, \widehat{me}\rangle \models^{\widehat{m}}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U})$ *and* $\Sigma \vdash \langle l_p, [\,]\rangle \Downarrow c$ *then* $\mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \langle \mathcal{T}_{\mathcal{U}}(l), [\,]\rangle \Downarrow c$

Proof: *We can easily prove* $\langle l, [\,]\rangle\ \mathcal{R}^{cfa}_{\widehat{\Phi},\,\mathcal{U}}\ (\langle l, \widehat{me}\rangle, \widehat{\Phi}, \mathcal{U}))$ *by checking definition 5.20. By applying theorem 5.5.4 to the initial conditions we get* $Slice_{rm}(\langle l, [\,]\rangle, \Sigma, \mathcal{U})$.

We have:

$$Slice_{rm}(\langle l, [\,]\rangle, \Sigma, \mathcal{U}) \qquad \Sigma \vdash \langle l, [\,]\rangle \Downarrow c$$

Therefore, by applying the definition of $Slice_{rm}$ *(5.3.1) we derive* $\mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \langle \mathcal{T}_{\mathcal{U}}(l), [\,]\rangle \Downarrow c$.

[**QED**]

Furthermore, the results computed by the analysis contain valid control and data flow information of the transformed program. The following is true:

Theorem 5.5.6
If $\langle l, \widehat{me}\rangle \models^{\widehat{m}}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U})$ *and* $\forall \widehat{m}, l, l'.\ \widehat{\mathcal{I}}(\Sigma, l, \widehat{m}, l') = \widehat{\mathcal{I}}(\Sigma', l, \widehat{m}, l')$ *then*

$$\langle \mathcal{T}_{\mathcal{U}}(l), \mathcal{T}_{\mathcal{U}}(\widehat{me})\rangle \models^{\widehat{m}}_{rm} (\mathcal{T}_{\mathcal{U}}(\Sigma), \mathcal{T}_{\mathcal{U}}(\widehat{\Phi}), \mathcal{U})$$

where all abstract context environments $\widehat{me}$ have to be trimmed off of useless variables in order to satisfy the analysis conditions.

5.6 A Removable Slice 0CFA

We present the 0CFA version of the removable slice CFA presented in section 5.5. The analysis abstracts all binding contexts to the single context $\bullet$ and ignores constants. The new domain of values ($\widehat{Val}$) and the new abstract cache ($\widehat{\Phi}$) follow the definitions used in sections 3.3 and 4.8.

$$l \models^{\bullet}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U}) \quad \Leftrightarrow \quad \mathcal{U}(l) \Rightarrow \ldots$$

$(\mathbf{r\ 12.1})\quad \Sigma(l) = c \Rightarrow \mathit{true}$

$(\mathbf{r\ 12.2})\quad \Sigma(l) = x \Rightarrow \widehat{\Phi}(x) \subseteq \widehat{\Phi}(l) \ \wedge\ \mathcal{U}(x)$

$(\mathbf{r\ 12.3})\quad \Sigma(l) \in \mathit{Fun} \Rightarrow l \in \widehat{\Phi}(l)$

$$(\mathbf{r\ 12.4})\quad \Sigma(l) = (p\ t_1^{l_1} \ldots t_n^{l_n}) \Rightarrow i \in \{1, \ldots, n\} : \begin{cases} l_i \models^{\bullet}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U}) \\ \mathcal{U}(l_i) \end{cases}$$

$$(\mathbf{r\ 12.5})\quad \Sigma(l) = (t_0^{l_0}\ t_1^{l_1} \ldots t_n^{l_n}) \Rightarrow \begin{cases} \mathcal{U}(l_0) \\ l_i \models^{\bullet}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U}) \quad i \in \{0, \ldots, n\} \\ \forall l_f \in \widehat{\Phi}(l_0) : \\ \quad \begin{cases} (\Sigma(l_f) = (\texttt{fn}\ x_1 \ldots x_n \Rightarrow t_b^{l_b}) \vee \\ \ \Sigma(l_f) = (\texttt{fun}\ y\ x_1 \ldots x_n \Rightarrow t_b^{l_b})) \Rightarrow \\ \quad \begin{cases} l_b \models^{\bullet}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U}) \ \wedge\ \mathcal{U}(l_b) \\ i = 1, \ldots, n : \begin{cases} \widehat{\Phi}(l_i) \subseteq \widehat{\Phi}(x_i) \\ \mathcal{U}(x_i) \ \Leftrightarrow\ \mathcal{U}(l_i) \end{cases} \\ \widehat{\Phi}(l_f) \subseteq \widehat{\Phi}(l) \end{cases} \\ \Sigma(l_f) = (\texttt{fun}\ y\ x_1 \ldots x_n \Rightarrow t_f^{l_f}) \Rightarrow l_f \in \widehat{\Phi}(y) \end{cases} \end{cases}$$

$$(\mathbf{r\ 12.6})\quad \Sigma(l) = (\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2}) \Rightarrow \begin{cases} l_i \models^{\bullet}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U}) \ \wedge\ \mathcal{U}(l_i) \quad i \in \{1, 2, 3\} \\ \widehat{\Phi}(l_1) \subseteq \widehat{\Phi}(l) \\ \widehat{\Phi}(l_2) \subseteq \widehat{\Phi}(l) \end{cases}$$

$$(\mathbf{r\ 12.7})\quad \Sigma(l) = (\texttt{let}\ x = t_1^{l_1}\ \texttt{in}\ t_2^{l_2}) \Rightarrow \begin{cases} l_i \models^{\bullet}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U}) \quad i \in \{1, 2\} \\ \widehat{\Phi}(l_1) \subseteq \widehat{\Phi}(x) \\ \widehat{\Phi}(l_2) \subseteq \widehat{\Phi}(l) \\ \mathcal{U}(l_2) \ \wedge\ \mathcal{U}(x) \ \Leftrightarrow\ \mathcal{U}(l_1) \end{cases}$$

Figure 5.4: Removable Slice 0CFA

Definition 5.6.1 (Removable Slice 0CFA) *The judgement* $l \models^{\bullet}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U})$ *is defined as the greatest relation that satisfies the rules in Figure 5.4.*

The rules that define $l \models^{\bullet}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U})$ are an special instance of the rules in Figure 5.3. The analysis deals with flow ($\widehat{\Phi}$) and slice information ($\mathcal{U}$). Flow information is generated only for the expressions and variables that form part of the removable slice $\mathcal{U}$ and, at the same time, $\mathcal{U}$ is generated based on the flow information kept by $\widehat{\Phi}$.

The analysis enjoys the same properties of the general removable slice CFA. In particular, it defines a valid removable slice and a meaning-preserving transformation $\mathcal{T}_{\mathcal{U}}$:

Theorem 5.6.2 *Given closed program* $\Sigma(l) = P$. *If* $l \models^{\bullet}_{rm} (\Sigma, \widehat{\Phi}, \mathcal{U})$ *and* $\Sigma \vdash \langle l, [\,] \rangle \Downarrow c$ *then* $\mathcal{T}_{\mathcal{U}}(\Sigma) \vdash \langle \mathcal{T}_{\mathcal{U}}(l), [\,] \rangle \Downarrow c$.

5.7 Previous and Related work

Wand and Siveroni [WS99] define an 0CFA-based analysis that computes the useless variables of a program and prove the correctness of a transformation that removes such variables from the program. This chapter gives a more precise treatment of the problem.

The transformation and analysis has been presented as a type system by Kobayashi [Kob00], and Fischbach and and Hannan [Kob00, FH01]. The type system defined in [Kob00] is based in a property defined in section 5.3.2:

$$\Sigma \vdash \mathcal{C} \Downarrow v \ \wedge \ \left\{ \begin{array}{l} (\Sigma,\mathcal{U}) \vdash \mathcal{C} \Downarrow^{\star} \dot{v} \ \wedge \ (\dot{v} \neq \star) \\ \mathit{Params}([(\Sigma,\mathcal{U}) \vdash \mathcal{C} \Downarrow^{\star} \dot{v}]) \end{array} \right\} \quad \Rightarrow \quad \mathit{Slice}_{rm}(\Sigma,\mathcal{U},\mathcal{C})$$

The type system assigns a special type $\tau_\star$ to the value $\star$ and introduces a non-deterministic rule that assigns $\tau_\star$ to any expression. Thus, a type assignment that preserves the original type defines a removable slice. It is enough to prove that the type of the program is not $\tau_\star$.

The conditions defined by $Params(-)$ are enforced by the type system: the type of the formals must match the type of the actuals. The algorithm is simple and efficient, due to the simplicity of the type reconstruction techniques.

Fischbach and Hannan [FH01] propose another type-based method to remove useless variables. Their method is based on type annotations that indicate if a variable or expression is needed or not. Their type system takes into consideration effects. This method computes the same kind of information that we do.

Chapter 6

Conclusions and Future Work

The main goal of this dissertation has been to show the importance of specifying program properties in terms of actual executions of programs. Most known analyses are defined based in some intuitive notion of the program property they want to compute, which is then confirmed by proving that the transformations that take advantage of those properties are meaning preserving[1]. Instead, we make the notion of program property explicit. We believe that such definitions provide a better understanding of the problem and can be used to derive more precise analyses.

We also show that an analysis can be derived by abstracting the execution traces of a program where the program property is defined. If the abstract interpretation of the program safely simulates the concrete interpretation, then we can prove that the information (statically) computed by the analysis satisfies the property specification (dynamic).

We have applied the notions of derivation trees and tree (bi)simulations to establish relationships between different evaluations. We are particularly interested in the safety relations between the nodes of a concrete interpretation and an abstract simulation, where

[1]In those cases, it can be said that the property G that the analysis computes is defined as the set of G's that make the transformation correct.

the latter safely simulates the former. This relationship can be used to show that some properties of the abstract interpretation are also properties of the concrete interpretation, which is the essence of program analysis.

The transformations presented in this dissertation do not deal with non-terminating programs. Non-termination, in most cases, is not preserved. For example, the transformation that removes expressions and variables outside a removable slice can turn a looping program into a terminating one. Similarly, a program that fails to terminate successfully due to a run-time error may be transformed into a terminating program. Therefore, we cannot say that the transformations are meaning preserving for all programs if termination is considered as part of the observables of our system.

The natural semantics presented in chapter 2 only defines correct, terminating evaluations. We cannot reason about erroneous and non-convergent programs, that is, given $\mathcal{C}$, $\not\exists v.\ \Sigma \vdash \mathcal{C} \Downarrow v$. It will be of great interest to study how our techniques behave in the presence of derivation trees of non-terminating evaluations, and how the analyses can be integrated with current termination analysis techniques.

We have only dealt with transformations that either replace or remove code from a program. The applicability of the simulation techniques in the presence of transformations that change the position of code (code motion) is yet to be studied. Labeling strategies will be of particular importance in such cases.

Finally, we would like apply the discipline of defining dynamic properties to a wider range of program properties. In particular, we are interested in the definition of safety and security properties that can be used to protect software system against malicious attacks [Sec]. This is part of our current research.

Bibliography

[ABHR99] Martin Abadi, Anindya Banerjee, Nevin Heintze, and John G. Riecke. A core calculus of dependency. In *Conference Record of POPL '99: The 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 147–160, San Antonio, Texas, 20–22 January 1999.

[ALL96] Martín Abadi, Butler Lampson, and Jean-Jacques Lévy. Analysis and caching of dependencies. In *Proceedings of the ACM '96 International Conference on Functional Programming*, pages 83–91, 1996.

[App98] Andrew W. Appel. *Modern Compiler Implementation in ML*. Cambridge University Press, Cambridge, UK, 1998.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, MA, 1986.

[BG85] Rastislav Bodík and Rajiv Gupta. Partial dead code elimination using slicing transformations. In *Proceedings of the ACM SIGPLAN '97 Conference on Programming Language Design and Implementation*, pages 7–21, 1985.

[BHA86] G. L. Burn, C. Hankin, and S. Abramsky. Strictness analysis for higher-order functions. *Science of Computer Programming*, 7:249–278, 1986.

[Bis97a] Sandip K. Biswas. A demand-driven set-based analysis. In *Conference Record of POPL '97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 372–385, Paris, France, 15–17 January 1997.

[Bis97b] Sandip K. Biswas. *Dynamic Slicing in Higher-Order Programming Languages.* PhD thesis, University of Pennsylvania, Philadelphia, PA, 1997.

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings 4th Annual ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[CC92] Patrick Cousot and Radhia Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, August 1992.

[FH01] Adam Fischbach and John Hannan. Type systems and algorithms for useless-variable elimination. In *Symposium on Programs as Data Objects*, volume 2053 of *Lecture Notes in Computer Science*, pages 25–38, Aarhus, Denmark, May 2001.

[GVS95] Milind Gandhe, G. Venkatesh, and Amitabha Sanyal. Labeled λ-calculus and a generalised notion of strictness. In *Asian Computing Science Conference*, Lecture Notes in Computer Science. Springer-Verlag, December 1995.

[HRB84] Susan Horwitz, Thomas Reps, and David Binkley. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, August 1984.

[Hug88] John Hughes. Backward Analysis of Functional Programs. In A.P. Ershov D Bjørner and N.D. Jones, editors, *Proceedings of the IFIP TC2 workshop on Partial Evaluation and Mixed Computation*, pages 187–208. North-Holland, 1988.

[HY91] Paul Hudak and Jonathan Young. Collecting interpretations of expressions. *ACM Transactions on Programming Languages and Systems*, 13(2):269–290, 1991.

[JW94] Suresh Jagannathan and Stephen Weeks. A unified treatment of flow analysis in higher-order languages. In *POPL '95*, 1994.

[JW95] S. Jagannathan and A. Wright. Effective flow analysis for avoiding run-time checks. *Lecture Notes in Computer Science*, 983:207–??, 1995.

[Kah87] Gilles Kahn. Natural Semantics. In *Proc. STACS '87*, volume 247 of *Lecture Notes in Computer Science*, pages 22–39, Berlin, 1987. Springer-Verlag.

[Kob00] Naoki Kobayashi. Type-based useless variable elimination. In *Proceedings of the 2000 ACM SIGPLAN Workshop on Evaluation and Semantics-Based Program Manipulation (PEPM-00)*, pages 84–93, N.Y., January 22–23 2000. ACM Press.

[KRS94] James Knoop, Oliver Rüthing, and Bernhard Steffen. Partial dead code elimination. *SIGPLAN Notices*, 29(6):147–158, 1994. *Proceedings of the ACM SIGPLAN '94 Conference on Programming Language Design and Implementation.*

[Muc97] Steven S. Muchnick. *Advanced Compiler Design and Implementation.* Morgan Kaufmann Publishers, 1997.

[NN97] Flemming Nielson and Hanne Riis Nielson. Infinitary control flow analysis: a collecting semantics for closure analysis. In *Proceedings 24th Annual ACM Symposium on Principles of Programming Languages*, pages 332–345. ACM, January 1997.

[NNH99] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis.* Springer-Verlag, 1999.

[Pal95] Jens Palsberg. Closure analysis in constraint form. *ACM Transactions on Programming Languages and Systems*, 17(1):47–62, January 1995.

[RT96] Thomas Reps and Todd Turnidge. Program specialization via program slicing. In *O. Danvy, R. Glueck, and P. Thiemann (eds.): Dagstuhl Seminar on Partial Evaluation, (Schloss Dagstuhl, Wadern, Germany, Feb. 12-16, 1996)*, volume 1110 of *Lecture Notes in Computer Science*, pages 409–429, New York, NY, 1996. Springer-Verlag.

[Sch95] David A. Schmidt. Natural-semantics-based abstract interpretation. *Lecture Notes in Computer Science*, 983:1–18, 1995.

[Sch98] David A. Schmidt. Data flow analysis is model checking of abstract interpretation. In *Conference Record of POPL '98: The 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 38–48, San Diego, California, 19–21 January 1998.

[Sec] The SECSAFE project. http://www.doc.ic.ac.uk/˜siveroni/secsafe.

[Ses91] P Sestoft. *Analysis and efficient implementation of functional programs.* PhD thesis, DIKU, University of Copenhagen, Denmark, October 1991.

[SF94] Amr Sabry and Matthias Felleisen. Is continuation-passing useful for data flow analysis? In *Proceedings of the Conference on Programming Language Design and Implementation*, pages 1–12, New York, NY, USA, June 1994. ACM Press.

[Shi88] O. Shivers. Control flow analysis in scheme. In *Proceedings of the SIGPLAN '88 Conference on Programming Language Design and Implementation*, pages 164–174, 1988.

[Shi91] Olin Shivers. *Control-Flow Analysis of Higher-Order Languages.* PhD thesis, Carnegie-Mellon University, May 1991.

[Ven91] G. A. Venkatesh. The semantics approach to program slicing. In *Proceedings of the ACM SIGPLAN '91 Conference on Programming language design and implementation*, pages 107–109, Toronto, Canada, 24–28 June 1991. ACM.

[Wei84] Mark Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, August 1984.

[WS99] Mitchell Wand and Igor Siveroni. Constraint systems for useless variable elimination. In *Conference Record of POPL '99: The 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 291–302, San Antonio, Texas, 20–22 January 1999.