Systematic Derivation of Static Analyses for Software Product Lines

Jan Midtgaard
Claus Brabrand
Andrzej Wąsowski

IT University Technical Report Series TR-2014-170

ISSN 1600–6100 March 2014

ISSN 1600–6100

ISBN 978-87-7949-308-7

Copies may be obtained by contacting:

IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S
Denmark

Telephone: +45 72 18 50 00
Telefax: +45 72 18 50 01
Web www.itu.dk

Original version appeared in the *13th International Conference on Modularity*, 2014

Systematic Derivation of Static Analyses for Software Product Lines *

(Full version)

Jan Midtgaard
Dept. of Computer Science, Aarhus University
Aabogade 34, 8200 Aarhus N, Denmark
jmi@cs.au.dk

Claus Brabrand Andrzej Wąsowski
IT University of Copenhagen
Rued Langgaards Vej 7, 2300 Copenhagen S, Denmark
{brabrand,wasowski}@itu.dk

Abstract

A recent line of work *lifts* particular verification and analysis methods to Software Product Lines (SPL). In an effort to generalize such case-by-case approaches, we develop a systematic methodology for lifting program analyses to SPLs using abstract interpretation. Abstract interpretation is a classical framework for deriving static analyses in a compositional, step-by-step manner. We show how to take an analysis expressed as an abstract interpretation and lift each of the abstract interpretation steps to a family of programs. This includes schemes for lifting domain types, and combinators for lifting analyses and Galois connections. We prove that for analyses developed using our method, the soundness of lifting follows by construction. Finally, we discuss approximating variability in an analysis and we derive variational data-flow equations for an example analysis, a constant propagation analysis for a simple imperative language.

Categories and Subject Descriptors D.2.4 [*Software Engineering*]: Software/Program Verification; F.3.2 [*Theory of Computation*]: Semantics of Programming Languages —*Program Analysis*

General Terms Languages, Theory, Verification

Keywords Software Product Lines, Verification, Static Analysis, Abstract Interpretation

1. Introduction

The methodology of *Software Product Lines* (SPLs) [10] enables systematic development of *program families* by maximizing reuse in order to decrease development cost and time-to-market. The SPL method has grown in popularity over the last 20 years, especially in the domain of embedded systems, including safety critical systems with stringent quality requirements on produced code.

* Supported by *The Danish Council for Independent Research* under the Sapere Aude scheme, projects SADL and VARIETE.

MODULARITY '14, April 22–26, 2014, Lugano, Switzerland.

http://dx.doi.org/10.1145/2577080.2577091Reprinted from MODULARITY '14, Proceedings of the 13th International Conference on Modularity, April 22–26, 2014, Lugano, Switzerland, pp. 1–12.

While program families can be implemented using domain specific languages and general purpose model transformation [19, 45], often it is possible to use simpler methods that are more easily amenable to testing and analysis. The most popular [31] implementation method relies on a simple form of two staged computation in preprocessor style: the programming language used (often C) is enriched with the ability to express simple compile time computations (often C preprocessor). At build-time, the source code is first configured, a variant describing a particular product is derived, and only then is this variant compiled or interpreted.

In this two-stage process the compiler handles only the second stage artifacts—the code of the actual product variant. Consequently, all its static analysis mechanisms (such as type checking, data and control-analyses) do not analyze the entire program source code, but only the variant specialized for a particular product. This is sufficient for analyses that aim for program optimization, but entirely unacceptable for analyses that aim at identifying program errors. Often, it is not feasible for the vendor shipping the code to analyze each of the variants separately, due to a combinatorial explosion of the number of products. For example, if variability is used to provide personalization of software for various users, it suffices to have 33 independent features to yield more configurations than people on the planet (2^{33}). As little as 320 optional features yield more configurations than the number of atoms in the universe. Now, the Linux kernel code base contains more than 10,000 configuration options [4]. The problem is particularly burning when runtime errors remain disguised because exhaustive analysis is not possible.

In the last decade, many existing program analysis and verification techniques have been *lifted* to work on program families leading to the emergence of so-called *family-based analyses* [47] (see the related work section for discussion of some of these). The main advantage of these analyses is that they do *not* work in two stages, but analyze the entire code base—all configuration variants at once—at a cost much lower than the accumulated cost of analyzing each of the product variants separately.

Unfortunately, along with the growth of the collection of available lifted analysis methods, a more fundamental worry became increasingly clear: does the variability challenge require redevelopment of the entire language and compiler engineering theory? In response, the industry initiated standardization efforts to codify common understanding of what variability in languages is (for example [29]). In research, a number of papers have started to appear that tackle the more fundamental question of "what is variability in a programming language?" [23, 24].

As part of this larger effort, we attack the problem by developing a systematic understanding of (1) how a single program analysis relates to the lifted analysis, (2) how programming language defini-

tions (including semantics) are enriched with variability and (3) how a program analysis developed formally for a single program can be systematically lifted into a correct analysis for a set of programs.

We develop a systematic methodology for lifting single program analyses using abstract interpretation [13]. Abstract interpretation is a unifying theory of sound abstraction and approximation of structures; a well-established general framework, which can express many analyses (including data-flow analyses [13], control-flow analyses [37], model-checking [16, 17], and type checking [11]). Our method exploits knowledge about a single program analysis to obtain a family-based analyses. The family-based analyses derived using this method are not only sound, but also formally and intimately related to their single program origins. The method is applicable to any analysis expressible as an abstract interpretation. We contribute the following:

- A systematic method for compositional derivation of *sound* SPL analyses based on abstract interpretation.
- Understanding of the structure of the space of family-based analyses (how single program analyses induce family-based analyses, and which of their abstraction components can be reused at family level).
- Understanding of individual family-based analyses (in particular, precisely where analysis precision is lost).
- Transfer of the usual benefits of abstract interpretation to family-based analyses (for example, techniques for trading precision for speed and methods for proving analyses to be semantically sound).
- A step-by-step example-driven demonstration of how to derive a family-based analysis.

We have deliberately chosen a tutorial style of presentation for the introduction to systematic derivation of analyses (based on the calculational approach to abstract interpretation [12], on which our results are founded). For this reason, our results are postponed until Section 4, after SPLs and systematic derivation of analyses have been properly introduced (Sections 2 and 3). We hope that this presentation style maximizes the potential benefits of this paper for the research community developing product line analysis tools. We present a simple imperative language as the running example, formalize its semantics, derive a constant propagation analysis, and show how the whole derivation process and the resulting analysis can be lifted to the family level for analyzing SPLs.

2. From Programs to Software Product Lines

We begin with settling the programming language that we want to analyze. Then, we develop a formal understanding of its semantics (as we aim at provably sound analyses). Finally, we introduce static variability into the language, and into its formal semantics.

IMP Programs: Implementing Single Systems

We use a simple imperative language, IMP, as an example to demonstrate abstract interpretation. In this paper, IMP impersonates a regular general-purpose programming language, aimed at the development of single programs (as opposed to program families). IMP is a well established minimal language, used in teaching and research. We give a brief account of IMP, referring the interested reader to textbooks [41, 48] for more details. We stress that IMP is the running example in the paper. However the presented systematic methodology is not limited to IMP or its features.

Syntax. IMP is structured into two syntactic categories: expressions (integer constants, variables, and binary operations) and statements (no-ops, assignments, statement sequences, conditional statements, and while loops). Its abstract syntax is summarized using the following context free grammar:

$$
\begin{array}{lll}
e & ::= & n \mid \texttt{x} \mid e_0 \oplus e_1 \\
s & ::= & \texttt{skip} \mid \texttt{x := } e \mid s_0 \texttt{ ; } s_1 \mid \\
 & & \texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1 \mid \texttt{while } e \texttt{ do } s
\end{array}
$$

In the above, n stands for an integer constant, $\texttt{x}$ stands for a variable name, and $\oplus$ stands for a binary operator. The precise choice of available operators is immaterial for the remainder of the paper. We denote by *Stm* and *Exp* the set of all statements, s, and expressions, e, generated by the above grammar.

$$\frac{}{\langle \texttt{skip}, \sigma\rangle \to \sigma}\ \textsc{Skip} \qquad \frac{\mathcal{E}(e,\sigma) = v}{\langle \texttt{x := } e, \sigma\rangle \to \sigma[\texttt{x} \mapsto v]}\ \textsc{Assign}$$

$$\frac{\langle s_0, \sigma\rangle \to \langle s_0', \sigma'\rangle}{\langle s_0 \texttt{ ; } s_1, \sigma\rangle \to \langle s_0' \texttt{ ; } s_1, \sigma'\rangle}\ \textsc{Seq1}$$

$$\frac{\langle s_0, \sigma\rangle \to \sigma'}{\langle s_0 \texttt{ ; } s_1, \sigma\rangle \to \langle s_1, \sigma'\rangle}\ \textsc{Seq2}$$

$$\frac{\mathcal{E}(e,\sigma) = v \qquad v \neq 0}{\langle \texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1, \sigma\rangle \to \langle s_0, \sigma\rangle}\ \textsc{If1}$$

$$\frac{\mathcal{E}(e,\sigma) = v \qquad v = 0}{\langle \texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1, \sigma\rangle \to \langle s_1, \sigma\rangle}\ \textsc{If2}$$

$$\frac{\mathcal{E}(e,\sigma) = v \qquad v \neq 0}{\langle \texttt{while } e \texttt{ do } s, \sigma\rangle \to \langle s \texttt{ ; while } e \texttt{ do } s, \sigma\rangle}\ \textsc{While1}$$

$$\frac{\mathcal{E}(e,\sigma) = v \qquad v = 0}{\langle \texttt{while } e \texttt{ do } s, \sigma\rangle \to \sigma}\ \textsc{While2}$$

Figure 1. Small-step structural operational semantics for IMP

Semantics. A state of an IMP program is an abstraction of memory storage (a *store*) mapping variables to values (integer numbers). We write, *Store*, to denote the set of all possible stores. IMP expressions are computed in a given store, denoted by σ below. A function, $\mathcal{E}$, defined below by structural induction, maps an expression and a store to a value, thereby formalizing evaluation of expressions.

$$
\begin{array}{ll}
 & \mathcal{E} : \mathit{Exp} \times \mathit{Store} \to \mathit{Val} \\
\mathit{Val} = \mathbb{Z} & \mathcal{E}(n,\sigma) = n \\
\mathit{Store} = \mathit{Var} \to \mathit{Val} & \mathcal{E}(\texttt{x},\sigma) = \sigma(\texttt{x}) \\
 & \mathcal{E}(e_0 \oplus e_1, \sigma) = \mathcal{E}(e_0,\sigma) \oplus \mathcal{E}(e_1,\sigma)
\end{array}
$$

Figure 1 presents a small-step structural operational semantics for the language. Following the convention popularized by C, we model Boolean values as integers, with zero interpreted as false and everything else as true (see rules IF2 and WHILE2, respectively, IF1 and WHILE1). Note the two types of rules: the typical small-step rules (for instance, SEQ1 or SEQ2), which rewrite a complex statement into a simpler one, possibly updating the store; and the completion rules which execute a statement to completion producing a new store (for instance, SKIP or WHILE2).

Product Families: Lifting IMP to Staged Computation

Implementation of SPL Architectures [10] relies on the existence of a variability mechanism [19] that allows early, or *staged*, configuration of program functionality (i.e., ability to configure program behaviour at build time or compile time). This way, a single program can encode multiple variations of a software product, maximizing code reuse. An individual product is derived by specializing the multi-staged program at product derivation time, before it is built.

$$
\begin{aligned}
P[\![\texttt{skip}]\!]_k &= \texttt{skip} \\
P[\![\texttt{x := } e]\!]_k &= \texttt{x := } e \\
P[\![s_0 \texttt{ ; } s_1]\!]_k &= P[\![s_0]\!]_k \texttt{ ; } P[\![s_1]\!]_k \\
P[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!]_k &= \texttt{if } e \texttt{ then } P[\![s_0]\!]_k \texttt{ else } P[\![s_1]\!]_k \\
P[\![\texttt{while } e \texttt{ do } s]\!]_k &= \texttt{while } e \texttt{ do } P[\![s]\!]_k \\
P[\![\texttt{\#if } \varphi\ s]\!]_k &= \begin{cases} P[\![s]\!]_k & k \vDash \varphi \\ \texttt{skip} & k \nvDash \varphi \end{cases}
\end{aligned}
$$

Figure 2. Preprocessor from $\overline{\text{IMP}}$ to IMP for configuration, k.

A simple form of two-staged computation involving a C-style preprocessor is the most common variability mechanism in practice [31]. We will now lift IMP from describing single programs to program families, admitting two-staged computation in this style.

The compile-time computation is controlled by a product configuration k—a set of product *features* that should be included in the build process. A finite set $\mathbb{F}$ of Boolean variables, f, describes available features, $f \in \mathbb{F}$. A configuration, k, is a subset of *selected* features: $k \subseteq \mathbb{F}$. We write $\mathbb{K}$ for the set of all valid configurations. We only consider valid configurations in the remainder of the paper.

The set of legal product configurations is typically described by a *feature model* [30] or a configuration model in another similar notation [4, 21]. The results of this paper are independent of the choice of configuration language syntax representing the set $\mathbb{K}$, as we are concerned with mathematical proofs more than with implementation details (so the set-theoretic view is simple and convenient). In practice, syntax of feature models can be easily related to sets of valid configurations [3]. An exhaustive account of feature modeling and domain modeling can be found in [19].

Syntax. The programming language $\overline{\text{IMP}}$ is our two-stage extension of IMP. Its abstract syntax includes the same expression and statement languages as IMP, but we add a new compile-time-conditional statement, with keyword `#if`. It takes a condition over features (φ) and a statement (s) that should be executed (included in the product) if the condition is satisfied by the product configuration.

$$
\begin{aligned}
s &::= \ \ldots \mid \texttt{\#if } \varphi\ s \\
\varphi &::= \ f \in \mathbb{F} \mid \neg\varphi \mid \varphi_0 \wedge \varphi_1
\end{aligned}
$$

We also add a syntactic category of Boolean expressions (φ) to write compile-time propositional logic formulae over features. We write, $\overline{FeatExp}$, for the set of all Boolean expressions over features, and $\overline{Stm}$ for the set of all statements of $\overline{\text{IMP}}$. To stress the variability aspect, we will sometimes write $\overline{s}$ to denote a statement from $\overline{Stm}$ (despite the notational overhead). The set of expressions *Exp* remains the same as for IMP.

Observe that adding preprocessor directives to the abstract syntax of IMP was essentially a mechanical transformation of the grammar that will look similar for other, more complex languages.

Semantics: From $\overline{\text{IMP}}$ to IMP. $\overline{\text{IMP}}$'s semantics has two stages: first, given a configuration k compute an IMP program for a given product variant; second, execute the IMP program using regular IMP semantics. Below we present the first stage of $\overline{\text{IMP}}$'s semantics.

We capture the meaning of static conditional expressions over features using a *satisfiability relation*, $\vDash \subseteq \mathbb{K} \times \overline{FeatExp}$, between configurations and Boolean expressions:

$$
\begin{aligned}
k \vDash f \quad &\text{iff} \quad f \in k \\
k \vDash \neg\varphi \quad &\text{iff} \quad k \nvDash \varphi \\
k \vDash \varphi_0 \wedge \varphi_1 \quad &\text{iff} \quad k \vDash \varphi_0 \wedge k \vDash \varphi_1
\end{aligned}
$$

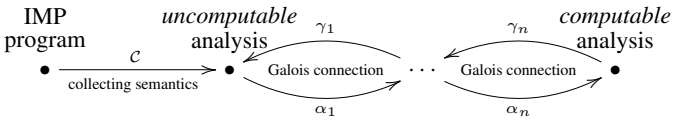


Figure 3. An overview of the abstract interpretation process.

The semantics of the first stage of the computation—a simple preprocessor from $\overline{\text{IMP}}$ to IMP, is specified by the function $P : \overline{Stm} \to \mathbb{K} \to Stm$ in Figure 2. The semantic function P recursively pre-processes all sub-statements of its input. The last case checks whether a feature constraint is satisfied and, if so, it includes the guarded statement. Otherwise it reduces to `skip`, which has the effect of removing the guarded statement. Again, observe that the above rules are independent of the semantics of IMP, so specifying the semantics of the preprocessor is essentially a mechanical process.

3. Systematic Derivation of Analyses

In this section, we assume familiarity with *partial orders*, *complete lattices*, *monotone functions*, *fixed points*, and *the fixed-point theorem*. Appendix A summarizes these concepts.

We leave $\overline{\text{IMP}}$ aside for a few pages and work only with single programs and IMP in the following. We will systematically *derive* static analyses for IMP in a step-by-step compositional manner, using abstract interpretation. We include this section for pedagogical purposes. An analysis designer working with an existing language and analyses for which the abstract interpretation setup exists, would not need to do the work presented here. Instead, she would start right away with lifting the analyses to the family-based setting, following the steps outlined in Sect. 4.

We first introduce a so-called *collecting semantics* for IMP, which is the starting point in abstract interpretation. A collecting semantics takes a program as an argument and then defines how to "collect" information of interest in the given program. It can be seen as an analysis that does not introduce any imprecision (no approximation). Such an analysis is obviously *uncomputable*—it cannot be computed statically. Then, we introduce the notion of a so-called *Galois connection*—a pair of functions capturing information loss between two domains. Finally, we demonstrate how to combine collecting semantics and Galois connections to derive approximate, albeit computable analyses, which can statically determine dynamic properties of programs. An overview of this derivation process is shown in Fig. 3. We use a constant propagation analysis for IMP as the running example.

The process assumes that we have a semantics for our language (cf. Fig. 1), and define a compatible *collecting semantics*. A collecting semantics mimics the behavior of a structural-operational semantics, but with one important difference. Instead of working on *stores*, it works on *sets of stores*. In other words: our property of interest is the *possible* memories (modeled as a set of stores) that may arise at each program point. Furthermore, unknown program input can be modeled as *any* possible input (the set of stores in which a dedicated input variable can take on *any* run time value). Finally, the set of stores is naturally ordered under the subset ordering, $\subseteq$. In this way, the collecting semantics can already be thought of as a fully precise (but uncomputable) analysis. Then the actual computable analyses can be defined as approximations of this semantics.

The collecting semantics for IMP is given in Fig. 4. Going from the semantics to the collecting semantics is straightforward. The function $\mathcal{C}[\![s]\!]$ captures the effect of executing statement s on a set of input stores, by computing the set of possible output stores

$$
\begin{aligned}
\mathcal{C}[\![\texttt{skip}]\!] &= \lambda c.\, c \\
\mathcal{C}[\![\texttt{x := } e]\!] &= \lambda c.\, \{\sigma[\texttt{x} \mapsto v] \mid \sigma \in c \ \wedge\ v \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \\
\mathcal{C}[\![s_0 \texttt{ ; } s_1]\!] &= \mathcal{C}[\![s_1]\!] \circ \mathcal{C}[\![s_0]\!] \\
\mathcal{C}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!] &= \lambda c.\, \mathcal{C}[\![s_0]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\} \\
&\quad \cup\, \mathcal{C}[\![s_1]\!]\{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \\
\mathcal{C}[\![\texttt{while } e \texttt{ do } s]\!] &= \mathrm{lfp}\, \lambda\Phi.\, \lambda c.\, \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \\
&\quad \cup\, \Phi(\mathcal{C}[\![s]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{C}'[\![n]\!] &= \lambda c.\, \{n\} \\
\mathcal{C}'[\![\texttt{x}]\!] &= \lambda c.\, \{\sigma(\texttt{x}) \mid \sigma \in c\} \\
\mathcal{C}'[\![e_0 \oplus e_1]\!] &= \lambda c.\, \{v \mid v \in \{v_0\} \dot{\oplus} \{v_1\} \ \wedge\ \sigma \in c \ \wedge \\
&\qquad v_0 \in \mathcal{C}'[\![e_0]\!]\{\sigma\} \ \wedge\ v_1 \in \mathcal{C}'[\![e_1]\!]\{\sigma\}\}
\end{aligned}
$$

Figure 4. Collecting semantics for IMP where we have that $\mathcal{C}[\![s]\!] : 2^{Store} \to 2^{Store}$ and $\mathcal{C}'[\![e]\!] : 2^{Store} \to 2^{Val}$.

(memory contents after executing s). For instance, since the SKIP rule (cf. Fig. 1) does not modify the store, the corresponding case in the collecting semantics function becomes the identity function on sets of stores: $\lambda c.\, c$. The `if` case results in the *union* of the effect from the two corresponding rules (IF1 and IF2) with a contribution from s_0 (for the stores where the condition evaluates to a non-zero value) and one from s_1 (for the stores where the condition evaluates to zero). The only slightly more complex case is that of the `while` statement which is now given in a standard *fixed-point formulation* (see Appendix A). The case similarly combines the effects corresponding to the two rules (WHILE1 and WHILE2) although with an application of Φ to capture additional iterations of the loop. Observe, that the subordinate function $\mathcal{C}'[\![e]\!]$ does the same exercise for expressions. The symbol, $\dot{\oplus}$, denotes lifting of $\oplus$ to sets—an operator that produces a set of possible values of the expression for each combination of arguments from argument sets.

The collecting semantics captures precisely all executions of the structural operational semantics (SOS). Whenever a store is reachable by derivations of the SOS, then it is included in the corresponding denotation of the collecting semantics (and vice-versa). Formally:

Theorem 1 (Correctness of statement collecting semantics)**.**

$$\forall s \in Stm, c \in 2^{Store} :\ \mathcal{C}[\![s]\!]c = \{\sigma' \mid \sigma \in c \wedge \langle s, \sigma\rangle \to^* \sigma'\}$$

Importantly, given a statement, s, our collecting semantics $\mathcal{C}[\![s]\!] : 2^{Store} \to 2^{Store}$, and in particular the fixed point functional of the while rule:

$$\lambda\Phi.\, \lambda c.\, \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \Phi(\mathcal{C}[\![s]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})$$

are now *monotone* functions over *complete lattices* (see Appendix B for proofs). By Tarski's Fixed-Point Theorem, they admit a unique least fixed point (cf. Appendix A). However, since these lattices have *infinite height*, it is not guaranteed that we can compute a fixed-point in finite time. Indeed, by reduction from the halting problem: if this analysis was *computable*, we would be able to decide whether an input program terminates by comparing the resulting store to lattice bottom. Since IMP is a Turing complete language, this cannot be the case; hence, the analysis must be *uncomputable*.

A *Galois connection* is a pair of functions, $\alpha : \mathbb{C} \to \mathbb{A}$ and $\gamma : \mathbb{A} \to \mathbb{C}$ (respectively known as the *abstraction* and *concretization* functions), connecting two partially ordered sets, $\langle \mathbb{C}, \leqslant\rangle$ and $\langle \mathbb{A}, \sqsubseteq\rangle$ (often called the *concrete* and *abstract* domain, respectively), such that:

$$\forall c \in \mathbb{C}, a \in \mathbb{A} :\quad \alpha(c) \sqsubseteq a \ \Leftrightarrow\ c \leqslant \gamma(a) \qquad (1)$$

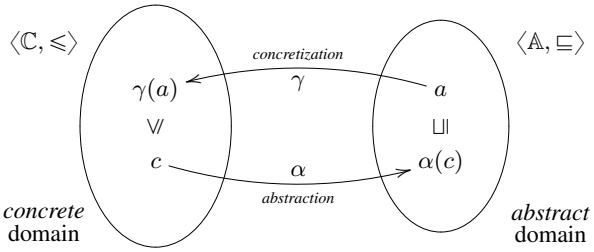


Figure 5. A *Galois connection* between a *concrete*, $\langle \mathbb{C}, \leqslant\rangle$, and an *abstract* domain, $\langle \mathbb{A}, \sqsubseteq\rangle$, connected via an *abstraction*, $\alpha : \mathbb{C} \to \mathbb{A}$, and a *concretization* function, $\gamma : \mathbb{A} \to \mathbb{C}$.

which is often typeset as: $\langle \mathbb{C}, \leqslant\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathbb{A}, \sqsubseteq\rangle$. Figure 5 illustrates a Galois connection graphically. For a concrete domain $\mathbb{C}$, we define *abstraction* and *concretization* functions to and from a more abstract domain $\mathbb{A}$, where information has been abstracted away. Later we will use the Galois connections to approximate an uncomputable analysis formulated over $\mathbb{C}$ with a computable analysis formulated over $\mathbb{A}$.

The seemingly innocent concept has a number of important properties [14]:

(i) α is *monotone*: i.e., $c \leqslant c' \Rightarrow \alpha(c) \sqsubseteq \alpha(c')$, for all $c, c' \in \mathbb{C}$;

(ii) γ is *monotone*; i.e., $a \sqsubseteq a' \Rightarrow \gamma(a) \leqslant \gamma(a')$, for all $a, a' \in \mathbb{A}$;

(iii) $\gamma \circ \alpha$ is *extensive*; i.e., $c \leqslant (\gamma \circ \alpha)(c)$, for all $c \in \mathbb{C}$;

(iv) $\alpha \circ \gamma$ is *reductive*; i.e., $(\alpha \circ \gamma)(a) \sqsubseteq a$, for all $a \in \mathbb{A}$;

(v) If $\mathbb{A}$ and $\mathbb{C}$ are *complete lattices*, then α is a *complete join morphism* (CJM), i.e.,

$$\alpha(\bigcup_{c \in \mathbb{C}} c) \ =\ \bigsqcup_{c \in \mathbb{C}} \alpha(c)$$

where $\cup$ and $\sqcup$ represent lattice joins in $\mathbb{C}$ and $\mathbb{A}$, respectively.

(vi) The composition of two Galois connections is itself a Galois connection (*closure under composition*):

$$\left(\langle \mathbb{C}, \leqslant\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathbb{B}, \subseteq\rangle \ \wedge\ \langle \mathbb{B}, \subseteq\rangle \underset{\alpha'}{\overset{\gamma'}{\leftrightarrows}} \langle \mathbb{A}, \sqsubseteq\rangle\right) \Rightarrow \langle \mathbb{C}, \leqslant\rangle \underset{\alpha' \circ \alpha}{\overset{\gamma \circ \gamma'}{\leftrightarrows}} \langle \mathbb{A}, \sqsubseteq\rangle$$

Due to this last closure property, abstraction can be split into several steps by composing successive Galois connections that incrementally abstracts away information. Indeed, we will do exactly that in the derivation of a computable constant propagation analysis for the IMP language. Collectively, properties (i)–(iv) are equivalent to (1). Hence to test whether two functions form a Galois connection one can either check (1) or check properties (i)–(iv).

Let us now return to our IMP example and show how to use a Galois connection to abstract away information yielding a less precise analysis (although, in this case, still intractable).

Recall that the collecting semantics of a statement s works on sets of stores: it transforms sets of stores to sets of stores, cf. the signature $2^{Store} \to 2^{Store}$ of $\mathcal{C}[\![s]\!]$ in Fig. 4. Figure 6 defines a Galois connection to abstract away information from sets of stores to multi-valued stores, so from $2^{Store} = 2^{Var \to Val}$ to $Var \to 2^{Val}$. Multi-valued stores are less precise than sets of stores, because they lose relational information about the values of different variables. Consider the (concrete) store set, $c = \{[x{\mapsto}1, y{\mapsto}2], [x{\mapsto}2, y{\mapsto}1]\}$, as an example. The abstraction function, α_{CB}, abstracts the store set c into $b = \alpha_{\text{CB}}(c) = [x{\mapsto}\{1,2\}, y{\mapsto}\{1,2\}]$. Like most *path-insensitive* program analyses, which merge and abstract away analysis information at control-flow confluence points (e.g., where `then` *joins*

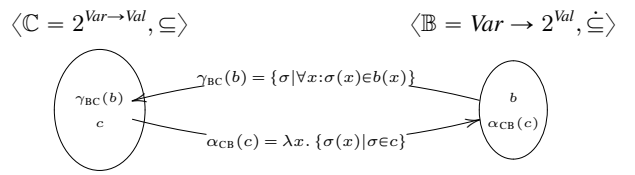


Figure 6. Galois connection between domains, $\mathbb{C}$ and $\mathbb{B}$: $\langle 2^{Var \to Val}, \subseteq \rangle \underset{\alpha_{CB}}{\overset{\gamma_{BC}}{\leftrightarrows}} \langle Var \to 2^{Val}, \dot{\subseteq} \rangle$.

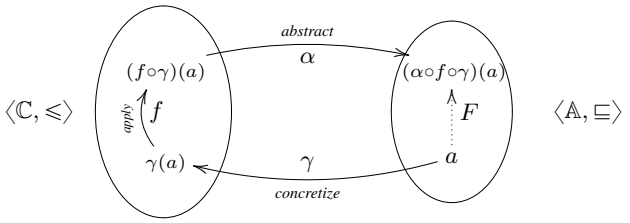


Figure 7. Abstract domain function, $F = \alpha \circ f \circ \gamma : \mathbb{A} \to \mathbb{A}$, *derived* from a concrete domain function, $f : \mathbb{C} \to \mathbb{C}$, via a round-trip in the Galois connection.

`else`), our abstract store b, will now have "forgotten" which values of variable x go with which values of y. Now if the next statement computes, say, multiplication $x * y$, the analysis will *conservatively over-approximate* the set of possible values to $\{1, 2, 4\}$, admitting *spurious values*, 1 and 4, in addition to the precise answer: $\{2\}$. The approximate response is bound to include the precise answer; in other words, we have a *sound* analysis: $\{2\} \subseteq \{1, 2, 4\}$.

Not only *values* can be abstracted from $\mathbb{C}$ to $\mathbb{A}$. In fact, also *functions* defined on the concrete domain, $f : \mathbb{C} \to \mathbb{C}$, can be abstracted to work on the abstract domain, $\alpha \circ f \circ \gamma = F : \mathbb{A} \to \mathbb{A}$. Figure 7 illustrates this process which transforms an argument, $a \in \mathbb{A}$, in three simple steps: (1) *concretize* a, $\gamma(a) \in \mathbb{C}$; (2) *apply* f, $(f \circ \gamma)(a) \in \mathbb{C}$; and (3) *abstract* the result, $(\alpha \circ f \circ \gamma)(a) \in \mathbb{A}$. Also, if f is monotone, then its composition with a monotone α and γ is monotone. In general, any monotone over-approximation of the composition is sufficient for a sound analysis.

In our case, we derive from, $\mathcal{C}[\![s]\!] : 2^{Var \to Val} \to 2^{Var \to Val}$, a function working on the abstracted domain, $\alpha_{CB} \circ \mathcal{C}[\![s]\!] \circ \gamma_{BC} : (Var \to 2^{Val}) \to (Var \to 2^{Val})$. This step is crucial—we use the Galois connection to derive a more abstract semantics (and thus a more approximating analysis) from the less abstract semantics (here the collecting semantics). Let us elaborate on this.

Cousot and Cousot [13] observed that even *fixed points* transfer from $\mathbb{C}$ to $\mathbb{A}$. If $\langle \mathbb{C}, \leqslant \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathbb{A}, \sqsubseteq \rangle$ is a Galois connection whose domains, $\mathbb{C}$ and $\mathbb{A}$, are *complete lattices* and, f, is a *monotone* function on $f : \mathbb{C} \to \mathbb{C}$, then by the *fixed point transfer theorem* [13]:

$$\alpha(\mathrm{lfp}\, f) \;\sqsubseteq\; \mathrm{lfp}\, F \;\sqsubseteq\; \mathrm{lfp}\, F^{\#}$$

where $F = \alpha \circ f \circ \gamma$ and $F^{\#}$ is some monotone, conservative *over*-approximation of F; formally: $F \dot{\sqsubseteq} F^{\#}$ (i.e., $\forall a \in \mathbb{A} : F(a) \sqsubseteq F^{\#}(a)$). Note that F represents the *best possible function* over the chosen abstract domain [14]. The above version of the fixed point transfer theorem still lets us approximate the desired fixed point. Under the stronger assumption that $\alpha \circ f = F \circ \alpha$ then a stronger version of the theorem guarantees that no approximation of the fixed point is taking place: $\alpha(\mathrm{lfp}\, f) = \mathrm{lfp}\, F$ [13].

The approach to abstract interpretation adopted in this paper, known as the *calculational approach* [12], advocates simple algebraic manipulation to obtain a *direct expression* for the function, F

$$\begin{aligned}
\mathcal{B}[\![\texttt{skip}]\!] &= \lambda b.\ b \\
\mathcal{B}[\![\texttt{x := } e]\!] &= \lambda b.\ b[\texttt{x} \mapsto \mathcal{B}'[\![e]\!]b] \\
\mathcal{B}[\![s_0 \texttt{ ; } s_1]\!] &= \mathcal{B}[\![s_1]\!] \circ \mathcal{B}[\![s_0]\!] \\
\mathcal{B}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!] &= \lambda b.\ \mathcal{B}[\![s_0]\!]b \dot{\cup} \mathcal{B}[\![s_1]\!]b \\
\mathcal{B}[\![\texttt{while } e \texttt{ do } s]\!] &= \mathrm{lfp}\, \lambda \Phi.\ \lambda b.\ b \dot{\cup} \Phi(\mathcal{B}[\![s]\!]b) \\
\\
\mathcal{B}'[\![n]\!] &= \lambda b.\ \{n\} \\
\mathcal{B}'[\![\texttt{x}]\!] &= \lambda b.\ b(\texttt{x}) \\
\mathcal{B}'[\![e_0 \oplus e_1]\!] &= \lambda b.\ \mathcal{B}'[\![e_0]\!]b \dot{\oplus} \mathcal{B}'[\![e_1]\!]b
\end{aligned}$$

Figure 9. Systematically derived *over-approximated* abstracted collecting semantics, $\mathcal{B}[\![s]\!] : (Var \to 2^{Val}) \to (Var \to 2^{Val})$ and $\mathcal{B}'[\![e]\!] : (Var \to 2^{Val}) \to 2^{Val}$.

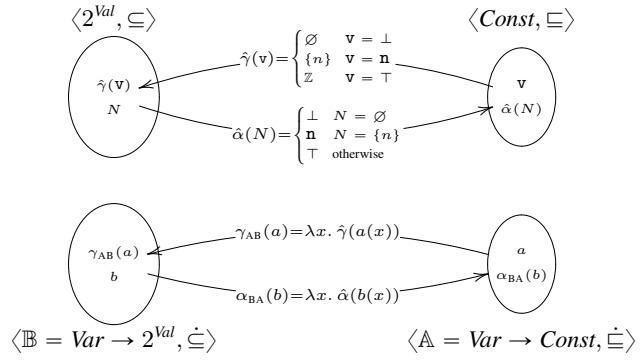


Figure 10. Galois connection: $\langle 2^{Val}, \subseteq \rangle \underset{\hat{\alpha}}{\overset{\hat{\gamma}}{\leftrightarrows}} \langle Const, \sqsubseteq \rangle$ (top diagram) along with its pointwise lifting (bottom diagram): $\langle Var \to 2^{Val}, \dot{\subseteq} \rangle \underset{\alpha_{BA}}{\overset{\gamma_{AB}}{\leftrightarrows}} \langle Var \to Const, \dot{\sqsubseteq} \rangle$.

(if, indeed, it exists); or, a *sound approximation* thereof, $F^{\#}$. It is thus a *systematic* (as in *"pen and paper"*) rather than *automatic* (as in *"computer generated"*) approach for *deriving* analyses.

Returning to our example, we now apply these ideas to our IMP analysis to obtain a *direct expression* for an over-approximation of $\alpha_{CB} \circ \mathcal{C}[\![s]\!] \circ \gamma_{BC}$ which we henceforth abbreviate, $\mathcal{B}[\![s]\!]$. Figure 8 illustrates how this may be done for the `if` case. Notice, how additional approximation is introduced (highlighted in boldface).

If we repeat this systematic derivation process for all the remaining cases, we can derive the over-approximated abstracted collecting semantics, $\mathcal{B}[\![s]\!]$, shown in Fig. 9 (where the case for `if` was the one we derived in Figure 8). Formally, $\mathcal{B}$ is related to $\mathcal{C}$ as follows:

Theorem 2 (Soundness of approximate statement semantics)**.**

$$\forall s \in Stm, b \in \mathbb{B} :\ (\alpha_{CB} \circ \mathcal{C}[\![s]\!] \circ \gamma_{BC})(b) \dot{\subseteq} \mathcal{B}[\![s]\!]\, b$$

(See [39], for proof.) This is now starting to look like a conventional static analysis. However, it is still intractable. The program:

```
x := 1 ; while (1) x := x + 1
```

will give rise to an *infinite* multi-valued abstract store $b = [x \mapsto \{1, 2, 3, ...\}]$. Our next Galois connection will remedy this in abstracting our abstract domain, $Var \to 2^{Val}$, even further into a domain with *finite* height, thereby guaranteeing an analysis computable with a Kleene fixed point iteration.

Figure 10 presents a Galois connection between $\mathbb{B} = \mathrm{Var} \to 2^{Val}$ and $\mathbb{A} = \mathrm{Var} \to Const$, $\langle \mathbb{B}, \dot{\subseteq} \rangle \underset{\alpha_{BA}}{\overset{\gamma_{AB}}{\leftrightarrows}} \langle \mathbb{A}, \dot{\sqsubseteq} \rangle$, for abstracting the multi-valued store domain even further to a pointwise lifted

$$
\begin{array}{ll}
(\alpha_{\text{CB}} \circ \mathcal{C}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!] \circ \gamma_{\text{BC}})(b) & \text{(start of derivation)} \\
= (\alpha_{\text{CB}} \circ (\lambda c.\, \mathcal{C}[\![s_0]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \mathcal{C}[\![s_1]\!]\{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\}) \circ \gamma_{\text{BC}})(b) & \text{(by def. of } \mathcal{C}\text{, Fig. 4)} \\
= \alpha_{\text{CB}}(\mathcal{C}[\![s_0]\!]\{\sigma \in \gamma_{\text{BC}}(b) \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \mathcal{C}[\![s_1]\!]\{\sigma \in \gamma_{\text{BC}}(b) \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\}) & (\beta\text{-reduction}) \\
= \alpha_{\text{CB}}(\mathcal{C}[\![s_0]\!]\{\sigma \in \gamma_{\text{BC}}(b) \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}) \;\dot\cup\; \alpha_{\text{CB}}(\mathcal{C}[\![s_1]\!]\{\sigma \in \gamma_{\text{BC}}(b) \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\}) & (\alpha_{\text{CB}} \text{ is a complete join morphism, p. 4 (v)}) \\
\dot\subseteq \alpha_{\text{CB}}(\mathcal{C}[\![s_0]\!](\gamma_{\text{BC}}(b))) \;\dot\cup\; \alpha_{\text{CB}}(\mathcal{C}[\![s_1]\!](\gamma_{\text{BC}}(b))) & \textbf{(over-approximation: } \mathcal{C} \text{ and } \alpha_{\text{CB}} \text{ monotone)} \\
= (\alpha_{\text{CB}} \circ \mathcal{C}[\![s_0]\!] \circ \gamma_{\text{BC}})(b) \;\dot\cup\; (\alpha_{\text{CB}} \circ \mathcal{C}[\![s_1]\!] \circ \gamma_{\text{BC}})(b) & \text{(by def. of function composition)} \\
\dot\subseteq \mathcal{B}[\![s_0]\!]\, b \;\dot\cup\; \mathcal{B}[\![s_1]\!]\, b & \text{(by inductive hypothesis: } \alpha_{\text{CB}} \circ \mathcal{C}[\![s_i]\!] \circ \gamma_{\text{BC}} \dot\subseteq \mathcal{B}[\![s_i]\!]\text{, twice)} \\
= \mathcal{B}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!]b & \text{(by def. of } \mathcal{B}\text{)}
\end{array}
$$

Figure 8. Systematic derivation of *over-approximating* semantics, $\mathcal{B}[\![s]\!] : (\mathit{Var} \to 2^{\mathit{Val}}) \to (\mathit{Var} \to 2^{\mathit{Val}})$, for `if`, by abstracting collecting semantics, $\alpha_{\text{CB}} \circ \mathcal{C}[\![s]\!] \circ \gamma_{\text{BC}}$. Operators, $\dot\cup$ and $\dot\subseteq$, are extended to functions: $f \mathbin{\dot\cup} g = \lambda x.\, f(x) \cup g(x)$ and $f \mathbin{\dot\subseteq} g = \forall x.\, f(x) \subseteq g(x)$.

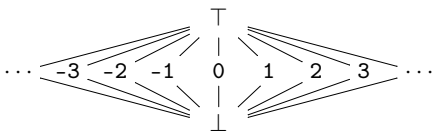


Figure 11. The constant propagation lattice: $\langle \mathit{Const}, \sqsubseteq \rangle$ with $\sqcup$ as its least upper bound operator (aka., *join*).

$$
\begin{array}{rl}
\mathcal{A}[\![\texttt{skip}]\!] &= \lambda a.\, a \\
\mathcal{A}[\![\texttt{x := } e]\!] &= \lambda a.\, a[\texttt{x} \mapsto \mathcal{A}'[\![e]\!]a] \\
\mathcal{A}[\![s_0 \texttt{ ; } s_1]\!] &= \mathcal{A}[\![s_1]\!] \circ \mathcal{A}[\![s_0]\!] \\
\mathcal{A}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!] &= \mathcal{A}[\![s_0]\!] \mathbin{\ddot\cup} \mathcal{A}[\![s_1]\!] \\
\mathcal{A}[\![\texttt{while } e \texttt{ do } s]\!] &= \mathrm{lfp}\, \lambda\Phi.\, \lambda a.\, a \mathbin{\dot\cup} \Phi(\mathcal{A}[\![s]\!]a)
\end{array}
$$

$$
\begin{array}{rl}
\mathcal{A}'[\![n]\!] &= \lambda a.\, \mathtt{n} \\
\mathcal{A}'[\![\texttt{x}]\!] &= \lambda a.\, a(\texttt{x}) \\
\mathcal{A}'[\![e_0 \oplus e_1]\!] &= \lambda a.\, \mathcal{A}'[\![e_0]\!]a \mathbin{\hat\oplus} \mathcal{A}'[\![e_1]\!]a
\end{array}
$$

Figure 12. Constant propagation $\mathcal{A}[\![s]\!] : (\mathit{Var} \to \mathit{Const}) \to (\mathit{Var} \to \mathit{Const})$ and $\mathcal{A}'[\![e]\!] : (\mathit{Var} \to \mathit{Const}) \to \mathit{Const}$.

constant propagation lattice (see Figure 11). If we now repeat the systematic derivation steps analogous to the steps of Figure 8 on the further abstracted analysis, $\alpha_{\text{BA}} \circ \mathcal{B}[\![s]\!] \circ \gamma_{\text{AB}}$, we can finally derive a *computable* constant propagation analysis as an over-approximation of $\alpha_{\text{BA}} \circ \mathcal{B}[\![s]\!] \circ \gamma_{\text{AB}}$, which we call $\mathcal{A}[\![s]\!]$—see Fig. 12. We show the derivation steps for the conditional statement in Fig. 13.

Since operators are functions, they too get abstracted by our Galois connection. Recall that our example uses $\dot\oplus$, the pointwise extension of the binary operator $\oplus$, defined as $V_0 \mathbin{\dot\oplus} V_1 = \{v_0 \oplus v_1 \mid v_0 \in V_0 \wedge v_1 \in V_1\}$. The abstract counterpart, $\hat\oplus$, can be calculated by following the same recipe: $\hat\alpha(\hat\gamma(V) \mathbin{\dot\oplus} \hat\gamma(V')) \sqsubseteq V \mathbin{\hat\oplus} V'$, i.e., by concretizing its arguments, performing the corresponding concrete operation, and finally abstracting the outcome. The resulting abstract operator, $\hat\oplus$, can be computed effectively (in constant time) for all concrete binary operators:

$$
v_0 \mathbin{\hat\oplus} v_1 = \begin{cases} \bot & \text{if } v_0 = \bot \vee v_1 = \bot \\ \mathtt{n} & \text{if } v_0 = \mathtt{n}_0 \wedge v_1 = \mathtt{n}_1, \text{ where } \mathtt{n} = \mathtt{n}_0 \oplus \mathtt{n}_1 \\ \top & \text{otherwise} \end{cases}
$$

Finally, we write, $\dot\sqcup$, to denote the pointwise join in the $\mathit{Var} \to \mathit{Const}$ lattice: $a_0 \mathbin{\dot\sqcup} a_1 = \lambda \texttt{x}.\, a_0(\texttt{x}) \sqcup a_1(\texttt{x})$. This operator is then further lifted pointwise: $f \mathbin{\ddot\cup} g = \lambda a.\, f(a) \mathbin{\dot\sqcup} g(a)$.

$$
\begin{array}{ll}
(\alpha_{\text{BA}} \circ \mathcal{B}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!] \circ \gamma_{\text{AB}})(a) & \text{(start of derivation)} \\
= (\alpha_{\text{BA}} \circ (\lambda b.\, \mathcal{B}[\![s_0]\!]b \;\dot\cup\; \mathcal{B}[\![s_1]\!]b) \circ \gamma_{\text{AB}})(a) & \text{(by def. of } \mathcal{B}\text{, Fig. 9)} \\
= \alpha_{\text{BA}}(\mathcal{B}[\![s_0]\!](\gamma_{\text{AB}}(a)) \;\dot\cup\; \mathcal{B}[\![s_1]\!](\gamma_{\text{AB}}(a))) & (\beta\text{-reduction}) \\
= \alpha_{\text{BA}}(\mathcal{B}[\![s_0]\!](\gamma_{\text{AB}}(a))) \mathbin{\dot\sqcup} \alpha_{\text{BA}}(\mathcal{B}[\![s_1]\!](\gamma_{\text{AB}}(a))) & (\alpha_{\text{BA}} \text{ is a CJM, p. 4 (v)}) \\
\dot\subseteq \mathcal{A}[\![s_0]\!]\, a \mathbin{\dot\sqcup} \mathcal{A}[\![s_1]\!]\, a & \text{(by inductive hypothesis, twice)} \\
= \mathcal{A}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!]\, a & \text{(by def. of } \mathcal{A}\text{)}
\end{array}
$$

Figure 13. Systematic derivation of $\mathcal{A}[\![s]\!]$ from $\alpha_{\text{BA}} \circ \mathcal{B}[\![s]\!] \circ \gamma_{\text{AB}}$ for the `if` case.

$$
\begin{array}{rl}
[\![\texttt{skip}^{\ell}]\!]_{\text{out}} &= [\![\texttt{skip}^{\ell}]\!]_{\text{in}} \\
[\![\texttt{x :=}^{\ell} e]\!]_{\text{out}} &= [\![\texttt{x :=}^{\ell} e]\!]_{\text{in}}[\texttt{x} \mapsto \mathcal{A}'[\![e]\!][\![\texttt{x :=}^{\ell} e]\!]_{\text{in}}] \\
[\![s_0^{\ell_0} \texttt{ ;}^{\ell} s_1^{\ell_1}]\!]_{\text{out}} &= [\![s_1^{\ell_1}]\!]_{\text{out}} \\
[\![s_1^{\ell_1}]\!]_{\text{in}} &= [\![s_0^{\ell_0}]\!]_{\text{out}} \\
[\![s_0^{\ell_0}]\!]_{\text{in}} &= [\![s_0^{\ell_0} \texttt{ ;}^{\ell} s_1^{\ell_1}]\!]_{\text{in}} \\
[\![\texttt{if}^{\ell} e \texttt{ then } s_0^{\ell_0} \texttt{ else } s_1^{\ell_1}]\!]_{\text{out}} &= [\![s_0^{\ell_0}]\!]_{\text{out}} \mathbin{\dot\sqcup} [\![s_1^{\ell_1}]\!]_{\text{out}} \\
[\![s_0^{\ell_0}]\!]_{\text{in}} &= [\![\texttt{if}^{\ell} e \texttt{ then } s_0^{\ell_0} \texttt{ else } s_1^{\ell_1}]\!]_{\text{in}} \\
[\![s_1^{\ell_1}]\!]_{\text{in}} &= [\![\texttt{if}^{\ell} e \texttt{ then } s_0^{\ell_0} \texttt{ else } s_1^{\ell_1}]\!]_{\text{in}} \\
[\![\texttt{while}^{\ell} e \texttt{ do } s^{\ell_0}]\!]_{\text{out}} &= [\![s^{\ell_0}]\!]_{\text{in}} \\
[\![s^{\ell_0}]\!]_{\text{in}} &= [\![\texttt{while}^{\ell} e \texttt{ do } s^{\ell_0}]\!]_{\text{in}} \mathbin{\dot\sqcup} [\![s^{\ell_0}]\!]_{\text{out}}
\end{array}
$$

Figure 14. Data-flow equations for constant propagation of Fig. 12

Since our domain now has a finite height, we have a tractable analysis. Indeed our example program from before gives rise to a *finite* abstract store, $a = [x \mapsto \top]$. Also, as a byproduct of the calculation, the analysis is provably sound:

Theorem 3 (Soundness of statement analysis)**.**

$$\forall s \in \mathit{Stm}, a \in \mathbb{A} :\ (\alpha_{\text{BA}} \circ \mathcal{B}[\![s]\!] \circ \gamma_{\text{AB}})(a) \mathbin{\dot\subseteq} \mathcal{A}[\![s]\!]\, a$$

Notice again how this follows the recurring α-γ composition pattern. Also, Thm. 3 composes with the result of Thm. 2 yielding soundness of the analysis not only with respect to the approximate semantics $\mathcal{B}$, but also with respect to the original collecting semantics $\mathcal{C}$.

We may choose to implement the analysis in Figure 12 directly. Since the collecting semantics was compositional so is the resulting analysis, i.e., the analysis of straight-line code is straight-line (without fixed point computation), only loops require local fixed point computations. We may use Kleene's Fixed-Point Theorem to calculate these iteratively (cf. Appendix A).

To extract corresponding data-flow equations we assume the individual statements have been uniquely labelled with labels, ℓ,

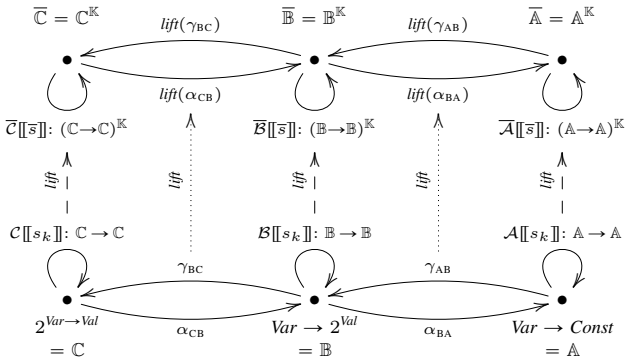


Figure 15. Abstract interpretation of programs (bottom line) along with *lifted* "variational abstract interpretation" of SPLs (top line).

to distinguish the individual flow to and from them and adapt $\mathcal{A}$ to work over these. The corresponding data-flow equations are shown in Fig. 14. Again the transformation from Fig. 12 to Fig. 14 is essentially mechanical. For each statement s^ℓ (program point) we generate two flow variables $[\![s^\ell]\!]_{\text{in}}$ and $[\![s^\ell]\!]_{\text{out}}$ for the input and output store, respectively. Then for each statement we simply write down that the input and output variable are related by an expression of the right-hand-side of the corresponding domain transformer in Fig. 12, where the input variable is substituted for the parameter, and the output variable for the value of the function (the same could be done for all expressions, but for brevity we refer directly to the semantics of expressions in Fig. 14). Observe that in the while equations the fixed point operator is stripped, and the value of the output variable is used for the recursive reference. The iteration used to compute the analysis result using these equations will handle the fixed point in the while rule at the meta-level. The iteration starts from the bottom value of the semantic domain assigned to all flow variables (if we disregard input), and stops when a fixed point is reached. Formally, a solution to the data-flow equations is sound with respect to the derived analysis:

Theorem 4 (Soundness of data-flow analysis)**.** *For all s^ℓ, such that $[\![s^\ell]\!]_{\text{in}}$, $[\![s^\ell]\!]_{\text{out}}$ satisfies the data-flow equations:*

$$\mathcal{A}[\![s^\ell]\!]([\![s^\ell]\!]_{\text{in}}) \mathrel{\dot\sqsubseteq} [\![s^\ell]\!]_{\text{out}}$$

The resulting constant propagation analysis is the same as the data-flow analysis presented in, e.g., [6], but with one crucial difference; it has been *systematically derived* using the abstract interpretation framework, resulting in a *provably sound analysis*.

4. Systematic Derivation of Analyses for SPLs

We are now ready to discuss how the analysis obtained in Sect. 3 can be effectively lifted to work on Software Product Lines—the *variational abstract interpretation* for systematic derivation of analyses for SPLs. Figure 15 presents and relates the abstract interpretation of single programs and program families. The bottom part of the figure shows the derivation process for single programs presented in Sect. 3 (see also Fig. 3). The top part shows the same derivation process only lifted to work on SPLs. This top line of the workflow requires deriving the collecting semantics for the language with variability (in our example $\overline{\text{IMP}}$), and repeating the same abstraction steps as before at the level of program families. However, if we did this, we would almost completely ignore the artifacts accumulated during creation of the single program analysis! The core idea of the *variational abstract interpretation* is that the analyses at the single program level can be systematically lifted to

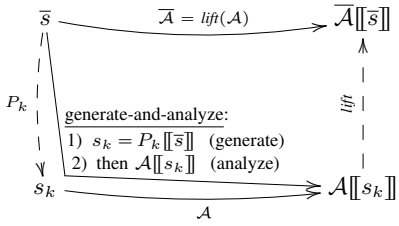


Figure 16. Generate-and-analyze vs. lifted analysis.

work on the family level without rerunning the entire derivation process: you arrive at the same, provably sound lifted analysis by commutation of the diagram.

The final constant propagation $\mathcal{A}$ can be lifted to family-based constant propagation $\overline{\mathcal{A}}$ by applying a lifting combinator (*lift*) to $\mathcal{A}$ and performing simplifying calculations. In the following, we discuss how this is done in detail and obtain a correctness result. We show how the domains of analyses, the analyses themselves (the transfer functions), and the Galois connections are lifted to the family level. Two kinds of upward arrows (dashed and dotted) lift us from the single program world to the program family world in Fig.15. There is a dashed upward arrow for lifting *analyses*, e.g. $\mathcal{A}[\![s]\!] : \mathbb{A} \to \mathbb{A}$ is lifted to $\overline{\mathcal{A}}[\![\overline{s}]\!] : (\mathbb{A} \to \mathbb{A})^{\mathbb{K}}$; and a dotted upward arrow for lifting *Galois connections*: $\mathbb{C} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathbb{B}$ is *lifted* to $\overline{\mathbb{C}} \underset{lift(\alpha)}{\overset{lift(\gamma)}{\leftrightarrows}} \overline{\mathbb{B}}$. In the following we explain the meaning and interaction of these arrows.

4.1 Lifting Domains

We first lift the semantic domains. Recall that $\mathbb{K}$ denotes a finite set of valid configurations. A domain, $(\mathbb{C}, \sqsubseteq)$, is lifted to a *variability domain*, $(\overline{\mathbb{C}}, \dot\sqsubseteq)$, by taking $\overline{\mathbb{C}}$ to be $\mathbb{C}^{\mathbb{K}}$ (i.e., a tuple of $|\mathbb{K}|$ copies of $\mathbb{C}$, one for each valid configuration), and lifting the ordering $\dot\sqsubseteq$ configuration-wise; i.e., $\overline{c} \mathrel{\dot\sqsubseteq} \overline{c}' \equiv_{def}$ for all $k \in \mathbb{K} : \pi_k(\overline{c}) \sqsubseteq \pi_k(\overline{c}')$, where π_k selects the k^{th} component of a tuple.

4.2 Lifting Analyses

The lifted domain representation, $\overline{\mathbb{A}} = \mathbb{A}^{\mathbb{K}}$, and Fig. 15 suggest that the lifted analysis, $\overline{\mathcal{A}}$, should be one complex function from $\mathbb{A}^{\mathbb{K}}$ to $\mathbb{A}^{\mathbb{K}}$. However, it turns out that using a tuple of $|\mathbb{K}|$ independent simple functions, $(\mathbb{A} \to \mathbb{A})^{\mathbb{K}}$, is a much better alternative. This models our intuition that lifting corresponds to running $|\mathbb{K}|$ analyses in parallel. Functions of type $(\mathbb{A} \to \mathbb{A})^{\mathbb{K}}$ are essentially a well behaved subset of functions from $\mathbb{A}^{\mathbb{K}}$ to $\mathbb{A}^{\mathbb{K}}$—namely those, for which the k^{th} component of the function value only depends on the k^{th} component of the argument. This warrants no problems with interference between configurations, which is critical for correctness of lifting.

To help readability, we introduce notational conventions that allow using tuples of functions, as if they were functions on tuples. We admit direct application of tuples of functions to tuples of arguments: if $\overline{f} : (\mathbb{A} \to \mathbb{A})^{\mathbb{K}}$ is a tuple of functions indexed by of $\mathbb{K}$, we write $\overline{f}(\overline{a})$ to mean the tuple of $|\mathbb{K}|$ values created by applying each function to the corresponding argument in the tuple of arguments: $\prod_{k\in K} \pi_k(\overline{f})(\pi_k(\overline{a}))$. Similarly, we overload the λ-abstraction notation, so creating a tuple of functions looks like creating a function on tuples: we write $\lambda\overline{a}.\prod_{k\in\mathbb{K}} f(\pi_k(\overline{a}))$ to mean $\prod_{k\in\mathbb{K}} \lambda a_k.f(a_k)$.

The straightforward way of analyzing a configuration, k, of an SPL, $\overline{s}$, using a conventional single-program analysis, $\mathcal{A}$, is to first *generate* product, $s_k = P_k[\![\overline{s}]\!]$, using the preprocessor; then, *analyze*

$$
\begin{aligned}
&\mathit{lift}(\mathcal{A})[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!] \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}[\![P[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!]_k]\!](\pi_k(\overline{a})) && \text{(by def. of } \mathit{lift}) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}[\![\texttt{if } e \texttt{ then } P[\![s_0]\!]_k \texttt{ else } P[\![s_1]\!]_k]\!](\pi_k(\overline{a})) && \text{(by def. of } P) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} (\mathcal{A}[\![P[\![s_0]\!]_k]\!] \mathbin{\ddot{\cup}} \mathcal{A}[\![P[\![s_1]\!]_k]\!])(\pi_k(\overline{a})) && \text{(by def. of } \mathcal{A}) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}[\![P[\![s_0]\!]_k]\!](\pi_k(\overline{a})) \mathbin{\dot{\cup}} \mathcal{A}[\![P[\![s_1]\!]_k]\!](\pi_k(\overline{a})) && \text{(by def. of } \ddot{\cup}) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \pi_k(\overline{\mathcal{A}}[\![s_0]\!]\overline{a}) \mathbin{\dot{\cup}} \pi_k(\overline{\mathcal{A}}[\![s_1]\!]\overline{a}) && \text{(by inductive hypothesis, twice)} \\
&= \lambda\overline{a}.\ \overline{\mathcal{A}}[\![s_0]\!]\overline{a} \mathbin{\ddot{\cup}} \overline{\mathcal{A}}[\![s_1]\!]\overline{a} && \text{(by def. of } \ddot{\cup}) \\
&= \overline{\mathcal{A}}[\![s_0]\!] \mathbin{\ddot{\cup}} \overline{\mathcal{A}}[\![s_1]\!] && (\eta\text{-reduce}) \\
&= \overline{\mathcal{A}}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!] && \text{(by def. of } \overline{\mathcal{A}})
\end{aligned}
$$

Figure 17. Deriving lifted constant propagation, $\overline{\mathcal{A}} = \mathit{lift}(\mathcal{A})$, for conditional statements: $\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1$.

the generated product, s_k, using the conventional analysis: $\mathcal{A}[\![s_k]\!]$. This two stage process is depicted in Fig. 16 (cf. arrow labeled *generate-and-analyze*). However, it only analyzes *one* configuration of the SPL (the arrow ends up at the bottom part of Fig. 16).

To lift the analysis to the family level, we need to execute $\mathcal{A}$ for each of the valid configurations. Simply applying an analysis to *all* configurations, yields the formal specification of the lifting combinator for analyses. If $\mathcal{A}[\![s]\!] : \mathbb{A} \to \mathbb{A}$ is a single analysis function, then we require that its lifted version $\overline{\mathcal{A}}[\![\overline{s}]\!] : (\mathbb{A} \to \mathbb{A})^{\mathbb{K}}$ satisfies the following:

$$\overline{\mathcal{A}}[\![\overline{s}]\!] = \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}[\![P[\![\overline{s}]\!]_k]\!](\pi_k(\overline{a})) \tag{2}$$

The equation stipulates that running the aggregate analysis $\overline{\mathcal{A}}$ must be equivalent to running the original analysis $\mathcal{A}$ for each variant separately, after deriving it using the preprocessor P. An analysis $\overline{\mathcal{A}}$ satisfying (2) transforms a lifted store, $\overline{a} \in \overline{\mathbb{A}} = \mathbb{A}^{\mathbb{K}}$, into another lifted store, $\overline{a}' = \prod_{k\in\mathbb{K}} \mathcal{A}[\![P[\![\overline{s}]\!]_k]\!]\pi_k(\overline{a})$, of the same type. In other words, $\overline{\mathcal{A}}$ is a transformer between aggregated state of all configurations on entry to a given program point to a set of aggregated states of all configurations on the exit from that point.

This specification of lifting works for any single program analysis, not just for constant propagation. We formulate it as a general analysis-independent and language-independent combinator.

Definition 5. *The generic lifting of analysis, $\mathcal{X} : \mathbb{X} \to \mathbb{X}$, working on domain $\mathbb{X}$, is:*

$$\mathit{lift}(\mathcal{X})[\![\overline{s}]\!] \quad = \quad \lambda\overline{x}.\ \prod_{k\in\mathbb{K}} \mathcal{X}[\![P[\![\overline{s}]\!]_k]\!](\pi_k(\overline{x}))$$

In Fig. 15 the dashed upward arrows represent applications of the above lifting combinator. They transform an analysis function (solid loop arrows at the bottom of the figure), to a *family-based* analysis (solid loop arrows at the top).

Unfortunately, Def. 5 cannot be used as a direct definition of analysis $\overline{\mathcal{A}}$ as it still depends on the single program analysis. Implementing $\overline{\mathcal{A}}$ naively, directly following (2), would merely apply the conventional analysis $|\mathbb{K}|$ times (one for each $k \in \mathbb{K}$). While this would give the correct results, it is not what we wanted! We seek an analysis that will analyse all configurations simultaneously. The question is how to obtain a definition of $\overline{\mathcal{A}}$ that is independent of $\mathcal{A}$, yet satisfies equation (2). To achieve this we simplify equation (2), similarly to how we simplified the composition of analysis functions with Galois connections. As such, our lifting is calculational in nature, following the natural steps in abstract interpretation. If

$$
\begin{aligned}
\overline{\mathcal{A}}[\![\texttt{skip}]\!] &= \lambda\overline{a}.\ \overline{a} \\
\overline{\mathcal{A}}[\![\texttt{x := } e]\!] &= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} (\pi_k(\overline{a}))[\texttt{x} \mapsto \pi_k(\overline{\mathcal{A}'}[\![e]\!]\overline{a})] \\
\overline{\mathcal{A}}[\![s_0 \texttt{ ; } s_1]\!] &= \overline{\mathcal{A}}[\![s_1]\!] \circ \overline{\mathcal{A}}[\![s_0]\!] \\
\overline{\mathcal{A}}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!] &= \overline{\mathcal{A}}[\![s_0]\!] \mathbin{\ddot{\cup}} \overline{\mathcal{A}}[\![s_1]\!] \\
\overline{\mathcal{A}}[\![\texttt{while } e \texttt{ do } s]\!] &= \mathrm{lfp}\,\lambda\overline{\Phi}.\ \lambda\overline{a}.\ \overline{a} \mathbin{\ddot{\cup}} \overline{\Phi}(\overline{\mathcal{A}}[\![s]\!]\,\overline{a}) \\
\\
\overline{\mathcal{A}}[\![\texttt{\#if } \varphi\ s]\!] &= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \begin{cases} \pi_k(\overline{\mathcal{A}}[\![s]\!]\overline{a}) & k \vDash \varphi \\ \pi_k(\overline{a}) & k \nvDash \varphi \end{cases} \\
\\
\overline{\mathcal{A}'}[\![n]\!] &= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathtt{n} \\
\overline{\mathcal{A}'}[\![\texttt{x}]\!] &= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \pi_k(\overline{a})(\texttt{x}) \\
\overline{\mathcal{A}'}[\![e_0 \oplus e_1]\!] &= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \pi_k(\overline{\mathcal{A}'}[\![e_0]\!]\overline{a}) \mathbin{\hat{\oplus}} \pi_k(\overline{\mathcal{A}'}[\![e_1]\!]\overline{a})
\end{aligned}
$$

Figure 18. Lifted constant propagation analysis of $\overline{\text{IMP}}$ where, $\overline{\mathcal{A}}[\![\overline{s}]\!] : ((\mathit{Var} \to \mathit{Const}) \to (\mathit{Var} \to \mathit{Const}))^{\mathbb{K}}$ and $\overline{\mathcal{A}'}[\![\overline{e}]\!] : ((\mathit{Var} \to \mathit{Const}) \to \mathit{Const})^{\mathbb{K}}$.

$$
\begin{aligned}
[\![\texttt{skip}^{\ell}]\!]_{\overline{\texttt{out}}} &= [\![\texttt{skip}^{\ell}]\!]_{\overline{\texttt{in}}} \\
\underline{\forall k \in \mathbb{K}}\text{: } \pi_k([\![\texttt{x :=}^{\ell} e^{\ell_0}]\!]_{\overline{\texttt{out}}}) &= \pi_k([\![\texttt{x :=}^{\ell} e^{\ell_0}]\!]_{\overline{\texttt{in}}})[\texttt{x} \mapsto \pi_k(\overline{\mathcal{A}'}[\![e^{\ell_0}]\!][\![\texttt{x :=}^{\ell} e^{\ell_0}]\!]_{\overline{\texttt{in}}})] \\
\\
[\![s_0^{\ell_0} \texttt{ ;}^{\ell} s_1^{\ell_1}]\!]_{\overline{\texttt{out}}} &= [\![s_1^{\ell_1}]\!]_{\overline{\texttt{out}}} \\
[\![s_1^{\ell_1}]\!]_{\overline{\texttt{in}}} &= [\![s_0^{\ell_0}]\!]_{\overline{\texttt{out}}} \\
[\![s_0^{\ell_0}]\!]_{\overline{\texttt{in}}} &= [\![s_0^{\ell_0} \texttt{ ;}^{\ell} s_1^{\ell_1}]\!]_{\overline{\texttt{in}}} \\
\\
[\![\texttt{if}^{\ell} e \texttt{ then} s_0^{\ell_0} \texttt{ else} s_1^{\ell_1}]\!]_{\overline{\texttt{out}}} &= [\![s_0^{\ell_0}]\!]_{\overline{\texttt{out}}} \mathbin{\ddot{\cup}} [\![s_1^{\ell_1}]\!]_{\overline{\texttt{out}}} \\
[\![s_0^{\ell_0}]\!]_{\overline{\texttt{in}}} &= [\![\texttt{if}^{\ell} e \texttt{ then } s_0^{\ell_0} \texttt{ else } s_1^{\ell_1}]\!]_{\overline{\texttt{in}}} \\
[\![s_1^{\ell_1}]\!]_{\overline{\texttt{in}}} &= [\![\texttt{if}^{\ell} e \texttt{ then } s_0^{\ell_0} \texttt{ else } s_1^{\ell_1}]\!]_{\overline{\texttt{in}}} \\
\\
[\![\texttt{while}^{\ell} e \texttt{ do } s^{\ell_0}]\!]_{\overline{\texttt{out}}} &= [\![s^{\ell_0}]\!]_{\overline{\texttt{in}}} \\
[\![s^{\ell_0}]\!]_{\overline{\texttt{in}}} &= [\![\texttt{while}^{\ell} e \texttt{ do } s^{\ell_0}]\!]_{\overline{\texttt{in}}} \mathbin{\ddot{\cup}} [\![s^{\ell_0}]\!]_{\overline{\texttt{out}}} \\
\\
\underline{\forall k \in \mathbb{K}}\text{: } \pi_k([\![\texttt{\#if}^{\ell} \varphi\ s^{\ell_0}]\!]_{\overline{\texttt{out}}}) &= \pi_k([\![s^{\ell_0}]\!]_{\overline{\texttt{out}}}) \quad \text{if} \quad k \vDash \varphi \\
\underline{\forall k \in \mathbb{K}}\text{: } \pi_k([\![\texttt{\#if}^{\ell} \varphi\ s^{\ell_0}]\!]_{\overline{\texttt{out}}}) &= \pi_k([\![\texttt{\#if}^{\ell} \varphi\ s^{\ell_0}]\!]_{\overline{\texttt{in}}}) \quad \text{if} \quad k \nvDash \varphi \\
\underline{\forall k \in \mathbb{K}}\text{: } \pi_k([\![s^{\ell_0}]\!]_{\overline{\texttt{in}}}) &= \pi_k([\![\texttt{\#if}^{\ell} \varphi\ s^{\ell_0}]\!]_{\overline{\texttt{in}}}) \quad \text{if} \quad k \vDash \varphi
\end{aligned}
$$

Figure 19. Flow equations for lifted constant propagation of Fig. 18.

we perform the composition and simplify the resulting expression systematically, we can eliminate the intermediate product generation step and obtain a direct expression as shown in Fig. 18 (corresponding to the top arrow in Fig. 16). It is essential to emphasize that this calculation, when completed for all cases, actually proves a theorem that the analyses specified in equation (2) and in Fig. 18 are the same:

Theorem 6. *The lifting of the constant propagation analysis is correct in the sense of requirement* (2)*, so $\mathit{lift}(\mathcal{A}) = \overline{\mathcal{A}}$.*

The equality sign in this theorem captures that lifting has introduced no approximation: the family-based analyses obtained this way are as precise as running the original analysis for each configuration individually. In Appendix B we prove this theorem for all three semantics. Here we briefly discuss it for constant propagation. Figure 17 illustrates how the calculation is done for conditional statements. In the fifth step we use the inductive hypothesis, which here means applying the definition of *lift* in the reverse direction to structurally smaller statements. Note that the

pointwise join operator $\dot\sqcup$ defined in Section 3 is lifted to a join over tuples, $\ddot\sqcup$, defined as $\overline{a}_0 \ddot\sqcup \overline{a}_1 = \prod_{k\in\mathbb{K}} \pi_k(\overline{a}_0) \dot\sqcup \pi_k(\overline{a}_1)$. The operator is then further lifted pointwise to functions, $\ddot\sqcup$, as well.

The calculation looks similar for most other statements. The while-case is however non-trivial, due to the need of lifting the fixed point expressions. In particular, rather than to lift a fixed point computation to a tuple of fixed point computations, we wish to equate two fixed points. To do so, we define an abstraction, which projects a particular k configuration entry, along with the corresponding concretization function.

$$\alpha_k : \mathbb{A}^{\mathbb{K}} \to \mathbb{A} \qquad \gamma_k : \mathbb{A} \to \mathbb{A}^{\mathbb{K}}$$

$$\alpha_k(\overline{a}) = \pi_k(\overline{a}) \qquad \gamma_k(a) = \prod_{k'\in\mathbb{K}} \begin{cases} a & k = k' \\ \dot\top & k \neq k' \end{cases}$$

Such projecting abstractions are well known to be Galois connections. We then lift this Galois connection to a Galois connection between monotone transfer functions [14]:

$$\alpha_{\to} : (\mathbb{A}^{\mathbb{K}} \xrightarrow{m} \mathbb{A}^{\mathbb{K}}) \to \mathbb{A} \xrightarrow{m} \mathbb{A} \qquad \gamma_{\to} : (\mathbb{A} \xrightarrow{m} \mathbb{A}) \to \mathbb{A}^{\mathbb{K}} \xrightarrow{m} \mathbb{A}^{\mathbb{K}}$$

$$\alpha_{\to}(\overline{\Phi}) = \alpha_k \circ \overline{\Phi} \circ \gamma_k \qquad \gamma_{\to}(\Phi) = \gamma_k \circ \Phi \circ \alpha_k$$

where we write $X \xrightarrow{m} Y$ for the domain of monotone function from X to Y. Then we show that

$$\alpha_{\to} \circ (\lambda\overline{\Phi}.\, \lambda\overline{a}.\, \overline{a} \ddot\sqcup \overline{\Phi}(\overline{\mathcal{A}}[\![s]\!]\overline{a})) = (\lambda\Phi.\, \lambda a.\, a \dot\sqcup \Phi(\mathcal{A}[\![P[\![s]\!]_k]\!]a)) \circ \alpha_{\to}$$

which we can use to transfer fixed points without needless approximation, using the stronger fixed point theorem (see Sec. 3).

$$\begin{aligned}
&\alpha_{\to}(\overline{\mathcal{A}}[\![P[\![\texttt{while}\ e\ \texttt{do}\ \overline{s}]\!]_k]\!]) \\
&= \alpha_{\to}(\overline{\mathcal{A}}[\![\texttt{while}\ e\ \texttt{do}\ P[\![\overline{s}]\!]_k]\!]) && \text{(by def. of } P\text{)} \\
&= \mathcal{A}[\![\texttt{while}\ e\ \texttt{do}\ P[\![\overline{s}]\!]_k]\!] && \text{(by above and stronger fixed point thm.)} \\
&= \mathcal{A}[\![P[\![\texttt{while}\ e\ \texttt{do}\ \overline{s}]\!]_k]\!] && \text{(by def. of } P\text{)}
\end{aligned}$$

We can now calculate the closed form for while loops in the same style as for conditional statements (using the above law in one of the rewrite steps). This proof method is independent of the transfer function (here $\mathcal{A}$). In Appendix H and I we use it to lift fixed points for the other two semantics.

The resulting formulation in Fig. 18 no longer depends on $\mathcal{A}$, but specifies $\overline{\mathcal{A}}$ directly. Just like in Sect. 3, we can use this formulation to derive data flow equations. This is a fairly mechanical process that results in the equations of Fig. 19 (compare to figures 18, 14). As shown in previous work [5, 6], this simplified version can be implemented to run much faster than the naive approach. The obtained data-flow equations are now variability aware and provably sound:

Theorem 7. *(Soundness of lifted data-flow analysis) For all $\overline{s}^{\ell}$, such that $[\![\overline{s}^{\ell}]\!]_{\overline{\mathsf{in}}}, [\![s^{\ell}]\!]_{\overline{\mathsf{out}}}$ satisfies the data-flow equations:*

$$\overline{\mathcal{A}}[\![s^{\ell}]\!]([\![s^{\ell}]\!]_{\overline{\mathsf{in}}}) \ \ddot\sqsubseteq\ [\![s^{\ell}]\!]_{\overline{\mathsf{out}}}$$

An attentive reader may question if we obtained any aggregate analysis here. After all, the specification in Fig. 19 appears to be exponential in the size of the valid configurations. It is crucial to understand that this mathematical specification for computing aggregate analysis, is orthogonal to an implementation (including choices of data structures). In particular for SPL analysis in practice, many of the entries in the $\mathbb{K}$-indexed tuples and transfer functions will be identical (many program points are identical/act identically for most configurations). Thus they can be executed and represented efficiently, storing and running them once, instead of exponentially many times. This is also the reason why the analyses in [5, 6] are so efficient in practice.

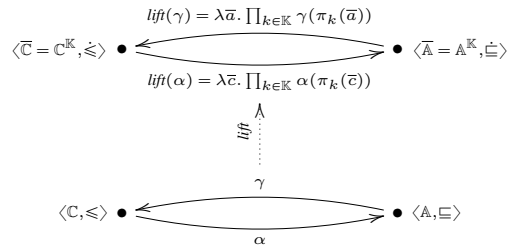


Figure 20. Pointwise lifting of a Galois connection.

4.3 Many Routes to Family-based Analysis

If we wanted to consider correctness of the lifted analysis $\overline{\mathcal{A}}$ using the classical abstract interpretation approach, we should devise a collecting semantics $\overline{\mathcal{C}}$ and a Galois connection relating them. If we wanted to follow the same incremental process as in Sect. 3, then we would need a chain of Galois connections:

$$\langle \overline{\mathbb{C}}, \dot\subseteq \rangle \underset{\overline{\alpha_{\mathrm{CB}}}}{\overset{\overline{\gamma_{\mathrm{BC}}}}{\leftrightarrows}} \langle \overline{\mathbb{B}}, \ddot\subseteq \rangle \underset{\overline{\alpha_{\mathrm{BA}}}}{\overset{\overline{\gamma_{\mathrm{AB}}}}{\leftrightarrows}} \langle \overline{\mathbb{A}}, \ddot\sqsubseteq \rangle$$

Then, we would have to compute $\overline{\mathcal{A}}$ by composing these Galois connections with $\overline{\mathcal{C}}$ and prove that the resulting analysis is identical to the lifting of $\mathcal{A}$, so that the diagram in Fig. 15 commutes. A detailed development taking this route is available in the extended version [39]. But there is an easier route as we have just seen! Instead of devising the collecting semantics at family level, $\overline{\mathcal{C}}$, and then a sequence of Galois connections, we can obtain them all by lifting. The transfer functions, including the collecting semantics, can be lifted like in Sect. 4.3 (see [39]). We can lift the Galois connections using a third combinator:

$$\mathit{lift}(\varphi) \quad = \quad \lambda\overline{d}.\, \prod_{k\in\mathbb{K}} \varphi(\pi_k(\overline{d})) \tag{3}$$

Figure 20 illustrates the lifting of a Galois connection. By viewing *lift* (over a tuple) as a pointwise lifting (to configuration accepting functions) this is a well-known lifting of Galois connections [15]. This way, no invention of new analyses for the family level is needed. Instead, all analyses can be uniformly lifted and composed. This is by no means automatic, but it is systematic; it does not require any design effort, as the original analysis is a sufficient source of information for obtaining the family-based analysis.

The following theorem states that the result of lifting the final single-program analysis is equivalent to lifting and recalculating all intermediate steps. This result does not depend on any particular analysis. It states that if a static analysis $\mathcal{X}[\![s]\!]$ is obtained from a more concrete analysis $\mathcal{Y}[\![s]\!]$ by applying a Galois connection and simplifying (possibly with some approximation), then the lifting of this analysis can be soundly obtained by applying a lifted Galois connection to the lifting of $\mathcal{Y}$. Effectively, the diagram of Fig. 15 commutes. It is sound to develop the single program analysis and lift it as in Sect. 4.3, instead of lifting the collecting semantics and developing the entire analysis anew at the family level.

Theorem 8. *If for all programs s we have that $\alpha \circ \mathcal{Y}[\![s]\!] \circ \gamma \ \dot\sqsubseteq\ \mathcal{X}[\![s]\!]$ then also for each program $\overline{s}$ with variability*

$$\mathit{lift}(\alpha) \circ \mathit{lift}(\mathcal{Y})[\![\overline{s}]\!] \circ \mathit{lift}(\gamma) \ \ddot\sqsubseteq\ \mathit{lift}(\mathcal{X})[\![\overline{s}]\!]$$

Moreover, if no approximation is introduced during the derivation of a single program analysis $\mathcal{X}$ (so that $\alpha \circ \mathcal{Y}[\![s]\!] = \mathcal{X}[\![s]\!] \circ \alpha$) then the lifting introduces no additional abstraction at the family level: $\mathit{lift}(\alpha) \circ \mathit{lift}(\mathcal{Y})[\![\overline{s}]\!] = \mathit{lift}(X)[\![\overline{s}]\!] \circ \mathit{lift}(\alpha)$. With this general theorem, the soundness for the example analysis now follows as a corollary from Thm. 2, 3, 6 and 8:

Corollary 9 (Soundness). *For all* $\overline{s} \in \overline{Stm}$*:*

$$lift(\alpha_{BA} \circ \alpha_{CB}) \circ lift(C)[\![\overline{s}]\!] \circ lift(\gamma_{BC} \circ \gamma_{AB}) \ddot{\sqsubseteq} lift(\mathcal{A})[\![\overline{s}]\!] = \overline{\mathcal{A}}[\![\overline{s}]\!]$$

4.4 Variability Relevant Abstractions

So far, we have argued that it is most practical to develop analyses for single programs, and then apply our lifting combinator to lift their definition to program families via a formal calculation. This process appears most straightforward, but it has one disadvantage: all the abstractions applied in the derivation of a single program analysis are unaware of variability. This way it is impossible to abstract over variability, which could sometimes be beneficial. For example, when the configuration space is too large, it may be difficult or impossible to represent lifted stores symbolically, so that they take little space in memory. Variability abstractions can only be applied at the family level: one needs an analysis formulated at the family level and then apply the variability aware abstraction to it, in the very same way as we applied usual abstractions on the single program level in Sect. 3.

Variability-aware abstractions can be plentiful. In this section we show one example: an abstraction that ignores a certain subset of features, presumably meant to have insignificant impact on the analysis results. Let $F \subset \mathbb{F}$ be a set of features that we deem relevant for the analysis. Then if $k \in \mathbb{K}$ is a valid configuration, $k \cap F$ is a simplification of this configuration to relevant features only. Let $\mathbb{K}_F$ be the set of valid configurations over relevant features (so $\mathbb{K}_F = \{k \cap F \mid k \in \mathbb{K}\}$). Let $\langle \mathbb{X}, \sqsubseteq \rangle$ stand for *any* complete lattice domain, which is lifted as usual, so $\overline{\mathbb{X}} = \mathbb{X}^{\mathbb{K}}$. We write $\overline{\mathbb{X}}_F$ for lifting $\mathbb{X}$ to the set of valid configurations over only the relevant features, so $\overline{\mathbb{X}}_F = \mathbb{X}^{\mathbb{K}_F}$. Both $\langle \overline{\mathbb{X}}, \dot{\sqsubseteq} \rangle$ and $\langle \overline{\mathbb{X}}_F, \dot{\sqsubseteq} \rangle$ are complete lattices. Clearly since the latter tracks the analysis values for a smaller set of configurations, it is a more abstract domain, thereby collapsing more information. Indeed, one can formulate abstraction and concretization functions between the two lifted domains:

$$\alpha_F(\overline{x}) = \prod_{k_F \in \mathbb{K}_F} \bigsqcup_{\{k \in \mathbb{K} \mid k_F = k \cap F\}} \pi_k(\overline{x}) \tag{4}$$

$$\gamma_F(\overline{x}_F) = \prod_{k \in \mathbb{K}} \pi_{(k \cap F)}(\overline{x}_F) \tag{5}$$

It is easy to show (see [39]) that $\langle \overline{\mathbb{X}}, \dot{\sqsubseteq} \rangle \underset{\alpha_F}{\overset{\gamma_F}{\leftrightarrows}} \langle \overline{\mathbb{X}}_F, \dot{\sqsubseteq} \rangle$ is a Galois connection. This Galois connection can be composed with any family-based analysis transfer function to produce a version of the analysis that is less precise regarding the set of valid configurations. In particular, it could be composed with our constant propagation analysis $\overline{\mathcal{A}}$. In the extreme case, if we ask for an analysis that is insensitive to all features (so $F = \varnothing$), we obtain an abstracted analysis, which conservatively detects which values are constant (same) in all configurations.

In general, the process of developing an analysis, which should abstract variability, starts at a single program level (Fig. 15). We recommend developing the analysis for single programs first, and then applying lifting at a convenient intermediate step. After lifting the intermediate analysis function, one can apply a variability abstraction (for example $\langle \overline{\mathbb{X}}, \dot{\sqsubseteq} \rangle \underset{\alpha_F}{\overset{\gamma_F}{\leftrightarrows}} \langle \overline{\mathbb{X}}_F, \dot{\sqsubseteq} \rangle$ presented above) and then continue applying the liftings of the remaining abstractions (Galois connections) to develop the final analysis. In simple words: it is possible to switch the level in Fig.15 at a convenient point, where abstracting over variability is beneficial for the design.

5. The Variational Abstract Interpretation Method

Let us summarize the methodology of developing analyses of program families. The main purpose of this section is to highlight the abstract steps and results of our method independently of the IMP language. The first three steps are the traditional steps of calculational abstract interpretation:

1. Develop formal operational semantics for your language.
2. Design collecting semantics for your language. Show equivalence of the operational and collecting semantics. Steps 1–2 are often given for existing established languages.
3. Specify a series of abstractions applied to the semantics in the form of Galois connections and compose them with the collecting semantics to obtain a single program analysis. The calculation of compositions includes developing an inductive proof that the resulting analysis is sound.

Once the single program analysis is establish we set off to develop the aggregate family-based analysis:

4. Extend the syntax of the language with a preprocessor, and give semantics to the preprocessor P mapping syntactic constructs with variability to syntactic constructs without variability.
   Remark. The preprocessor may apply to all syntactic categories of the language, and the language does not need to have any particular flavour. In the example we only applied the preprocessor to statements, and IMP was an imperative language—these choices were made purely for pedagogical reasons, and are not restrictions of variational abstract interpretation.
5. Apply the lifting combinator *lift* to the analysis calculated in step 3 above.
   Remark. In the paper we only applied *lift* to transfer functions, which were endofunctions. This is not a requirement. For example, when lifting expression semantics, we had to lift functions that given a store argument produce a simple value as a result (see [39]). So variational abstract interpretation can be applied not only to languages expressing computations (state transfers), but also to others, for example constraint languages.
6. Simplify the resulting function to obtain a lifted analysis that is formulated independently of the original single-program analysis (prove theorem akin to Thm. 6)
7. Soundness of the lifted analysis at the family level now follows from from combining the calculations in steps 3,6 with Thm. 8

For large configuration spaces it may be beneficial to include a variability abstraction in the process:

1. Decide at which point in the design of single program analysis the variability abstraction should be inserted. Compose the collecting semantics and all Galois connections until this point to obtain a partially specified analysis for single programs.
2. Apply the lifting combinator to the obtained analysis, and simplify the result to obtain the partially specified lifted analysis. As before, correctness follows from combining the previous calculations and Thm. 8.
3. Lift the remaining Galois connections to program families by applying our lifting combinator for Galois connections. By property of the lifting combinator, the lifted functions form composable Galois connections between lifted domains. Lifting is the only operation necessary, no properties of Galois connections need to be re-proven.
4. Formulate the Galois connection abstracting configurations.
   Remark. You may want to use the feature abstraction specified in equations (4) and (5), which is independent of $\overline{\text{IMP}}$ and the analysis domains used in our running example.
5. Compose the lifted Galois connections with the lifted partial analysis, in order to obtain the final formulation of the lifted analysis that includes variability abstraction. Soundness of the result follows from the soundness of the calculation argument and the soundness of the partially lifted analysis.

6. Related Work

We divide our discussion of related work into five categories; *abstract interpretation*, *lifting representations*, *lifting data-flow analyses*, *lifting other analyses*, and *multi-staged program analysis*.

Abstract interpretation: Abstract interpretation is a general theory that unifies *data-flow analysis* [13], *model checking* [16, 17], *type systems* [11], *verification* [18], and *testing* [28]. Our analyses have been developed using the classical Galois connection framework [13]. In particular, we follow the calculational and compositional approach advocated by Cousot [12]. With this approach, soundness follows from a systematic derivation. Indeed, this is the case for the data-flow analysis derived in Fig. 19. This approach has previously been used by the first author to derive, for instance, iterative graph algorithms [44] and modular control-flow analyses [38].

Lifting representations: Kästner et al. [33] show how languages with preprocessor syntax can be parsed and represented in syntax trees with variability, even if the preprocessor syntax is not properly nested in the main language syntax (as it was the case for IMP). Erwig and Walkingshaw [24] present the Choice Calculus, which can be seen as a more expressive and elegant version of a preprocessor with a fixed and well-defined semantics. It would be interesting to develop variational abstract interpretation further, to support richer preprocessors (like the Choice Calculus), and ill-formed preprocessor use. The former appears a rather straightforward extension, while the latter likely remains a challenge due to difficulty of defining semantics elegantly in a syntax-directed manner. One angle of attack would be to apply preprocessor normalization via rewrites as suggested by Garrido and Johnson [25, 26].

Lifting data-flow analysis: Previous work lifts data-flow analysis, resulting in *feature-sensitive* data-flow analysis [6], corresponding to our Figure 19. Lifted data-flow analyses are much faster than ones based on $|\mathbb{K}|$ runs of the naive generate-and-analyze strategy [6]. Indeed, inter-procedural application of the lifted analysis approach of SPL$^{\text{LIFT}}$ [5] achieves several orders of magnitude speed-ups through the use of BDD-based sharing of configurations and encoding of lifted transfer functions and control-flow as graphs for which the fixed-point computation can be rephrased as graph reachability. This technique works for analyses phrased within the IFDS framework [43], a subset of data-flow analyses, which can then be transparently lifted without programmer intervention. Recently, larger SPLs based on C have been analyzed [36] via lifted type checking and liveness data-flow analysis.

Lifting other analyses: Recent work [47] has surveyed analysis strategies for SPLs and proposes a taxonomy of such which would classify our lifted analyses as *family-based analyses* (whereas the *generate-and-analyze* strategy yields a *product-based analysis*).

The approaches of *type checking*, *model checking*, and *verification* are complementary to abstract interpretation and share the commendable goal of detecting errors at compile-time as opposed to at runtime. There is work on lifting all of them in an attempt to find errors at SPL compile-time as opposed to at post product-instantiation time, when a product happens to be compiled, possibly long after it has been developed: *lifted type checking* [1, 32], *lifted well-formedness checking*[20], *lifted model checking* [8, 9, 27], and *lifted verification* [2, 35, 42]. With abstract interpretation, however, analysis soundness comes for free, by derivation—and as we have shown, even at the SPL level.

Safe composition [1, 22, 32, 34, 46] is about verification and safe generation of properties for SPL assets and aims to provide guarantees that only products where certain properties are obeyed can be generated. Errors detected include type and definition-usage errors (e.g., undeclared variables, undeclared fields, and unimplemented abstract methods). We complement this with an approach based on abstract interpretation with which analyses intercepting those kinds of errors can be derived.

Multi-staged program analysis: Our work is related to *multi-staged program analysis*, analyzing "programs that generate programs", e.g., [7, 40]. In the context of Software Product Lines, however, we are in a much simpler case where the first stage is significantly more restrictive than a Turing-complete programming language and can thus be dealt with without approximation. For SPLs, our approach is simpler and sufficient; and without loss of precision on the variability level.

7. Conclusion

We have shown how compositional and systematic derivation of static analyses based on abstract interpretation can be lifted to Software Product Lines. The result is variational abstract interpretation—a compositional and systematic approach for the derivation of variability-aware product line analyses, with the following distinctive components and properties:

- A scheme to lift domain types, and combinators for lifting analyses and Galois connections.
- A general soundness-by-construction result (Thm. 8), allowing to lift a formally developed analysis, without re-proving the entire abstract interpretation process. This crucially reuses all the effort invested in developing a single-program analysis, to obtain a provably sound family-based analysis.
- A possibility of incorporating abstractions that involve configuration space; including an example of one such abstraction.
- Precise control over precision of analyses (lifting does not lose any information *per se*).
- A scheme to obtain data-flow equations for family-based analyses from the abstract interpretation definition.

Variational abstract interpretation mixes language-independent and language-specific elements. The main language specific theorem (Thm. 6) needs to be proven for each new analysis. We have proven it for all the three semantics of our running example and extracted a general proof methodology presented in this paper. On the other hand, the main language-independent soundness theorem (Thm. 8) holds in general and needs not be re-proven.

Abstract interpretation is a unifying theory that allows the derivation of data-flow analyses, control-flow analyses, model checking, type systems, verification, and even testing. Hence, variational abstract interpretation tells us how to systematically obtain lifted versions of all such analyses. We believe that in this sense, variational abstract interpretation, contributes to the understanding of how variability affects analysis of programs in general.

Finally, since the lifting operator can be applied to a directly formulated analysis, we claim that the obtained insight into lifting extends beyond abstract interpretation. In particular, the *lift* combinator can be applied to analyses developed in an ad hoc process, without abstract interpretation, but represented as transfer functions (soundness of such lifting requires a separate argument though).

Acknowledgements. The authors thank Hans Erik Bugge Grathwohl and Aleksandar Dimovski for fruitful technical discussions.

References

[1] S. Apel, C. Kästner, A. Grösslinger, and C. Lengauer. Type safety for feature-oriented product lines. *Automated Software Engineering*, 17: 251–300, September 2010.

[2] S. Apel, H. Speidel, P. Wendler, A. von Rhein, and D. Beyer. Detection of feature interactions using feature-aware verification. In *ASE'11*, Lawrence, USA, November 2011. IEEE Computer Society.

[3] D. Batory. Feature models, grammars, and propositional formulas. In *9th International Software Product Lines Conference*, volume 3714 of *LNCS*, pages 7–20. Springer-Verlag, 2005.

[4] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki. Variability modeling in the real: a perspective from the operating systems domain. In *ASE'10*, pages 73–82, 2010.

[5] E. Bodden, T. Tolêdo, M. Ribeiro, C. Brabrand, P. Borba, and M. Mezini. SPL$^{\text{LIFT}}$ - statically analyzing software product lines in minutes instead of years. In *PLDI'13*, 2013.

[6] C. Brabrand, M. Ribeiro, T. Tolêdo, J. Winther, and P. Borba. Intraprocedural dataflow analysis for software product lines. *Transactions on Aspect-Oriented Software Development*, 10:73–108, 2013. Earlier version in AOSD 2012.

[7] W. Choi, B. Aktemur, K. Yi, and M. Tatsuta. Static analysis of multi-staged programs via unstaging translation. *SIGPLAN Not.*, 46(1):81–92, Jan. 2011.

[8] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, and J.-F. Raskin. Model checking lots of systems: efficient verification of temporal properties in software product lines. In *ICSE'10*, pages 335–344. ACM, 2010.

[9] A. Classen, P. Heymans, P.-Y. Schobbens, and A. Legay. Symbolic model checking of software product lines. In *ICSE'11*, pages 321–330, 2011.

[10] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

[11] P. Cousot. Types as abstract interpretations. In *POPL'97*, pages 316–331, 1997.

[12] P. Cousot. The calculational design of a generic abstract interpreter. In M. Broy and R. Steinbrüggen, editors, *Calculational System Design*. NATO ASI Series F. IOS Press, Amsterdam, 1999.

[13] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *POPL'79*, pages 269–282, 1979.

[14] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3):103–179, 1992.

[15] P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages), invited paper. In *ICCL'94*, pages 95–112, Toulouse, France, May 1994.

[16] P. Cousot and R. Cousot. Refining model checking by abstract interpretation. *Autom. Softw. Eng.*, 6(1):69–95, 1999.

[17] P. Cousot and R. Cousot. Temporal abstract interpretation. In *POPL'00*, pages 12–25, 2000.

[18] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The Astreé analyzer. In *ESOP'05*, pages 21–30, 2005.

[19] K. Czarnecki and U. Eisenecker. *Generative programming: methods, tools, and applications*. Addison-Wesley, 2000.

[20] K. Czarnecki and K. Pietroszek. Verifying feature-based model templates against well-formedness ocl constraints. In *GPCE'06*, pages 211–220, New York, NY, USA, 2006. ACM.

[21] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A. Wasowski. Cool features and tough decisions: a comparison of variability modeling approaches. In *VaMoS'12*, pages 173–182, 2012.

[22] B. Delaware, W. R. Cook, and D. Batory. Fitting the pieces together: a machine-checked model of safe composition. In *ESEC/FSE'09*, pages 243–252, New York, NY, USA, 2009. ACM.

[23] B. Delaware, W. Cook, and D. Batory. Product lines of theorems. In *OOPSLA'11*, pages 595–608, New York, NY, USA, 2011. ACM.

[24] M. Erwig and E. Walkingshaw. The choice calculus: A representation for software variation. *ACM Trans. Softw. Eng. Methodol.*, 21(1):6:1–6:27, Dec. 2011.

[25] A. Garrido and R. E. Johnson. Refactoring C with conditional compilation. In *ASE'03*, pages 323–326. IEEE Computer Society, 2003. ISBN 0-7695-2035-9.

[26] A. Garrido and R. E. Johnson. Analyzing multiple configurations of a C program. In *ICSM'05*, pages 379–388. IEEE Computer Society, 2005. ISBN 0-7695-2368-4.

[27] A. Gruler, M. Leucker, and K. D. Scheidemann. Modeling and model checking software product lines. In *FMOODS'08*, pages 113–131, 2008.

[28] D. Guilbaud, E. Goubault, A. Pacalet, and B. S. F. Védrine. A simple abstract interpreter for threat detection and test case generation. In *WAPATV'01, with ICSE'01*, Toronto, 2001.

[29] IBM, Thales, F. FOKUS, and TCS. *Proposal for Common Variability Language (CVL) Revised Submission*, 2012.

[30] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[31] C. Kästner. *Virtual Separation of Concerns: Toward Preprocessors 2.0*. PhD thesis, University of Magdeburg, Germany, May 2010.

[32] C. Kästner and S. Apel. Type-checking software product lines - a formal approach. In *ASE'08*, pages 258–267, L'Aquila, Italy, 2008.

[33] C. Kästner, P. G. Giarrusso, T. Rendel, S. Erdweg, K. Ostermann, and T. Berger. Variability-aware parsing in the presence of lexical macros and conditional compilation. In *OOPSLA'11*, pages 805–824, Portland, OR, USA, 2011. ACM.

[34] C. Kästner, S. Apel, T. Thüm, and G. Saake. Type checking annotation-based product lines. *ACM Trans. Softw. Eng. Methodol.*, 21(3):14:1–14:39, July 2012.

[35] C. H. P. Kim, E. Bodden, D. Batory, and S. Khurshid. Reducing configurations to monitor in a software product line. In *1st International Conference on Runtime Verification (RV)*, volume 6418 of *LNCS*, Malta, November 2010. Springer.

[36] J. Liebig, A. von Rhein, C. Kästner, S. Apel, J. Dörre, and C. Lengauer. Scalable analysis of variable software. In *ESEC/FSE'13*, pages 81–91, New York, NY, 8 2013.

[37] J. Midtgaard and T. Jensen. A calculational approach to control-flow analysis by abstract interpretation. In *SAS'08*, volume 5079 of *LNCS*, pages 347–362, Valencia, Spain, July 2008. Springer-Verlag.

[38] J. Midtgaard, M. D. Adams, and M. Might. A structural soundness proof for Shivers's escape technique: A case for Galois connections. In *SAS'12*, volume 7460 of *LNCS*, pages 352–369, Deauville, France, Sept. 2011. Springer-Verlag.

[39] J. Midtgaard, C. Brabrand, and A. Wasowski. Systematic derivation of static analyses for software product lines. Technical Report TR-2014-170, IT University of Copenhagen, 2014.

[40] F. Nielson and H. R. Nielson. *Two-Level Functional Languages*. Cambridge Tracts in Theoretical Computer Science, vol. 34. Cambridge University Press, 1992.

[41] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag, Secaucus, USA, 1999.

[42] H. Post and C. Sinz. Configuration lifting: Verification meets software configuration. In *ASE'08*, pages 347–350, L´Aquila, Italy, 2008. IEEE Computer Society.

[43] T. Reps, S. Horwitz, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *POPL'95*, pages 49–61, New York, NY, USA, 1995. ACM.

[44] I. Sergey, J. Midtgaard, and D. Clarke. Calculating graph algorithms for dominance and shortest path. In *MPC'12*, volume 7342 of *LNCS*, pages 132–156, Madrid, Spain, June 2012. Springer.

[45] T. Stahl, M. Voelter, and K. Czarnecki. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.

[46] S. Thaker, D. Batory, D. Kitchin, and W. Cook. Safe composition of product lines. In *GPCE'07*, pages 95–104, New York, NY, USA, 2007. ACM.

[47] T. Thüm, S. Apel, C. Kästner, M. Kuhlemann, I. Schaefer, and G. Saake. Analysis strategies for software product lines. Technical Report FIN-004-2012, School of Computer Science, University of Magdeburg, Germany, 2012.

[48] G. Winskel. *The Formal Semantics of Programming Languages*. Foundation of Computing Series. The MIT Press, 1993.

A. Prerequisite Mathematics

A *partial order* is a mathematical structure, $\langle S, \sqsubseteq \rangle$, where S is a set equipped with a binary order relation, $\sqsubseteq$, with the properties:

$$\begin{array}{rll} \forall x \in S: & x \sqsubseteq x & \text{(reflexivity)} \\ \forall x, y, z \in S: & x \sqsubseteq y \wedge y \sqsubseteq x \implies x \sqsubseteq z & \text{(transitivity)} \\ \forall x, y \in S: & x \sqsubseteq y \wedge y \sqsubseteq x \implies x = y & \text{(anti-symmetry)} \end{array}$$

Let $X \sqsubseteq S$. We say that $u \in S$ is an *upper bound* for X, written $X \sqsubseteq u$, if we have $\forall x \in X : x \sqsubseteq u$. Similarly, $\ell \in S$ is a *lower bound* for X, written $\ell \sqsubseteq X$, if $\forall x \in X : \ell \sqsubseteq x$. A *least upper bound*, written $\sqcup X$, is defined by:

$$\forall x \in X : x \sqsubseteq \sqcup X \quad \wedge \quad \forall u \in S : X \sqsubseteq u \implies \sqcup X \sqsubseteq u$$

(Similarly a *greatest lower bound*, $\sqcap$, can be defined.) Usually, binary infix notation, $x \sqcup y$, is used whenever the operator is applied to only two elements; i.e., $x \sqcup y = \sqcup\{x, y\}$. A *complete lattice* is a partial order for which $\sqcup X$ and $\sqcap X$ exist for all subsets $X \subseteq S$. As a consequence, a complete lattice will always have a *unique largest element*, $\top$, and a *unique smallest element*, $\bot$, defined as: $\top = \sqcup S$ and $\bot = \sqcap S$. A function, $f : S \to S$, is *monotone* when $\forall x, y \in S : x \sqsubseteq y \implies f(x) \sqsubseteq f(y)$. An element, $x \in S$, is called a *fixed point* of $f : S \to S$, if $x = f(x)$.

Tarski's fixed-point theorem says that the fixed points of a monotone function, $f : S \to S$, on a complete lattice, $\langle S, \sqsubseteq \rangle$, themselves form a complete lattice. This guarantees the existence of a fixed point, and of a *unique least fixed point*, $\mathrm{lfp}(f) \in S$.

A fixed point formulated over an *infinite* height lattice is thus well defined mathematically speaking but it is not necessarily *computable*. However, when the height of a complete lattice is *finite* the fixed point may then be computed via *Kleene's fixed-point theorem*:

$$\mathrm{lfp}(f) = \sqcup_i f^i(\bot)$$

This basically says that a fixed point may be computed iteratively by computing larger and larger values of $f^i(\bot)$:

$$\bot \sqsubseteq f(\bot) \sqsubseteq f(f(\bot)) \sqsubseteq \cdots \sqsubseteq f^i(\bot) = f^{i+1}(\bot)$$

until a fixed point is reached, for some i where $f(f^i(\bot)) = f^i(\bot)$. When the lattice has finite height, we are bound to hit the fixed point eventually, for some i. This fact is often exploited in static analysis.

Note that a recursive function can easily be phrased and defined as a *least fixed point*. Suppose we want to define the factorial function $n!$ in this manner. If we extend the usual numeric ordering $\leqslant$ over natural numbers $\mathbb{N}$ to include infinity, ∞, and looping, $\bot$ (where, $\bot \leqslant n \leqslant \infty, \forall n \in \mathbb{N}$), the result forms a complete lattice $\mathbb{N}_\bot^\infty = \mathbb{N} \cup \{\bot, \infty\}$. Factorial can now be defined as a least fixed point of a monotone function over that lattice, $fac : \mathbb{N}_\bot^\infty \to \mathbb{N}_\bot^\infty$:

$$fac \quad =_{def} \quad \mathrm{lfp}\, \lambda\Phi.\, \lambda n. \begin{cases} \bot & \text{if } n = \bot \\ 1 & \text{if } n = 0 \\ n * \Phi(n-1) & \text{if } n > 0 \wedge n \neq \infty \\ \infty & \text{if } n = \infty \end{cases}$$

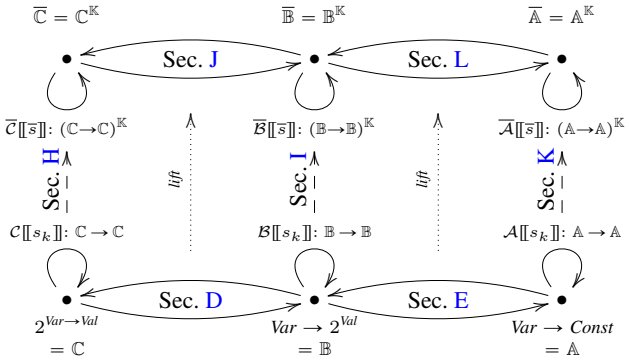
It is a simple exercise to check that the above functional is monotone by appeal to extended multiplication over $\mathbb{N}_\bot^\infty$ being monotone.

Functions defined in this way will have the general form: $\mathrm{lfp}\, \lambda\Phi.\, \lambda n.\, \cdots \Phi(\cdots) \cdots$ where n is the argument to the function being defined and $\Phi(\cdots)$ plays the role of a recursive application of the function being defined. Compare this with a *non-recursive* function with the same type signature, e.g., $succ : \mathbb{N}_\bot^\infty \to \mathbb{N}_\bot^\infty$, which can be phrased directly as: $succ \quad =_{def} \quad \lambda n. \begin{cases} \bot & n = \bot \\ n+1 & n \in \mathbb{N} \\ \infty & n = \infty \end{cases}$ (i.e.,without a fixed point).

B. Proof Overview

The rest of this appendix is structured as follows:

Section C proves the equivalence of the structural operational semantics and the collecting semantics. The remaining theorems are structured with respect to Figure 15 which we recall below, annotated with the corresponding appendix section.



The first half of the appendix is then concerned with the single-program analysis:

- Section D proves soundness of the approximate semantics wrt. the collecting semantics.
- Section E proves soundness of the constant propagation analysis wrt. the approximate semantics.
- Section F proves monotonicity for all single-program operations.
- Section G proves soundness of the data-flow equations with respect to the single-program constant propagation analysis.

The second half of the appendix is then concerned with lifting the single-program analysis to a family-based analysis:

- Section H proves the lifting of the single-program collecting semantics to a family-based collecting semantics.
- Section I proves the lifting of the single-program approximate semantics to a family-based approximate semantics.
- Section J proves soundness of the family-based approximate semantics wrt. the family-based collecting semantics.
- Section K proves the lifting of the single-program constant propagation analysis to a family-based constant propagation analysis.
- Section L proves soundness of the family-based constant propagation analysis wrt. the family-based approximate semantics.
- Section M proves monotonicity for the lifted operations.
- Section N proves soundness of the lifted data-flow equations wrt. the lifted constant propagation analysis.

Finally,

- Section O proves the generic soundness theorem 8, and
- Section P proves that the suggested variability abstraction of Section 4.4 constitutes a Galois connection.

In the simplest process of developing a lifted constant propagation one needs to follow the work done in appendices D, E, F, G and K. Only the last step, in fact, is required if your analysis is already formulated using abstract interpretation. The remaining appendices serve as evidence for other parts of the paper (general theorems, base material for extracting methodology, etc).

C. Equivalence of SOS and collecting semantics

C.1 Expression-level equivalence

Lemma 10 (Correctness of SOS and expression collecting semantics)**.**

$$\forall e \in \mathit{Exp}, \sigma \in \mathit{Store}.\ \{\mathcal{E}(e,\sigma)\} = \mathcal{C}'[\![e]\!]\{\sigma\}$$

Proof. Let $e \in \mathit{Exp}$, $\sigma \in \mathit{Store}$ be given. Proceed by structural induction on e.

Case n**:**

$$\begin{aligned}
&\mathcal{C}'[\![n]\!]\{\sigma\} \\
&= (\lambda c.\ \{n\})\{\sigma\} && \text{(by def of } \mathcal{C}') \\
&= \{n\} && (\beta\text{-reduction}) \\
&= \{\mathcal{E}(n,\sigma)\} && \text{(by def of } \mathcal{E})
\end{aligned}$$

Case `x`**:**

$$\begin{aligned}
&\mathcal{C}'[\![\mathtt{x}]\!]\{\sigma\} \\
&= (\lambda c.\ \{\sigma(\mathtt{x}) \mid \sigma \in c\})\{\sigma\} && \text{(by def of } \mathcal{C}') \\
&= \{\sigma(\mathtt{x}) \mid \sigma \in \{\sigma\}\} && (\beta\text{-reduction}) \\
&= \{\sigma(\mathtt{x})\} && \text{(simplify)} \\
&= \{\mathcal{E}(\mathtt{x},\sigma)\} && \text{(by def. of } \mathcal{E})
\end{aligned}$$

Case $e_0 \oplus e_1$**:**

$$\begin{aligned}
&\mathcal{C}'[\![e_0 \oplus e_1]\!]\{\sigma\} \\
&= (\lambda c.\ \{v'' \mid v'' \in \{v\} \mathbin{\dot\oplus} \{v'\} \wedge \sigma \in c \\
&\qquad \wedge\ v \in \mathcal{C}'[\![e_0]\!]\{\sigma\} \wedge v' \in \mathcal{C}'[\![e_1]\!]\{\sigma\}\})\{\sigma\} \\
&\qquad && \text{(by def of } \mathcal{C}') \\
&= \{v'' \mid v'' \in \{v\} \mathbin{\dot\oplus} \{v'\} \wedge \sigma \in \{\sigma\} \\
&\qquad \wedge\ v \in \mathcal{C}'[\![e_0]\!]\{\sigma\} \wedge v' \in \mathcal{C}'[\![e_1]\!]\{\sigma\}\} \\
&\qquad && (\beta\text{-reduction}) \\
&= \{v'' \mid v'' \in \{v\} \mathbin{\dot\oplus} \{v'\} \wedge v \in \mathcal{C}'[\![e_0]\!]\{\sigma\} \wedge v' \in \mathcal{C}'[\![e_1]\!]\{\sigma\}\} \\
&\qquad && \text{(simplify)} \\
&= \{v'' \mid v'' \in \{v\} \mathbin{\dot\oplus} \{v'\} \wedge v \in \{\mathcal{E}(e_0,\sigma)\} \wedge v' \in \{\mathcal{E}(e_1,\sigma)\}\} \\
&\qquad && \text{(by IH, twice)} \\
&= \{\mathcal{E}(e_0,\sigma)\} \mathbin{\dot\oplus} \{\mathcal{E}(e_1,\sigma)\} && \text{(simplify)} \\
&= \{\mathcal{E}(e_0,\sigma) \oplus \mathcal{E}(e_1,\sigma)\} && (\text{by def. of } \mathbin{\dot\oplus}) \\
&= \{\mathcal{E}(e_0 \oplus e_1,\sigma)\} && \text{(by def. of } \mathcal{E})
\end{aligned}$$

□

C.2 Helper lemmas

Lemma 11 (First sequence helper lemma)**.**

$$\forall s_0, s_1, \sigma, \sigma'.\ \langle s_0,\sigma\rangle \to^* \sigma' \implies \langle s_0\ ;\ s_1,\sigma\rangle \to^* \langle s_1,\sigma'\rangle$$

Proof. Assume $\langle s_0,\sigma\rangle \to^n \sigma'$ for some $n \geqslant 0$. We prove $\langle s_0\ ;\ s_1,\sigma\rangle \to^n \langle s_1,\sigma'\rangle$ by induction in n.

- $n = 0$**:** The assumption $\langle s_0,\sigma\rangle \to^0 \sigma'$ is impossible hence the conclusion holds vacuously.
- $n = k+1$**:** Assuming $\langle s_0,\sigma\rangle \to^{k+1} \sigma'$ there are two cases for the shape of the configuration after the first step.
  If $\langle s_0,\sigma\rangle \to \sigma''$ then $k = 0$, $\sigma' = \sigma''$ and by rule SEQ2 $\langle s_0\ ;\ s_1,\sigma\rangle \to \langle s_1,\sigma'\rangle$.
  If on the other hand $\langle s_0,\sigma\rangle \to \langle s_0',\sigma''\rangle \to^k \sigma'$ by rule SEQ1 and the IH $\langle s_0\ ;\ s_1,\sigma\rangle \to \langle s_0'\ ;\ s_1,\sigma''\rangle \to^k \langle s_1,\sigma'\rangle$.

□

Lemma 12 (Second sequence helper lemma)**.**

$$\begin{aligned}
&\forall s_0, s_1, \sigma, \sigma', \sigma''.\ \langle s_0\ ;\ s_1,\sigma\rangle \to^* \sigma' \\
&\qquad \iff \langle s_0,\sigma\rangle \to^* \sigma'' \wedge \langle s_1,\sigma''\rangle \to^* \sigma'
\end{aligned}$$

Proof. $\Longleftarrow$**:** Assume $\exists n_0 \geqslant 0, n_1 \geqslant 0.\ \langle s_0,\sigma\rangle \to^{n_0} \sigma'' \wedge \langle s_1,\sigma''\rangle \to^{n_1} \sigma'$. By Lemma 11 and the second assumption we get $\langle s_0\ ;\ s_1,\sigma\rangle \to^{n_0} \langle s_1,\sigma''\rangle \to^{n_1} \sigma'$.

$\Longrightarrow$**:** Assume $\exists n \geqslant 0.\ \langle s_0\ ;\ s_1,\sigma\rangle \to^n \sigma'$ By induction in n we prove $\exists n_0 \geqslant 0, n_1 \geqslant 0, \sigma''.\langle s_0,\sigma\rangle \to^{n_0} \sigma'' \wedge \langle s_1,\sigma''\rangle \to^{n_1} \sigma'$.

- **Case** $n = 0$**:** The assumption $\langle s_0\ ;\ s_1,\sigma\rangle \to^0 \sigma'$ is impossible hence the conclusion holds vacuously.
- **Case** $n = k+1$**:** Assuming $\langle s_0\ ;\ s_1,\sigma\rangle \to^{k+1} \sigma'$ there are two possible cases for the shape of the configuration after the first step.
  If $\langle s_0\ ;\ s_1,\sigma\rangle \to \sigma''' \to^k \sigma'$ then $k = 0$ and $\sigma' = \sigma'''$. But this is impossible as neither SEQ1 nor SEQ2 can reduce a sequence to a store in one step hence the conclusion holds vacuously.
  If on the other hand $\langle s_0\ ;\ s_1,\sigma\rangle \to \langle s,\sigma'''\rangle \to^k \sigma'$ the first step could result from each of the two possible sequence rules. We now argue for both.
  - **Subcase SEQ1:** Now $s = s_0'\ ;\ s_1$ and $\langle s_0,\sigma\rangle \to \langle s_0',\sigma'''\rangle$. By the IH $\exists n_0 \geqslant 0, n_1 \geqslant 0, \sigma''.\langle s_0',\sigma'''\rangle \to^{n_0} \sigma'' \wedge \langle s_1,\sigma''\rangle \to^{n_1} \sigma'$ and hence $\langle s_0,\sigma\rangle \to^{n_0+1} \sigma''$.
  - **Subcase SEQ2:** Now $s = s_1$ and $\langle s_0,\sigma\rangle \to^1 \sigma'''$ and $\langle s_1,\sigma'''\rangle \to^k \sigma'$.

□

Lemma 13 (If helper lemma)**.**

$$\begin{aligned}
&\forall e, s_0, s_1, \sigma, \sigma'. \\
&\langle \mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1,\sigma\rangle \to^* \sigma' \\
&\quad \iff (\mathcal{E}(e,\sigma) \neq 0 \wedge \langle s_0,\sigma\rangle \to^* \sigma') \vee (\mathcal{E}(e,\sigma) = 0 \wedge \langle s_1,\sigma\rangle \to^* \sigma')
\end{aligned}$$

Proof. $\Longleftarrow$**:** Assume $(\mathcal{E}(e,\sigma) \neq 0 \wedge \langle s_0,\sigma\rangle \to^* \sigma') \vee (\mathcal{E}(e,\sigma) = 0 \wedge \langle s_1,\sigma\rangle \to^* \sigma')$. If $\mathcal{E}(e,\sigma) \neq 0 \wedge \langle s_0,\sigma\rangle \to^* \sigma'$ by rule IF1 $\langle \mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1,\sigma\rangle \to \langle s_0,\sigma\rangle$ and hence $\langle \mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1,\sigma\rangle \to^* \sigma'$.
If $\mathcal{E}(e,\sigma) = 0 \wedge \langle s_1,\sigma\rangle \to^* \sigma'$ by rule IF2
$\langle \mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1,\sigma\rangle \to \langle s_1,\sigma\rangle$ and hence
$\langle \mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1,\sigma\rangle \to^* \sigma'$.

$\Longrightarrow$**:** Assume $\exists n \geqslant 0.\ \langle \mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1,\sigma\rangle \to^n \sigma'$. We proceed by induction in n.

- **Case** $n = 0$**:** The assumption $\langle \mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1,\sigma\rangle \to^0 \sigma'$ is impossible hence the conclusion holds vacuously.
- **Case** $n = k+1$**:** Assuming $\langle \mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1,\sigma\rangle \to^{k+1} \sigma'$ there are two possible cases for the shape of the configuration after the first step.
  If $\langle \mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1,\sigma\rangle \to \sigma'' \to^k \sigma'$ then $k = 0$ and $\sigma' = \sigma''$. But this is impossible as neither IF1 nor IF2 can reduce a conditional to a store in one step hence the conclusion holds vacuously.
  If on the other hand $\langle \mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1,\sigma\rangle \to \langle s,\sigma''\rangle \to^k \sigma'$ there are two possible cases for the first transition.
  - **Subcase IF1:** Now $s = s_0$, $\sigma = \sigma''$, $\mathcal{E}(e,\sigma) = v \neq 0$, and $\langle s_0,\sigma\rangle \to^k \sigma'$.
  - **Subcase IF2:** Now $s = s_1$, $\sigma = \sigma''$, $\mathcal{E}(e,\sigma) = 0$, and $\langle s_1,\sigma\rangle \to^k \sigma'$.

□

Lemma 14 (While helper lemma)**.**

$$\forall e, s, \sigma_1, \sigma_n.$$
$$\langle \texttt{while}\ e\ \texttt{do}\ s, \sigma_1\rangle \to^* \sigma_n$$
$$\iff$$
$$\forall i \in \{1, \ldots, n-1\}.\ (\mathcal{E}(e, \sigma_i) \neq 0 \wedge \langle s, \sigma_i\rangle \to^* \sigma_{i+1}) \wedge \mathcal{E}(e, \sigma_n) = 0$$

Proof. $\Longrightarrow$**:** Assume $\exists m \geqslant 0.\ \langle \texttt{while}\ e\ \texttt{do}\ s, \sigma_1\rangle \to^m \sigma_n$. We now show the right hand side by induction in m.

Subcase $m = 0$**:** The assumption $\langle \texttt{while}\ e\ \texttt{do}\ s, \sigma_1\rangle \to^0 \sigma_n$ is impossible, hence the conclusion holds vacuously.

Subcase $m = k + 1$**:** We can now case analyze the first step of the assumption $\langle \texttt{while}\ e\ \texttt{do}\ s, \sigma_1\rangle \to^{k+1} \sigma_n$.
If the first step is by rule WHILE1 we know $\mathcal{E}(e, \sigma_1) = v \neq 0$ and $\langle \texttt{while}\ e\ \texttt{do}\ s, \sigma_1\rangle \to \langle s\ \texttt{;}\ \texttt{while}\ e\ \texttt{do}\ s, \sigma_1\rangle \to^k \sigma_n$. By Lemma 12 $\langle s, \sigma_1\rangle \to^{k_1} \sigma_2$ and $\langle \texttt{while}\ e\ \texttt{do}\ s, \sigma_2\rangle \to^{k_2} \sigma_n$ for $k_1, k_2 \geqslant 0$ such that $k = k_1 + k_2$. Hence by the IH we have $(\mathcal{E}(e, \sigma_1) \neq 0 \wedge \langle s, \sigma_1\rangle \to^* \sigma_2) \wedge (\mathcal{E}(e, \sigma_i) \neq 0 \wedge \langle s, \sigma_i\rangle \to^* \sigma_{i+1}) \wedge \mathcal{E}(e, \sigma_n) = 0$ for all $i \in \{2, \ldots, n-1\}$.
If the first step is by rule WHILE2 we know $\mathcal{E}(e, \sigma_1) = 0$ and $\langle \texttt{while}\ e\ \texttt{do}\ s, \sigma_1\rangle \to \sigma_1 \to^k \sigma_n$ and hence $k = 0$, $n = 1$ and $\sigma_1 = \sigma_n$ which satisfies the right hand side: $\forall i \in \{1, \ldots, n-1\} = \varnothing.\ (\mathcal{E}(e, \sigma_i) \neq 0 \wedge \langle s, \sigma_i\rangle \to^* \sigma_{i+1}) \wedge \mathcal{E}(e, \sigma_n) = 0$.

$\Longleftarrow$**:** Assume $(\mathcal{E}(e, \sigma_1) \neq 0 \wedge \langle s, \sigma_1\rangle \to^* \sigma_2) \wedge \ldots \wedge (\mathcal{E}(e, \sigma_{n-1}) \neq 0 \wedge \langle s, \sigma_{n-1}\rangle \to^* \sigma_n) \wedge \mathcal{E}(e, \sigma_n) = 0$ for a sequence of stores $\sigma_1, \ldots, \sigma_n$. We prove the left hand side by induction in n.

Subcase $n = 1$**:** Now $\sigma_1 = \sigma_n$ and $\mathcal{E}(e, \sigma_1) = 0$. Hence by one application of rule WHILE2 $\langle \texttt{while}\ e\ \texttt{do}\ s, \sigma_1\rangle \to^1 \sigma_1 = \sigma_n$.

Subcase $n = k + 1$**:** Assume $(\mathcal{E}(e, \sigma_1) \neq 0 \wedge \langle s, \sigma_1\rangle \to^* \sigma_2) \wedge \ldots \wedge (\mathcal{E}(e, \sigma_{n-1}) \neq 0 \wedge \langle s, \sigma_{n-1}\rangle \to^* \sigma_n) \wedge \mathcal{E}(e, \sigma_n) = 0$. Now by one application of rule WHILE1 $\langle \texttt{while}\ e\ \texttt{do}\ s, \sigma_1\rangle \to \langle s\ \texttt{;}\ \texttt{while}\ e\ \texttt{do}\ s, \sigma_1\rangle$.
Since $\langle s, \sigma_1\rangle \to^m \sigma_2$ by Lemma 11 we get $\langle s\ \texttt{;}\ \texttt{while}\ e\ \texttt{do}\ s, \sigma_1\rangle \to^m \langle \texttt{while}\ e\ \texttt{do}\ s, \sigma_2\rangle$. By the IH we have $\langle \texttt{while}\ e\ \texttt{do}\ s, \sigma_2\rangle \to^* \sigma_n$ and hence $\langle \texttt{while}\ e\ \texttt{do}\ s, \sigma_1\rangle \to^* \sigma_n$.

□

C.3 Proof of Theorem 1: Statement-level equivalence

Proof. We prove

$$\mathcal{C}[\![s]\!]c = \{\sigma' \mid \sigma \in c \wedge \langle s, \sigma\rangle \to^* \sigma'\}$$

We proceed to calculate a direct expression for $\mathcal{C}$ by structural induction on s

Case `skip`**:**

$$\begin{aligned}
&\mathcal{C}[\![\texttt{skip}]\!]c \\
&= \{\sigma' \mid \sigma \in c \wedge \langle \texttt{skip}, \sigma\rangle \to^* \sigma'\} && \text{(by above def.)} \\
&= \{\sigma \mid \sigma \in c \wedge \langle \texttt{skip}, \sigma\rangle \to \sigma\} && \text{(by inversion)} \\
&= c && \text{(simplify)}
\end{aligned}$$

Case `x := e`**:**

$$\begin{aligned}
&\mathcal{C}[\![\texttt{x := e}]\!]c \\
&= \{\sigma' \mid \sigma \in c \wedge \langle \texttt{x := e}, \sigma\rangle \to^* \sigma'\} && \text{(by above def.)} \\
&= \{\sigma' \mid \sigma \in c \wedge \langle \texttt{x := e}, \sigma\rangle \to \sigma'\} && \text{(by inversion)} \\
&= \{\sigma[\texttt{x} \mapsto v] \mid \sigma \in c \wedge \mathcal{E}(e, \sigma) = v\} && \text{(by rule ASSIGN)} \\
&= \{\sigma[\texttt{x} \mapsto v] \mid \sigma \in c \wedge v \in \{\mathcal{E}(e, \sigma)\}\} && \text{(by def. of } \in) \\
&= \{\sigma[\texttt{x} \mapsto v] \mid \sigma \in c \wedge v \in \mathcal{C}'[\![e]\!]\{\sigma\}\} && \text{(by Lemma 10)}
\end{aligned}$$

Case s_0 `;` s_1**:**

$$\begin{aligned}
&\mathcal{C}[\![s_0\ \texttt{;}\ s_1]\!]c \\
&= \{\sigma' \mid \sigma \in c \wedge \langle s_0\ \texttt{;}\ s_1, \sigma\rangle \to^* \sigma'\} && \text{(by above def.)} \\
&= \{\sigma' \mid \sigma \in c \wedge \langle s_0, \sigma\rangle \to^* \sigma'' \wedge \langle s_1, \sigma''\rangle \to^* \sigma'\} && \text{(by Lemma 12 above)} \\
&= \{\sigma' \mid \sigma \in c \wedge \sigma'' \in \mathcal{C}[\![s_0]\!]\{\sigma\} \wedge \sigma' \in \mathcal{C}[\![s_1]\!]\{\sigma''\}\} && \text{(by IH, twice)} \\
&= \mathcal{C}[\![s_1]\!]\{\sigma'' \mid \sigma \in c \wedge \sigma'' \in \mathcal{C}[\![s_0]\!]\{\sigma\}\} && \text{(simplify)} \\
&= \mathcal{C}[\![s_1]\!](\mathcal{C}[\![s_0]\!]\{\sigma \mid \sigma \in c\}) && \text{(simplify)} \\
&= \mathcal{C}[\![s_1]\!](\mathcal{C}[\![s_0]\!]c) && \text{(simplify)} \\
&= (\mathcal{C}[\![s_1]\!] \circ \mathcal{C}[\![s_0]\!])c && \text{(by def. of } \circ)
\end{aligned}$$

Case `if` e `then` s_0 `else` s_1**:**

$$\begin{aligned}
&\mathcal{C}[\![\texttt{if}\ e\ \texttt{then}\ s_0\ \texttt{else}\ s_1]\!]c \\
&= \{\sigma' \mid \sigma \in c \wedge \langle \texttt{if}\ e\ \texttt{then}\ s_0\ \texttt{else}\ s_1, \sigma\rangle \to^* \sigma'\} && \text{(by above def.)} \\
&= \{\sigma' \mid \sigma \in c \wedge (\mathcal{E}(e, \sigma) \neq 0 \wedge \langle s_0, \sigma\rangle \to^* \sigma') \\
&\qquad \vee (\mathcal{E}(e, \sigma) = 0 \wedge \langle s_1, \sigma\rangle \to^* \sigma')\} && \text{(by Lemma 13 above)} \\
&= \{\sigma' \mid \sigma \in c \wedge \mathcal{E}(e, \sigma) \neq 0 \wedge \langle s_0, \sigma\rangle \to^* \sigma'\} \\
&\quad \cup \{\sigma' \mid \sigma \in c \wedge \mathcal{E}(e, \sigma) = 0 \wedge \langle s_1, \sigma\rangle \to^* \sigma'\} && \text{(by def. of } \vee) \\
&= \mathcal{C}[\![s_0]\!]\{\sigma \in c \mid \mathcal{E}(e, \sigma) \neq 0\} \\
&\quad \cup \mathcal{C}[\![s_1]\!]\{\sigma \in c \mid \mathcal{E}(e, \sigma) = 0\} && \text{(by IH, twice)} \\
&= \mathcal{C}[\![s_0]\!]\{\sigma \in c \mid 0 \notin \{\mathcal{E}(e, \sigma)\}\} \\
&\quad \cup \mathcal{C}[\![s_1]\!]\{\sigma \in c \mid 0 \in \{\mathcal{E}(e, \sigma)\}\} && \text{(by singleton } \in/\notin) \\
&= \mathcal{C}[\![s_0]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\} \\
&\quad \cup \mathcal{C}[\![s_1]\!]\{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} && \text{(by Lemma 10)}
\end{aligned}$$

Case `while` e `do` s**:**

$$
\begin{aligned}
&\mathcal{C}[\![\texttt{while } e \texttt{ do } s]\!] \\
&= \lambda c.\, \{\sigma_n \mid \sigma_1 \in c \wedge \langle \texttt{while } e \texttt{ do } s, \sigma_1\rangle \to^* \sigma_n\} && \text{(by above def.)} \\
&= \lambda c.\, \{\sigma_n \mid \sigma_1 \in c \wedge \forall i \in \{1,\dots,n-1\}. \\
&\qquad (\mathcal{E}(e,\sigma_i) \neq 0 \wedge \langle s, \sigma_i\rangle \to^* \sigma_{i+1}) \wedge \mathcal{E}(e,\sigma_n) = 0\} && \text{(by Lemma 14 above)} \\
&= \lambda c.\, \{\sigma_n \mid \sigma_1 \in c \wedge \forall i \in \{1,\dots,n-1\}. \\
&\qquad (\mathcal{E}(e,\sigma_i) \neq 0 \wedge \sigma_{i+1} \in \mathcal{C}[\![s]\!]\{\sigma_i\}) \wedge \mathcal{E}(e,\sigma_n) = 0\} && \text{(by IH, } n-1 \text{ times)} \\
&= \lambda c.\, \{\sigma_n \mid \sigma_1 \in c \wedge \forall i \in \{1,\dots,n-1\}. \\
&\qquad (\sigma_{i+1} \in \mathcal{C}[\![s]\!]\{\sigma_i \mid \mathcal{E}(e,\sigma_i) \neq 0\}) \wedge \mathcal{E}(e,\sigma_n) = 0\} && \text{(by def. of } \mathcal{C}) \\
&= \lambda c.\, \{\sigma_n \mid \sigma_n \in (\lambda c_i.\, \mathcal{C}[\![s]\!]\{\sigma_i \in c_i \mid \mathcal{E}(e,\sigma_i) \neq 0\})^{n-1} c \\
&\qquad \wedge \mathcal{E}(e,\sigma_n) = 0\} && \text{(by def. of } (-)^{n-1}) \\
&= \lambda c.\, \{\sigma_n \mid \sigma_n \in (\lambda c_i.\, \mathcal{C}[\![s]\!]\{\sigma_i \in c_i \mid 0 \notin \{\mathcal{E}(e,\sigma_i)\}\})^{n-1} c \\
&\qquad \wedge 0 \in \{\mathcal{E}(e,\sigma_n)\}\} && \text{(by singleton } \in/\notin) \\
&= \lambda c.\, \{\sigma_n \mid \sigma_n \in (\lambda c_i.\, \mathcal{C}[\![s]\!]\{\sigma_i \in c_i \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma_i\}\})^{n-1} c \\
&\qquad \wedge 0 \in \mathcal{C}'[\![e]\!]\{\sigma_n\}\} && \text{(by Lemma 10)} \\
&= \operatorname{lfp} \lambda\Phi.\, \lambda c.\, \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \\
&\qquad \cup \Phi(\mathcal{C}[\![s]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}) && \text{(see below)}
\end{aligned}
$$

For the last step we prove inclusion in both directions. For short hand notation we let

$$
\begin{aligned}
F = \lambda\Phi.\, \lambda c.\, &\{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \\
&\cup \Phi(\mathcal{C}[\![s]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})
\end{aligned}
$$

$\dot\supseteq$**:** We prove that the left hand side is a fixed point of F, hence greater or equal to the least fixed point of F by Tarski's fixed point theorem.

$$
\begin{aligned}
&F(\lambda c.\, \{\sigma_n \mid \sigma_n \in (\lambda c_i.\, \mathcal{C}[\![s]\!]\{\sigma_i \in c_i \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma_i\}\})^{n-1} c \\
&\qquad \wedge 0 \in \mathcal{C}'[\![e]\!]\{\sigma_n\}\}) \\
&= \lambda c.\, \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \\
&\qquad \cup \{\sigma_n \mid \sigma_n \in (\lambda c_i.\, \mathcal{C}[\![s]\!]\{\sigma_i \in c_i \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma_i\}\})^{n-1} \\
&\qquad\qquad (\mathcal{C}[\![s]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}) \\
&\qquad\qquad \wedge 0 \in \mathcal{C}'[\![e]\!]\{\sigma_n\}\} && (\beta\text{-reduction}) \\
&= \lambda c.\, \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \\
&\qquad \cup \{\sigma_n \mid \sigma_n \in (\lambda c_i.\, \mathcal{C}[\![s]\!]\{\sigma_i \in c_i \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma_i\}\})^{n} c \\
&\qquad\qquad \wedge 0 \in \mathcal{C}'[\![e]\!]\{\sigma_n\}\} && \text{(simplify)} \\
&= \lambda c.\, \{\sigma_n \mid \sigma_n \in (\lambda c_i.\, \mathcal{C}[\![s]\!]\{\sigma_i \in c_i \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma_i\}\})^{n-1} c \\
&\qquad\qquad \wedge 0 \in \mathcal{C}'[\![e]\!]\{\sigma_n\}\} && \text{(simplify)}
\end{aligned}
$$

$\dot\subseteq$**:** Let Φ' be another fixed point of F ($F\Phi' = \Phi'$). This means that

$$
\begin{aligned}
\Phi' c &= F\Phi' c && \text{(by def. of fixed point)} \\
&= \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \Phi'(\mathcal{C}[\![s]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}) \\
&&& (\beta\text{-reduction, twice})
\end{aligned}
$$

We prove that $\lambda c.\, \{\dots\} \dot\subseteq \Phi'$, i.e., $(\lambda c.\, \{\dots\})c \subseteq \Phi' c$ for any c. Let c be given.
Specifically we now prove

$$
\begin{aligned}
\forall n \geqslant 1.\, \{\sigma_n \mid \sigma_n \in (\lambda c_i.\, \mathcal{C}[\![s]\!]\{\sigma_i \in c_i \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma_i\}\})^{n-1} c \\
\wedge 0 \in \mathcal{C}'[\![e]\!]\{\sigma_n\}\} \subseteq \Phi' c
\end{aligned}
$$

by induction in n.

$n = 1$**:**

$$
\begin{aligned}
&\{\sigma_n \mid \sigma_n \in (\lambda c_i.\, \mathcal{C}[\![s]\!]\{\sigma_i \in c_i \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma_i\}\})^{0} c \\
&\qquad \wedge 0 \in \mathcal{C}'[\![e]\!]\{\sigma_n\}\} \\
&= \{\sigma_n \mid \sigma_n \in c \wedge 0 \in \mathcal{C}'[\![e]\!]\{\sigma_n\}\} && \text{(simplify)} \\
&\subseteq \Phi' c && \text{(by the above)}
\end{aligned}
$$

$n = k + 1$**:**

$$
\begin{aligned}
&\{\sigma_n \mid \sigma_n \in (\lambda c_i.\, \mathcal{C}[\![s]\!]\{\sigma_i \in c_i \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma_i\}\})^{k+1-1} c \\
&\qquad \wedge 0 \in \mathcal{C}'[\![e]\!]\{\sigma_n\}\} \\
&= \{\sigma_n \mid \sigma_n \in (\lambda c_i.\, \mathcal{C}[\![s]\!]\{\sigma_i \in c_i \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma_i\}\})^{k-1} \\
&\qquad\qquad (\mathcal{C}[\![s]\!]\{\sigma_i \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma_i\}\}) \\
&\qquad\qquad \wedge 0 \in \mathcal{C}'[\![e]\!]\{\sigma_n\}\} && \text{(by def. of } f^m) \\
&\subseteq \Phi'(\mathcal{C}[\![s]\!]\{\sigma_i \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma_i\}\}) && \text{(by the IH)} \\
&\subseteq \Phi' c && \text{(by the above)}
\end{aligned}
$$

□

D. Soundness of approximate semantics

D.1 Expression-level soundness

Lemma 15 (Soundness of approximate expression semantics)**.**

$$\forall e \in \mathit{Exp}, b \in \mathit{Var} \to 2^{\mathit{Val}}.\ (\mathcal{C}'[\![e]\!] \circ \gamma_{\text{BC}})(b) \subseteq \mathcal{B}'[\![e]\!]b$$

Proof. Let $e \in \mathit{Exp}$ and $b \in \mathit{Var} \to 2^{\mathit{Val}}$ be given. Proceed by structural induction on e.

Case n:

$$\begin{aligned}
&(\mathcal{C}'[\![n]\!] \circ \gamma_{\text{BC}})(b) \\
&= ((\lambda c.\ \{n\}) \circ \gamma_{\text{BC}})(b) && \text{(by def. of } \mathcal{C}') \\
&= \{n\} && (\beta\text{-reduction}) \\
&= \mathcal{B}'[\![n]\!]b && \text{(by def. of } \mathcal{B}')
\end{aligned}$$

Case $\mathtt{x}$:

$$\begin{aligned}
&(\mathcal{C}'[\![\mathtt{x}]\!] \circ \gamma_{\text{BC}})(b) \\
&= (\lambda b.\ \{\sigma(\mathtt{x}) \mid \sigma \in b\} \circ \gamma_{\text{BC}})(b) && \text{(by def. of } \mathcal{C}') \\
&= \{\sigma(\mathtt{x}) \mid \sigma \in \gamma_{\text{BC}}(b)\} && (\beta\text{-reduction}) \\
&= \{\sigma(\mathtt{x}) \mid \sigma \in \{\sigma \mid \forall \mathtt{x} \in \mathit{Var}.\ \sigma(\mathtt{x}) \in b(\mathtt{x})\}\} && \text{(by def. of } \gamma_{\text{BC}}) \\
&= \{\sigma(\mathtt{x}) \mid \forall \sigma, \mathtt{x} \in \mathit{Var}.\ \sigma(\mathtt{x}) \in b(\mathtt{x})\} && \text{(simplify)} \\
&\subseteq b(\mathtt{x}) && \text{(simplify)} \\
&= \mathcal{B}'[\![\mathtt{x}]\!]b && \text{(by def. of } \mathcal{B}')
\end{aligned}$$

Case $e \oplus e'$:

$$\begin{aligned}
&(\mathcal{C}'[\![e \oplus e']\!] \circ \gamma_{\text{BC}})(b) \\
&= (\lambda b.\ \{v'' \mid v'' \in \{v\} \mathbin{\dot\oplus} \{v'\} \\
&\qquad \wedge\ \sigma \in b \wedge v \in \mathcal{C}'[\![e]\!]\{\sigma\} \\
&\qquad \wedge\ v' \in \mathcal{C}'[\![e']\!]\{\sigma\}\} \circ \gamma_{\text{BC}})(b) && \text{(by def. of } \mathcal{C}') \\
&= \{v'' \mid v'' \in \{v\} \mathbin{\dot\oplus} \{v'\} \\
&\qquad \wedge\ \sigma \in \gamma_{\text{BC}}(b) \wedge v \in \mathcal{C}'[\![e]\!]\{\sigma\} \\
&\qquad \wedge\ v' \in \mathcal{C}'[\![e']\!]\{\sigma\}\} && (\beta\text{-reduction}) \\
&\subseteq \{v'' \mid v'' \in \{v\} \mathbin{\dot\oplus} \{v'\} \\
&\qquad \wedge\ v \in \mathcal{C}'[\![e]\!](\gamma_{\text{BC}}(b)) \wedge v' \in \mathcal{C}'[\![e']\!](\gamma_{\text{BC}}(b))\} && \text{(simplify)} \\
&= \mathcal{C}'[\![e]\!](\gamma_{\text{BC}}(b)) \mathbin{\dot\oplus} \mathcal{C}'[\![e']\!](\gamma_{\text{BC}}(b)) && \text{(simplify)} \\
&\subseteq \mathcal{B}'[\![e]\!]b \mathbin{\dot\oplus} \mathcal{B}'[\![e']\!]b && \text{(IH, twice, } \dot\oplus \text{ monotone)}
\end{aligned}$$

□

D.2 Proof of Theorem 2: Statement-level soundness

Proof. Let $s \in \mathit{Stm}$ and $b \in \mathit{Var} \to 2^{\mathit{Val}}$ be given. Proceed by structural induction on s.

Case $\mathtt{skip}$:

$$\begin{aligned}
&(\alpha_{\text{CB}} \circ \mathcal{C}[\![\mathtt{skip}]\!] \circ \gamma_{\text{BC}})(b) \\
&= (\alpha_{\text{CB}} \circ (\lambda b.\ b) \circ \gamma_{\text{BC}})(b) && \text{(by def. of } \mathcal{C}) \\
&= (\alpha_{\text{CB}} \circ \gamma_{\text{BC}})(b) && \text{(identity fun.)} \\
&\dot\subseteq b && (\alpha_{\text{CB}} \circ \gamma_{\text{BC}} \text{ reductive}) \\
&= \mathcal{B}[\![\mathtt{skip}]\!]b && \text{(def. of } \mathcal{B})
\end{aligned}$$

Case $\mathtt{x} := e$:

$$\begin{aligned}
&(\alpha_{\text{CB}} \circ \mathcal{C}[\![\mathtt{x} := e]\!] \circ \gamma_{\text{BC}})(b) \\
&= (\alpha_{\text{CB}} \circ (\lambda b.\ \{\sigma[\mathtt{x} \mapsto v] \mid \sigma \in b \wedge v \in \mathcal{C}'[\![e]\!]\{\sigma\}\}) \circ \gamma_{\text{BC}})(b) && \text{(by def. of } \mathcal{C}) \\
&= \alpha_{\text{CB}}(\{\sigma[\mathtt{x} \mapsto v] \mid \sigma \in \gamma_{\text{BC}}(b) \wedge v \in \mathcal{C}'[\![e]\!]\{\sigma\}\}) && (\beta\text{-reduction}) \\
&= \lambda \mathtt{y}.\ \{\sigma[\mathtt{x} \mapsto v](\mathtt{y}) \mid \sigma \in \gamma_{\text{BC}}(b) \wedge v \in \mathcal{C}'[\![e]\!]\{\sigma\}\} && \text{(by def. of } \alpha_{\text{CB}}) \\
&\dot\subseteq \lambda \mathtt{y}.\ \{\sigma[\mathtt{x} \mapsto v](\mathtt{y}) \mid \sigma \in \gamma_{\text{BC}}(b) \wedge v \in \mathcal{C}'[\![e]\!](\gamma_{\text{BC}}(b))\} && \text{(simplify)} \\
&= \lambda \mathtt{y}.\ b[\mathtt{x} \mapsto \mathcal{C}'[\![e]\!](\gamma_{\text{BC}}(b))](\mathtt{y}) && \text{(by def. of } \gamma_{\text{BC}}) \\
&= b[\mathtt{x} \mapsto \mathcal{C}'[\![e]\!](\gamma_{\text{BC}}(b))] && (\eta\text{-reduction}) \\
&\dot\subseteq b[\mathtt{x} \mapsto \mathcal{B}'[\![e]\!]b] && \text{(by Lemma 15)} \\
&= \mathcal{B}[\![\mathtt{x} := e]\!]b && \text{(by def. of } \mathcal{B})
\end{aligned}$$

Case $s_0\ ;\ s_1$:

$$\begin{aligned}
&(\alpha_{\text{CB}} \circ \mathcal{C}[\![s_0\ ;\ s_1]\!] \circ \gamma_{\text{BC}})(b) \\
&= (\alpha_{\text{CB}} \circ (\mathcal{C}[\![s_1]\!] \circ \mathcal{C}[\![s_0]\!]) \circ \gamma_{\text{BC}})(b) && \text{(by def. of } \mathcal{C}) \\
&\dot\subseteq (\alpha_{\text{CB}} \circ (\mathcal{C}[\![s_1]\!] \circ \gamma_{\text{BC}} \circ \alpha_{\text{CB}} \circ \mathcal{C}[\![s_0]\!]) \circ \gamma_{\text{BC}})(b) && (\gamma_{\text{BC}} \circ \alpha_{\text{CB}} \text{ extensive}) \\
&\dot\subseteq (\mathcal{B}[\![s_1]\!] \circ \mathcal{B}[\![s_0]\!])(b) && \text{(by IH, twice)} \\
&= \mathcal{B}[\![s_0\ ;\ s_1]\!]b && \text{(by def. of } \mathcal{B})
\end{aligned}$$

Case $\mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1$:

$$\begin{aligned}
&(\alpha_{\text{CB}} \circ \mathcal{C}[\![\mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1]\!] \circ \gamma_{\text{BC}})(b) \\
&= (\alpha_{\text{CB}} \circ (\lambda b.\ \mathcal{C}[\![s_0]\!]\{\sigma \in b \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\} \\
&\qquad \cup\ \mathcal{C}[\![s_1]\!]\{\sigma \in b \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\}) \circ \gamma_{\text{BC}})(b) && \text{(by def. of } \mathcal{C}) \\
&= \alpha_{\text{CB}}(\mathcal{C}[\![s_0]\!]\{\sigma \in \gamma_{\text{BC}}(b) \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\} \\
&\qquad \cup\ \mathcal{C}[\![s_1]\!]\{\sigma \in \gamma_{\text{BC}}(b) \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\}) && (\beta\text{-reduction}) \\
&= \alpha_{\text{CB}}(\mathcal{C}[\![s_0]\!]\{\sigma \in \gamma_{\text{BC}}(b) \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}) \\
&\qquad \dot\cup\ \alpha_{\text{CB}}(\mathcal{C}[\![s_1]\!]\{\sigma \in \gamma_{\text{BC}}(b) \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\}) && (\alpha_{\text{CB}} \text{ a CJM}) \\
&\dot\subseteq \alpha_{\text{CB}}(\mathcal{C}[\![s_0]\!](\gamma_{\text{BC}}(b))) \mathbin{\dot\cup} \alpha_{\text{CB}}(\mathcal{C}[\![s_1]\!](\gamma_{\text{BC}}(b))) && \text{(upward judge)} \\
&\dot\subseteq \mathcal{B}[\![s_0]\!]b \mathbin{\dot\cup} \mathcal{B}[\![s_1]\!]b && \text{(by IH, twice)} \\
&= \mathcal{B}[\![\mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1]\!]b && \text{(by def. of } \mathcal{B})
\end{aligned}$$

Case $\mathtt{while}\ e\ \mathtt{do}\ s$: In this case our higher-order Galois connection reads:

$$\begin{aligned}
\alpha_{\to}(\Phi) &= \lambda b.\ \alpha_{\text{CB}}(\Phi(\gamma_{\text{BC}}(b))) \\
\gamma_{\to}(\Phi) &= \lambda c.\ \gamma_{\text{BC}}(\Phi(\alpha_{\text{CB}}(c)))
\end{aligned}$$

For short-hand notation, we let

$$\begin{aligned}
F = &\lambda \Phi.\ \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \\
&\cup\ \Phi(\mathcal{C}[\![s]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})
\end{aligned}$$

First observe that for any given monotone Φ:

$$
\begin{aligned}
&(\alpha_{\to} \circ F \circ \gamma_{\to})\Phi \\
&= \alpha_{\to}(F(\gamma_{\to}(\Phi))) && \text{(by def. of } \circ) \\
&= \alpha_{\to}(F(\lambda c'.\, \gamma_{\text{BC}}(\Phi(\alpha_{\text{CB}}(c'))))) && \text{(by def. of } \gamma_{\to}) \\
&= \alpha_{\to}(\lambda c.\, \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \\
&\qquad \cup (\lambda c'.\, \gamma_{\text{BC}}(\Phi(\alpha_{\text{CB}}(c'))))\mathcal{C}[\![s]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}) \\
&&& \text{(by def. of } F) \\
&= \alpha_{\to}(\lambda c.\, \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \\
&\qquad \cup \gamma_{\text{BC}}(\Phi(\alpha_{\text{CB}}(\mathcal{C}[\![s]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})))) \\
&&& (\beta\text{-reduction}) \\
&\dot{\subseteq} \alpha_{\to}(\lambda c.\, c \cup \gamma_{\text{BC}}(\Phi(\alpha_{\text{CB}}(\mathcal{C}[\![s]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})))) \\
&&& (\alpha_{\to} \text{ monotone}) \\
&\dot{\subseteq} \alpha_{\to}(\lambda c.\, c \cup \gamma_{\text{BC}}(\Phi(\alpha_{\text{CB}}(\mathcal{C}[\![s]\!]c)))) \\
&&& (\mathcal{C}, \alpha_{\text{CB}}, \Phi', \gamma_{\text{BC}}, \alpha_{\to} \text{ monotone}) \\
&= \lambda b.\, \alpha_{\text{CB}}(\gamma_{\text{BC}}(b) \cup \gamma_{\text{BC}}(\Phi(\alpha_{\text{CB}}(\mathcal{C}[\![s]\!]\gamma_{\text{BC}}(b))))) \\
&&& \text{(by def. of } \alpha_{\to}) \\
&= \lambda b.\, \alpha_{\text{CB}}(\gamma_{\text{BC}}(b)) \dot{\cup} \alpha_{\text{CB}}(\gamma_{\text{BC}}(\Phi(\alpha_{\text{CB}}(\mathcal{C}[\![s]\!]\gamma_{\text{BC}}(b))))) \\
&&& (\alpha_{\text{CB}} \text{ a CJM}) \\
&\dot{\subseteq} \lambda b.\, b \dot{\cup} \Phi(\alpha_{\text{CB}}(\mathcal{C}[\![s]\!]\gamma_{\text{BC}}(b))) && (\alpha_{\text{CB}} \circ \gamma_{\text{BC}} \text{ reductive, twice}) \\
&\dot{\subseteq} \lambda b.\, b \dot{\cup} \Phi(\mathcal{B}[\![s]\!]b) && \text{(by the IH, } \Phi' \text{ monotone)}
\end{aligned}
$$

Since Φ and $\mathcal{B}$ are monotone, this functional is itself monotone. Now we can utilize these observations:

$$
\begin{aligned}
&(\alpha_{\text{CB}} \circ \mathcal{C}[\![\texttt{while}\ e\ \texttt{do}\ s]\!] \circ \gamma_{\text{BC}})(b) \\
&= \alpha_{\to}(\mathcal{C}[\![\texttt{while}\ e\ \texttt{do}\ s]\!])(b) && \text{(by def. of } \alpha_{\to}) \\
&= \alpha_{\to}(\text{lfp}\, F)(b) && \text{(by def. of } \mathcal{C}) \\
&\dot{\subseteq} (\text{lfp}\, \alpha_{\to} \circ F \circ \gamma_{\to})(b) && \text{(fixed point transfer theorem)} \\
&\dot{\subseteq} (\text{lfp}\, \lambda\Phi.\, \lambda b.\, b \dot{\cup} \Phi(\mathcal{B}[\![s]\!]b))(b) \\
&&& \text{(fixed point transfer theorem, above obs.)} \\
&= \mathcal{B}[\![\texttt{while}\ e\ \texttt{do}\ s]\!]b && \text{(by def. of } \mathcal{B})
\end{aligned}
$$

□

E. Soundness of constant propagation analysis

E.1 Expression level soundness

Lemma 16 (Soundness of expression analysis)**.**

$$\forall e \in \mathit{Exp}, a \in \mathbb{A} : \ (\hat{\alpha} \circ \mathcal{B}'[\![e]\!] \circ \gamma_{\text{AB}})(a) \sqsubseteq \mathcal{A}'[\![e]\!]\, a$$

Proof. Let $e \in \mathit{Exp}$ and $a \in \mathbb{A}$ be given. Proceed by structural induction on e.

Case n**:**

$$\begin{aligned}
&(\hat{\alpha} \circ \mathcal{B}'[\![n]\!] \circ \gamma_{\text{AB}})(a) \\
&= (\hat{\alpha} \circ (\lambda b.\, \{n\}) \circ \gamma_{\text{AB}})(a) && \text{(by def. of } \mathcal{B}') \\
&= \hat{\alpha}(\{n\}) && (\beta\text{-reduction}) \\
&= \mathtt{n} && \text{(by def. of } \hat{\alpha}) \\
&= \mathcal{A}'[\![n]\!]a && \text{(by def. of } \mathcal{A}')
\end{aligned}$$

Case `x`**:**

$$\begin{aligned}
&(\hat{\alpha} \circ \mathcal{B}'[\![\mathtt{x}]\!] \circ \gamma_{\text{AB}})(a) \\
&= (\hat{\alpha} \circ (\lambda b.\, b(\mathtt{x})) \circ \gamma_{\text{AB}})(a) && \text{(by def. of } \mathcal{B}') \\
&= \hat{\alpha}(\gamma_{\text{AB}}(a)(\mathtt{x})) && (\beta\text{-reduction}) \\
&= \hat{\alpha}(\hat{\gamma}(a(\mathtt{x}))) && \text{(by def. of } \gamma_{\text{AB}}) \\
&\sqsubseteq a(\mathtt{x}) && (\hat{\alpha} \circ \hat{\gamma} \text{ reductive}) \\
&= \mathcal{A}'[\![\mathtt{x}]\!]a && \text{(by def. of } \mathcal{A}')
\end{aligned}$$

Case $e_0 \oplus e_1$**:**

$$\begin{aligned}
&(\hat{\alpha} \circ \mathcal{B}'[\![e_0 \oplus e_1]\!] \circ \gamma_{\text{AB}})(a) \\
&= (\hat{\alpha} \circ (\lambda b.\, \mathcal{B}'[\![e_0]\!]b \mathbin{\dot{\oplus}} \mathcal{B}'[\![e_1]\!]b) \circ \gamma_{\text{AB}})(a) && \text{(by def. of } \mathcal{B}') \\
&= \hat{\alpha}(\mathcal{B}'[\![e_0]\!]\gamma_{\text{AB}}(a) \mathbin{\dot{\oplus}} \mathcal{B}'[\![e_1]\!]\gamma_{\text{AB}}(a)) && (\beta\text{-reduction}) \\
&\sqsubseteq \hat{\alpha}((\hat{\gamma} \circ \hat{\alpha})(\mathcal{B}'[\![e_0]\!]\gamma_{\text{AB}}(a)) \mathbin{\dot{\oplus}} (\hat{\gamma} \circ \hat{\alpha})(\mathcal{B}'[\![e_1]\!]\gamma_{\text{AB}}(a))) \\
&\qquad (\hat{\gamma} \circ \hat{\alpha} \text{ extensive, twice, } \dot{\oplus}, \hat{\alpha} \text{ monotone}) \\
&\sqsubseteq \hat{\alpha}((\hat{\gamma}(\mathcal{A}'[\![e_0]\!]a)) \mathbin{\dot{\oplus}} (\hat{\gamma}(\mathcal{A}'[\![e_1]\!]a))) && \text{(by IH, twice)} \\
&\sqsubseteq (\mathcal{A}'[\![e_0]\!]a) \mathbin{\hat{\oplus}} (\mathcal{A}'[\![e_1]\!]a) && \text{(by def. of } \hat{\oplus}) \\
&= \mathcal{A}'[\![e_0 \oplus e_1]\!]a && \text{(by def. of } \mathcal{A}')
\end{aligned}$$

□

E.2 Proof of Theorem 3: Statement-level soundness

Proof. Let $s \in \mathit{Stm}$ and $a \in \mathbb{A}$ be given. Proceed by structural induction on s.

Case `skip`**:**

$$\begin{aligned}
&(\alpha_{\text{BA}} \circ \mathcal{B}[\![\mathtt{skip}]\!] \circ \gamma_{\text{AB}})(a) \\
&= (\alpha_{\text{BA}} \circ (\lambda b.\, b) \circ \gamma_{\text{AB}})(a) && \text{(by def. of } \mathcal{B}) \\
&= (\alpha_{\text{BA}} \circ \gamma_{\text{AB}})(a) && \text{(identity fun.)} \\
&\dot{\sqsubseteq}\ a && (\alpha_{\text{BA}} \circ \gamma_{\text{AB}} \text{ reductive}) \\
&= \mathcal{A}[\![\mathtt{skip}]\!]a && \text{(by def. of } \mathcal{A})
\end{aligned}$$

Case `x := e`**:**

$$\begin{aligned}
&(\alpha_{\text{BA}} \circ \mathcal{B}[\![\mathtt{x := e}]\!] \circ \gamma_{\text{AB}})(a) \\
&= (\alpha_{\text{BA}} \circ (\lambda b.\, b[\mathtt{x} \mapsto \mathcal{B}'[\![e]\!](b)]) \circ \gamma_{\text{AB}})(a) && \text{(by def. of } \mathcal{B}) \\
&= \alpha_{\text{BA}}(\gamma_{\text{AB}}(a)[\mathtt{x} \mapsto \mathcal{B}'[\![e]\!](\gamma_{\text{AB}}(a))]) && (\beta\text{-reduction}) \\
&= (\alpha_{\text{BA}}(\gamma_{\text{AB}}(a)))[\mathtt{x} \mapsto \hat{\alpha}(\mathcal{B}'[\![e]\!](\gamma_{\text{AB}}(a)))] && \text{(by def. of } \alpha_{\text{BA}}) \\
&\dot{\sqsubseteq}\ a[\mathtt{x} \mapsto \hat{\alpha}(\mathcal{B}'[\![e]\!](\gamma_{\text{AB}}(a)))] && (\alpha_{\text{BA}} \circ \gamma_{\text{AB}} \text{ reductive}) \\
&\dot{\sqsubseteq}\ a[\mathtt{x} \mapsto \mathcal{A}'[\![e]\!]a] && \text{(by Lemma 16)} \\
&= \mathcal{A}[\![\mathtt{x := e}]\!]a && \text{(by def. of } \mathcal{A})
\end{aligned}$$

Case s_0 `;` s_1**:**

$$\begin{aligned}
&(\alpha_{\text{BA}} \circ \mathcal{B}[\![s_0 \ \mathtt{;}\ s_1]\!] \circ \gamma_{\text{AB}})(a) \\
&= (\alpha_{\text{BA}} \circ \mathcal{B}[\![s_1]\!] \circ \mathcal{B}[\![s_0]\!] \circ \gamma_{\text{AB}})(a) && \text{(by def. of } \mathcal{B}) \\
&\dot{\sqsubseteq}\ (\alpha_{\text{BA}} \circ \mathcal{B}[\![s_1]\!] \circ \gamma_{\text{AB}} \circ \alpha_{\text{BA}} \circ \mathcal{B}[\![s_0]\!] \circ \gamma_{\text{AB}})(a) \\
&\qquad (\gamma_{\text{AB}} \circ \alpha_{\text{BA}} \text{ is extensive}) \\
&\dot{\sqsubseteq}\ (\mathcal{A}[\![s_1]\!] \circ \mathcal{A}[\![s_0]\!])(a) && \text{(by IH, twice)} \\
&= \mathcal{A}[\![s_0 \ \mathtt{;}\ s_1]\!]a && \text{(by def. of } \mathcal{A})
\end{aligned}$$

Case `if` e `then` s_0 `else` s_1**:**

$$\begin{aligned}
&(\alpha_{\text{BA}} \circ \mathcal{B}[\![\mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1]\!] \circ \gamma_{\text{AB}})(a) \\
&= (\alpha_{\text{BA}} \circ (\lambda b.\, \mathcal{B}[\![s_0]\!]b \mathbin{\dot{\cup}} \mathcal{B}[\![s_1]\!]b) \circ \gamma_{\text{AB}})(a) && \text{(by def. of } \mathcal{B}) \\
&= \alpha_{\text{BA}}(\mathcal{B}[\![s_0]\!](\gamma_{\text{AB}}(a)) \mathbin{\dot{\cup}} \mathcal{B}[\![s_1]\!](\gamma_{\text{AB}}(a))) && (\beta\text{-reduction}) \\
&= \alpha_{\text{BA}}(\mathcal{B}[\![s_0]\!](\gamma_{\text{AB}}(a))) \mathbin{\dot{\sqcup}} \alpha_{\text{BA}}(\mathcal{B}[\![s_1]\!](\gamma_{\text{AB}}(a))) \\
&\qquad (\alpha_{\text{BA}} \text{ a CJM}) \\
&\dot{\sqsubseteq}\ \mathcal{A}[\![s_0]\!]a \mathbin{\dot{\sqcup}} \mathcal{A}[\![s_1]\!]a && \text{(by IH, twice)} \\
&= \mathcal{A}[\![\mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1]\!]a && \text{(by def. of } \mathcal{A})
\end{aligned}$$

Case `while` e `do` s**:** In this case our higher-order Galois connection reads:

$$\begin{aligned}
\alpha_{\to}(\Phi) &= \lambda a.\, \alpha_{\text{BA}}(\Phi(\gamma_{\text{AB}}(a))) \\
\gamma_{\to}(\Phi) &= \lambda b.\, \gamma_{\text{AB}}(\Phi(\alpha_{\text{BA}}(b)))
\end{aligned}$$

Let a monotone Φ' be given. Now observe that:

$$\begin{aligned}
&(\alpha_{\to} \circ (\lambda\Phi.\, \lambda b.\, b \mathbin{\dot{\cup}} \Phi(\mathcal{B}[\![s]\!]b)) \circ \gamma_{\to})\Phi' \\
&= (\alpha_{\to} \circ (\lambda\Phi.\, \lambda b.\, b \mathbin{\dot{\cup}} \Phi(\mathcal{B}[\![s]\!]b)))(\lambda b'.\, \gamma_{\text{AB}}(\Phi'(\alpha_{\text{BA}}(b')))) \\
&\qquad (\text{by def. of } \gamma_{\to}) \\
&= \alpha_{\to}(\lambda b.\, b \mathbin{\dot{\cup}} (\lambda b'.\, \gamma_{\text{AB}}(\Phi'(\alpha_{\text{BA}}(b'))))(\mathcal{B}[\![s]\!]b)) \\
&\qquad (\beta\text{-reduction}) \\
&= \alpha_{\to}(\lambda b.\, b \mathbin{\dot{\cup}} \gamma_{\text{AB}}(\Phi'(\alpha_{\text{BA}}(\mathcal{B}[\![s]\!]b)))) && (\beta\text{-reduction}) \\
&= \lambda a.\, \alpha_{\text{BA}}(\gamma_{\text{AB}}(a) \mathbin{\dot{\cup}} \gamma_{\text{AB}}(\Phi'(\alpha_{\text{BA}}(\mathcal{B}[\![s]\!](\gamma_{\text{AB}}(a)))))) \\
&\qquad (\text{by def. of } \alpha_{\to}) \\
&= \lambda a.\, \alpha_{\text{BA}}(\gamma_{\text{AB}}(a)) \mathbin{\dot{\sqcup}} \alpha_{\text{BA}}(\gamma_{\text{AB}}(\Phi'(\alpha_{\text{BA}}(\mathcal{B}[\![s]\!](\gamma_{\text{AB}}(a))))))) \\
&\qquad (\alpha_{\text{BA}} \text{ a CJM}) \\
&\dot{\sqsubseteq}\ \lambda a.\, a \mathbin{\dot{\sqcup}} \alpha_{\text{BA}}(\gamma_{\text{AB}}(\Phi'(\alpha_{\text{BA}}(\mathcal{B}[\![s]\!](\gamma_{\text{AB}}(a))))))) \\
&\qquad (\alpha_{\text{BA}} \circ \gamma_{\text{AB}} \text{ reductive}) \\
&\dot{\sqsubseteq}\ \lambda a.\, a \mathbin{\dot{\sqcup}} \Phi'(\alpha_{\text{BA}}(\mathcal{B}[\![s]\!](\gamma_{\text{AB}}(a))))) && (\alpha_{\text{BA}} \circ \gamma_{\text{AB}} \text{ reductive}) \\
&\dot{\sqsubseteq}\ \lambda a.\, a \mathbin{\dot{\sqcup}} \Phi'(\mathcal{A}[\![s]\!]a) && \text{(by IH, } \Phi' \text{ monotone)}
\end{aligned}$$

Since Φ' and $\mathcal{A}$ are monotone, this functional is itself monotone. Now, we utilize these observations in the following rewriting:

$$\begin{aligned}
&(\alpha_{\text{BA}} \circ \mathcal{B}[\![\mathtt{while}\ e\ \mathtt{do}\ s]\!] \circ \gamma_{\text{AB}})(a) \\
&= \alpha_{\to}(\mathcal{B}[\![\mathtt{while}\ e\ \mathtt{do}\ s]\!])(a) && \text{(by def. of } \alpha_{\to}) \\
&= \alpha_{\to}(\mathrm{lfp}\, \lambda\Phi.\, \lambda b.\, b \mathbin{\dot{\cup}} \Phi(\mathcal{B}[\![s]\!]b))(a) && \text{(by def. of } \mathcal{B}) \\
&\dot{\sqsubseteq}\ (\mathrm{lfp}\, \alpha_{\to} \circ (\lambda\Phi.\, \lambda b.\, b \mathbin{\dot{\cup}} \Phi(\mathcal{B}[\![s]\!]b)) \circ \gamma_{\to})(a) \\
&\qquad (\text{fixed point transfer theorem}) \\
&\dot{\sqsubseteq}\ (\mathrm{lfp}(\lambda\Phi.\, \lambda a.\, a \mathbin{\dot{\sqcup}} \Phi(\mathcal{A}[\![s]\!]a)))(a) \\
&\qquad (\text{fixed point transfer theorem, above obs.}) \\
&= \mathcal{A}[\![\mathtt{while}\ e\ \mathtt{do}\ s]\!]a && \text{(by def. of } \mathcal{A})
\end{aligned}$$

□

F. Monotonicity proofs

Please recall that the domain of monotone (endo-)functions over a complete lattice themselves constitute a complete lattice. We use this fact in the proofs of Theorems 18, 20, and 22.

Lemma 17 ($\mathcal{C}'$ monotone)**.**

$$\forall e, c, c'.\ c \subseteq c' \implies \mathcal{C}'[\![e]\!]c \subseteq \mathcal{C}'[\![e]\!]c'$$

Proof. Let e, $c \subseteq c'$ be given. We proceed by structural induction on e.

Case n**:**

$$\mathcal{C}'[\![n]\!]c = \{n\} = \mathcal{C}'[\![n]\!]c' \qquad \text{(by def. of } \mathcal{C}')$$

Case `x`**:**

$$\mathcal{C}'[\![\mathtt{x}]\!]c = \{\sigma(\mathtt{x}) \mid \sigma \in c\} \subseteq \{\sigma(\mathtt{x}) \mid \sigma \in c'\} = \mathcal{C}'[\![\mathtt{x}]\!]c' \qquad \text{(by def. of } \mathcal{C}')$$

Case $e_0 \oplus e_1$**:**

$$\begin{aligned}
&\mathcal{C}'[\![e_0 \oplus e_1]\!]c \\
&= \{v'' \mid v'' \in \{v\} \mathbin{\dot\oplus} \{v'\} \wedge \sigma \in c \\
&\qquad \wedge v \in \mathcal{C}'[\![e]\!]\{\sigma\} \wedge v' \in \mathcal{C}'[\![e']\!]\{\sigma\}\} && \text{(by def. of } \mathcal{C}') \\
&\subseteq \{v'' \mid v'' \in \{v\} \mathbin{\dot\oplus} \{v'\} \wedge \sigma \in c' \\
&\qquad \wedge v \in \mathcal{C}'[\![e]\!]\{\sigma\} \wedge v' \in \mathcal{C}'[\![e']\!]\{\sigma\}\} && \text{(by assumption)} \\
&= \mathcal{C}'[\![e_0 \oplus e_1]\!]c' && \text{(by def. of } \mathcal{C}')
\end{aligned}$$

□

Theorem 18 ($\mathcal{C}$ monotone)**.**

$$\forall s, c, c'.\ c \subseteq c' \implies \mathcal{C}[\![s]\!]c \subseteq \mathcal{C}[\![s]\!]c'$$

Proof. Let s and $c \subseteq c'$ be given. We proceed by structural induction on s.

Case `skip`**:**

$$\mathcal{C}[\![\mathtt{skip}]\!]c = c \subseteq c' = \mathcal{C}[\![\mathtt{skip}]\!]c' \qquad \text{(by def. of } \mathcal{C})$$

Case `x := e`**:**

$$\begin{aligned}
&\mathcal{C}[\![\mathtt{x := } e]\!]c \\
&= \{\sigma[\mathtt{x} \mapsto v] \mid \sigma \in c \wedge v \in \mathcal{C}'[\![e]\!]\{\sigma\}\} && \text{(by def. of } \mathcal{C}) \\
&\subseteq \{\sigma[\mathtt{x} \mapsto v] \mid \sigma \in c' \wedge v \in \mathcal{C}'[\![e]\!]\{\sigma\}\} && \text{(by assumption)} \\
&= \mathcal{C}[\![\mathtt{x := } e]\!]c' && \text{(by def. of } \mathcal{C})
\end{aligned}$$

Case s_0 `;` s_1**:**

$$\begin{aligned}
&\mathcal{C}[\![s_0 \mathtt{\ ;\ } s_1]\!]c \\
&= (\mathcal{C}[\![s_1]\!] \circ \mathcal{C}[\![s_0]\!])c && \text{(by def. of } \mathcal{C}) \\
&\subseteq (\mathcal{C}[\![s_1]\!] \circ \mathcal{C}[\![s_0]\!])c' && \text{(by IH, twice)} \\
&= \mathcal{C}[\![s_0 \mathtt{\ ;\ } s_1]\!]c' && \text{(by def. of } \mathcal{C})
\end{aligned}$$

Case `if` e `then` s_0 `else` s_1**:**

$$\begin{aligned}
&\mathcal{C}[\![\mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1]\!]c \\
&= \mathcal{C}[\![s_0]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \mathcal{C}[\![s_1]\!]\{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} && \text{(by def. of } \mathcal{C}) \\
&\subseteq \mathcal{C}[\![s_0]\!]\{\sigma \in c' \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \mathcal{C}[\![s_1]\!]\{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} && \text{(by IH)} \\
&\subseteq \mathcal{C}[\![s_0]\!]\{\sigma \in c' \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \mathcal{C}[\![s_1]\!]\{\sigma \in c' \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} && \text{(by IH)} \\
&= \mathcal{C}[\![\mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1]\!]c' && \text{(by def. of } \mathcal{C})
\end{aligned}$$

Case `while` e `do` s**:** Recall the while rule:

$$\mathcal{C}[\![\mathtt{while}\ e\ \mathtt{do}\ s]\!] = \operatorname{lfp} \lambda\Phi.\ \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \Phi(\mathcal{C}[\![s]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})$$

For convenience we name the functional F:

$$F = \lambda\Phi.\ \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \Phi(\mathcal{C}[\![s]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})$$

First we prove that applying the functional F to a monotone function Φ, yields a monotone function as a result. As a consequence the functional F constitutes an operator over the complete lattice of monotone functions.
Let $c \subseteq c'$ be given.

$$\begin{aligned}
&(F\Phi)c \\
&= \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \Phi(\mathcal{C}[\![s]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}) && (\beta\text{-reduction, twice}) \\
&\subseteq \{\sigma \in c' \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \Phi(\mathcal{C}[\![s]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}) && \text{(by assumption)} \\
&\subseteq \{\sigma \in c' \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \Phi(\mathcal{C}[\![s]\!]\{\sigma \in c' \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}) && \text{(by assumption, IH, monotonicity of } \Phi) \\
&= (\lambda\Phi.\ \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \Phi(\mathcal{C}[\![s]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}))\Phi c' && (\beta\text{-expansion, twice}) \\
&= (F\Phi)c' && \text{(be def. of } F)
\end{aligned}$$

Second we prove that the functional F itself is monotone which guarantees that the while rule is well defined by Tarski's fixed point theorem.
Let monotone functions $\Phi \mathrel{\dot\subseteq} \Phi'$ be given.

$$\begin{aligned}
&F\Phi \\
&= \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \Phi(\mathcal{C}[\![s]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}) && (\beta\text{-reduction}) \\
&\mathrel{\dot\subseteq} \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \Phi'(\mathcal{C}[\![s]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}) && \text{(by assumption)} \\
&= (\lambda\Phi.\ \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \Phi(\mathcal{C}[\![s]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}))\Phi' && (\beta\text{-expansion}) \\
&= F\Phi' && \text{(by def. of } F)
\end{aligned}$$

Since the least fixed point of F is itself an element of the lattice of monotone functions, it is monotone:

$$\mathcal{C}[\![\mathtt{while}\ e\ \mathtt{do}\ s]\!]c = (\operatorname{lfp} F)c \subseteq (\operatorname{lfp} F)c' = \mathcal{C}[\![\mathtt{while}\ e\ \mathtt{do}\ s]\!]c'$$

which concludes this case.

□

Lemma 19 ($\mathcal{B}'$ monotone)**.**

$$\forall e, b, b'.\ b \mathrel{\dot\subseteq} b' \implies \mathcal{B}'[\![e]\!]b \subseteq \mathcal{B}'[\![e]\!]b'$$

Proof. Let e, $b \mathrel{\dot\subseteq} b'$ be given. We proceed by structural induction on e.

Case n**:**

$$\mathcal{B}'[\![n]\!]b = \{n\} = \mathcal{B}'[\![n]\!]b' \qquad \text{(by def. of } \mathcal{B}')$$

Case `x`**:**

$$\mathcal{B}'[\![\mathtt{x}]\!]b = b(\mathtt{x}) \subseteq b'(\mathtt{x}) = \mathcal{B}'[\![\mathtt{x}]\!]b' \qquad \text{(by def. of } \mathcal{B}', \text{ assumption)}$$

Case $e_0 \oplus e_1$**:**

$$\begin{aligned}
&\mathcal{B}'[\![e_0 \oplus e_1]\!]b \\
&= \mathcal{B}'[\![e]\!]b \mathbin{\dot\oplus} \mathcal{B}'[\![e]\!]b && \text{(by def. of } \mathcal{B}') \\
&\subseteq \mathcal{B}'[\![e]\!]b' \mathbin{\dot\oplus} \mathcal{B}'[\![e]\!]b && \text{(by IH, monotonicity of } \dot\oplus) \\
&\subseteq \mathcal{B}'[\![e]\!]b' \mathbin{\dot\oplus} \mathcal{B}'[\![e]\!]b' && \text{(by IH, monotonicity of } \dot\oplus) \\
&= \mathcal{B}'[\![e_0 \oplus e_1]\!]b' && \text{(by def. of } \mathcal{B}')
\end{aligned}$$

□

Theorem 20 ($\mathcal{B}$ monotone)**.**

$$\forall s, b, b'.\ b \mathbin{\dot\subseteq} b' \implies \mathcal{B}[\![s]\!]b \mathbin{\dot\subseteq} \mathcal{B}[\![s]\!]b'$$

Proof. Let s, $b \mathbin{\dot\subseteq} b'$ be given. We proceed by structural induction on s.

Case `skip`**:**

$$\mathcal{B}[\![\texttt{skip}]\!]b = b \mathbin{\dot\subseteq} b' = \mathcal{B}[\![\texttt{skip}]\!]b' \qquad \text{(by def. of } \mathcal{B}\text{, assumption)}$$

Case `x := `e**:**

$$\begin{aligned}
&\mathcal{B}[\![\texttt{x := } e]\!]b \\
&= b[\texttt{x} \mapsto \mathcal{B}'[\![e]\!]b] && \text{(by def. of } \mathcal{B}) \\
&\mathbin{\dot\subseteq} b'[\texttt{x} \mapsto \mathcal{B}'[\![e]\!]b] && \text{(by assumption)} \\
&\mathbin{\dot\subseteq} b'[\texttt{x} \mapsto \mathcal{B}'[\![e]\!]b'] && \text{(by Lemma 19, def. of } \dot\subseteq) \\
&= \mathcal{B}[\![\texttt{x := } e]\!]b' && \text{(by def. of } \mathcal{B})
\end{aligned}$$

Case s_0 `;` s_1**:**

$$\begin{aligned}
&\mathcal{B}[\![s_0 \texttt{ ; } s_1]\!]b \\
&= (\mathcal{B}[\![s_1]\!] \circ \mathcal{B}[\![s_0]\!])b && \text{(by def. of } \mathcal{B}) \\
&\mathbin{\dot\subseteq} (\mathcal{B}[\![s_1]\!] \circ \mathcal{B}[\![s_0]\!])b' && \text{(by IH, twice)} \\
&= \mathcal{B}[\![s_0 \texttt{ ; } s_1]\!]b' && \text{(by def. of } \mathcal{B})
\end{aligned}$$

Case `if` e `then` s_0 `else` s_1**:**

$$\begin{aligned}
&\mathcal{B}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!]b \\
&= \mathcal{B}[\![s_0]\!]b \mathbin{\dot\cup} \mathcal{B}[\![s_1]\!]b && \text{(by def. of } \mathcal{B}) \\
&\mathbin{\dot\subseteq} \mathcal{B}[\![s_0]\!]b' \mathbin{\dot\cup} \mathcal{B}[\![s_1]\!]b && \text{(by IH)} \\
&\mathbin{\dot\subseteq} \mathcal{B}[\![s_0]\!]b' \mathbin{\dot\cup} \mathcal{B}[\![s_1]\!]b' && \text{(by IH)} \\
&= \mathcal{B}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!]b' && \text{(by def. of } \mathcal{B})
\end{aligned}$$

Case `while` e `do` s**:** Recall the while rule:

$$\mathcal{B}[\![\texttt{while } e \texttt{ do } s]\!] = \operatorname{lfp} \lambda\Phi.\ \lambda b.\ b \mathbin{\dot\cup} \Phi(\mathcal{B}[\![s]\!]b)$$

For convenience we again name the functional F: $F = \lambda\Phi.\ \lambda b.\ b \mathbin{\dot\cup} \Phi(\mathcal{B}[\![s]\!]b)$.
We first prove that applying the functional F to a monotone function Φ yields a monotone function as a result. As a consequence the functional F operates over the complete lattice of monotone functions.
Let $b \mathbin{\dot\subseteq} b'$ be given.

$$\begin{aligned}
&(F\Phi)b \\
&= b \mathbin{\dot\cup} \Phi(\mathcal{B}[\![s]\!]b) && (\beta\text{-reduction, twice}) \\
&\mathbin{\dot\subseteq} b' \mathbin{\dot\cup} \Phi(\mathcal{B}[\![s]\!]b) && \text{(by assumption)} \\
&\mathbin{\dot\subseteq} b' \mathbin{\dot\cup} \Phi(\mathcal{B}[\![s]\!]b') && \text{(by IH, } \Phi \text{ monotone)} \\
&= (F\Phi)b' && \text{(by def. of } F)
\end{aligned}$$

Second we prove that the functional itself is monotone. This guarantees that the while rule is well defined by Tarski's fixed point theorem.

Let $\Phi \mathbin{\ddot\subseteq} \Phi'$ be given.

$$\begin{aligned}
&F\Phi \\
&= \lambda b.\ b \mathbin{\dot\cup} \Phi(\mathcal{B}[\![s]\!]b) && (\beta\text{-reduction}) \\
&\mathbin{\ddot\subseteq} \lambda b.\ b \mathbin{\dot\cup} \Phi'(\mathcal{B}[\![s]\!]b) && \text{(by def. of } \ddot\subseteq\text{, assumption)} \\
&= F\Phi' && \text{(by def. of } F)
\end{aligned}$$

Since the least fixed point is itself an element of the complete lattice of monotone functions the while rule is monotone:

$$\mathcal{B}[\![\texttt{while } e \texttt{ do } s]\!]b = (\operatorname{lfp} F)b \mathbin{\dot\subseteq} (\operatorname{lfp} F)b' = \mathcal{B}[\![\texttt{while } e \texttt{ do } s]\!]b'$$

which concludes this case.

□

Lemma 21 ($\mathcal{A}'$ monotone)**.**

$$\forall e, a, a'.\ a \mathbin{\dot\subseteq} a' \implies \mathcal{A}'[\![e]\!]a \sqsubseteq \mathcal{A}'[\![e]\!]a'$$

Proof. Let e and $a \mathbin{\dot\subseteq} a'$ be given. We proceed by structural induction on e.

Case n**:**

$$\mathcal{A}'[\![e]\!]a = \alpha(\{n\}) = \mathcal{A}'[\![e]\!]a' \qquad \text{(by def. of } \mathcal{A}')$$

Case `x`**:**

$$\mathcal{A}'[\![\texttt{x}]\!]a = a(\texttt{x}) \sqsubseteq a'(\texttt{x}) = \mathcal{A}'[\![\texttt{x}]\!]a' \qquad \text{(by def. of } \mathcal{A}', \dot\subseteq)$$

Case $e_0 \oplus e_1$**:**

$$\begin{aligned}
&\mathcal{A}'[\![e_0 \oplus e_1]\!]a \\
&= \mathcal{A}'[\![e_0]\!]a \mathbin{\hat\oplus} \mathcal{A}'[\![e_1]\!]a && \text{(by def. of } \mathcal{A}') \\
&\sqsubseteq \mathcal{A}'[\![e_0]\!]a' \mathbin{\hat\oplus} \mathcal{A}'[\![e_1]\!]a && (\text{IH}, \hat\oplus \text{ monotone}) \\
&\sqsubseteq \mathcal{A}'[\![e_0]\!]a' \mathbin{\hat\oplus} \mathcal{A}'[\![e_1]\!]a' && (\text{IH}, \hat\oplus \text{ monotone}) \\
&= \mathcal{A}'[\![e_0 \oplus e_1]\!]a' && \text{(by def. of } \mathcal{A}')
\end{aligned}$$

□

Theorem 22 ($\mathcal{A}$ monotone)**.**

$$\forall s, a, a'.\ a \mathbin{\dot\subseteq} a' \implies \mathcal{A}[\![s]\!]a \mathbin{\dot\subseteq} \mathcal{A}[\![s]\!]a'$$

Proof. Let s and $a \mathbin{\dot\subseteq} a'$ be given. We proceed by structural induction on s.

Case `skip`**:**

$$\mathcal{A}[\![\texttt{skip}]\!]a = a \mathbin{\dot\subseteq} a' = \mathcal{A}[\![s]\!]a' \qquad \text{(by def. of } \mathcal{A})$$

Case `x := `e**:**

$$\begin{aligned}
&\mathcal{A}[\![\texttt{x := } e]\!]a \\
&= a[\texttt{x} \mapsto \mathcal{A}'[\![e]\!]a] && \text{(by def. of } \mathcal{A}) \\
&\mathbin{\dot\subseteq} a'[\texttt{x} \mapsto \mathcal{A}'[\![e]\!]a] && \text{(by def. of } \dot\subseteq) \\
&\mathbin{\dot\subseteq} a'[\texttt{x} \mapsto \mathcal{A}'[\![e]\!]a'] && \text{(by def. of } \dot\subseteq\text{, Lemma 21)} \\
&= \mathcal{A}[\![\texttt{x := } e]\!]a' && \text{(by def. of } \mathcal{A})
\end{aligned}$$

Case s_0 `;` s_1**:**

$$\begin{aligned}
&\mathcal{A}[\![s_0 \texttt{ ; } s_1]\!]a \\
&= (\mathcal{A}[\![s_1]\!] \circ \mathcal{A}[\![s_0]\!])a && \text{(by def. of } \mathcal{A}) \\
&\mathbin{\dot\subseteq} (\mathcal{A}[\![s_1]\!] \circ \mathcal{A}[\![s_0]\!])a' && \text{(IH, twice)} \\
&= \mathcal{A}[\![s_0 \texttt{ ; } s_1]\!]a' && \text{(by def. of } \mathcal{A})
\end{aligned}$$

Case `if` e `then` s_0 `else` s_1**:**

$$\begin{aligned}
&\mathcal{A}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!]a \\
&= \mathcal{A}[\![s]\!]a \mathbin{\dot\sqcup} \mathcal{A}[\![s']\!]a && \text{(by def. of } \mathcal{A}) \\
&\dot\sqsubseteq \mathcal{A}[\![s]\!]a' \mathbin{\dot\sqcup} \mathcal{A}[\![s']\!]a' && \text{(IH, twice)} \\
&= \mathcal{A}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!]a' && \text{(by def. of } \mathcal{A})
\end{aligned}$$

Case `while` e `do` s**:** Recall the while rule:

$$\mathcal{A}[\![\texttt{while } e \texttt{ do } s]\!] = \operatorname{lfp} \lambda\Phi.\, \lambda a.\, a \mathbin{\dot\sqcup} \Phi(\mathcal{A}[\![s]\!]a)$$

For convenience we again name the functional F:

$$F = \lambda\Phi.\, \lambda a.\, a \mathbin{\dot\sqcup} \Phi(\mathcal{A}[\![s]\!]a)$$

First we prove that applying the functional F to a monotone function Φ yields a monotone result, hence the functional F constitutes a function over the complete lattice of monotone functions.
Let $a \mathbin{\dot\sqsubseteq} a'$ and a monotone Φ be given.

$$\begin{aligned}
&F\Phi a \\
&= a \mathbin{\dot\sqcup} \Phi(\mathcal{A}[\![s]\!]a) && \text{(by def. of } F) \\
&\dot\sqsubseteq a' \mathbin{\dot\sqcup} \Phi(\mathcal{A}[\![s]\!]a) && \text{(by def. of } \dot\sqsubseteq) \\
&\dot\sqsubseteq a' \mathbin{\dot\sqcup} \Phi(\mathcal{A}[\![s]\!]a') && \text{(by IH, monotonicity of } \Phi\text{, def. of } \dot\sqsubseteq) \\
&= F\Phi a' && \text{(by def. of } F)
\end{aligned}$$

Second we prove that the functional F itself is monotone over the complete lattice of monotone functions), which guarantees that the while rule is well defined by Tarski's fixed point theorem.
Let monotone functions Φ and Φ' be given and assume $\Phi \mathbin{\ddot\sqsubseteq} \Phi'$

$$\begin{aligned}
&F\Phi \\
&= \lambda a.\, a \mathbin{\dot\sqcup} \Phi(\mathcal{A}[\![s]\!]a) && \text{(by def. of } F) \\
&\ddot\sqsubseteq \lambda a.\, a \mathbin{\dot\sqcup} \Phi'(\mathcal{A}[\![s]\!]a) && \text{(by assumption, def. of } \ddot\sqsubseteq) \\
&= F\Phi' && \text{(by def. of } F)
\end{aligned}$$

Since the resulting fixed point is an element in the complete lattice of monotone functions it is itself monotone:

$\mathcal{A}[\![\texttt{while } e \texttt{ do } s]\!]a = (\operatorname{lfp} F)a \mathbin{\dot\sqsubseteq} (\operatorname{lfp} F)a' = \mathcal{A}[\![\texttt{while } e \texttt{ do } s]\!]a'$

which concludes this case.

□

G. Proof of Theorem 4: Data-flow equation soundness

Note: the paper version of the data-flow equations are purely statement based, whereas the below development also formulates data-flow equations for (labelled) expressions. Because of the below equality, analyzing expressions with $\mathcal{A}'$ or with data-flow equations is equivalent.

We prove that a solution $[\![-]\!]_{\mathsf{in}}$, $[\![-]\!]_{\mathsf{out}}$ to the data-flow constraints is sound wrt. to the derived analysis:

$$\mathcal{A}'[\![e^\ell]\!]([\![e^\ell]\!]_{\mathsf{in}}) = [\![e^\ell]\!]_{\mathsf{out}}$$
$$\mathcal{A}[\![s^\ell]\!]([\![s^\ell]\!]_{\mathsf{in}}) \dot{\sqsubseteq} [\![s^\ell]\!]_{\mathsf{out}}$$

for the following expression related equations:

$$[\![n^\ell]\!]_{\mathsf{out}} = \mathtt{n}$$
$$[\![\mathtt{x}^\ell]\!]_{\mathsf{out}} = [\![\mathtt{x}^\ell]\!]_{\mathsf{in}}(\mathtt{x})$$
$$[\![e_0^{\ell_0}]\!]_{\mathsf{in}} = [\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]_{\mathsf{in}}$$
$$[\![e_1^{\ell_1}]\!]_{\mathsf{in}} = [\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]_{\mathsf{in}}$$
$$[\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]_{\mathsf{out}} = [\![e_0^{\ell_0}]\!]_{\mathsf{out}} \mathbin{\hat{\oplus}} [\![e_1^{\ell_1}]\!]_{\mathsf{out}}$$
$$[\![e_0^{\ell_0}]\!]_{\mathsf{in}} = [\![\mathtt{x} :=^\ell e_0^{\ell_0}]\!]_{\mathsf{in}}$$
$$[\![\mathtt{x} :=^\ell e_0^{\ell_0}]\!]_{\mathsf{out}} = [\![\mathtt{x} :=^\ell e_0^{\ell_0}]\!]_{\mathsf{in}}[\mathtt{x} \mapsto [\![e_0^{\ell_0}]\!]_{\mathsf{out}}]$$

Proof. **Expression soundness:** Let e^ℓ be given. We proceed by structural induction on e^ℓ.

Case n^ℓ:

$$\mathcal{A}'[\![n^\ell]\!]([\![n^\ell]\!]_{\mathsf{in}}) = \mathtt{n} = [\![n^\ell]\!]_{\mathsf{out}} \quad \text{(by def. of } \mathcal{A}', [\![n^\ell]\!]_{\mathsf{out}})$$

Case $\mathtt{x}^\ell$:

$$\mathcal{A}'[\![\mathtt{x}^\ell]\!]([\![\mathtt{x}^\ell]\!]_{\mathsf{in}}) = [\![\mathtt{x}^\ell]\!]_{\mathsf{in}}(\mathtt{x}^\ell) = [\![\mathtt{x}^\ell]\!]_{\mathsf{out}} \quad \text{(by def. of } \mathcal{A}', [\![\mathtt{x}^\ell]\!]_{\mathsf{out}})$$

Case $e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}$:

$$\begin{aligned}
&\mathcal{A}'[\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]([\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]_{\mathsf{in}}) \\
&= \mathcal{A}'[\![e_0^{\ell_0}]\!]([\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]_{\mathsf{in}}) \mathbin{\hat{\oplus}} \mathcal{A}'[\![e_1^{\ell_1}]\!]([\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]_{\mathsf{in}}) && \text{(by def. of } \mathcal{A}') \\
&= \mathcal{A}'[\![e_0^{\ell_0}]\!]([\![e_0^{\ell_0}]\!]_{\mathsf{in}}) \mathbin{\hat{\oplus}} \mathcal{A}'[\![e_1^{\ell_1}]\!]([\![e_1^{\ell_1}]\!]_{\mathsf{in}}) && \text{(by def. of } [\![e_0^{\ell_0}]\!]_{\mathsf{in}}, [\![e_1^{\ell_1}]\!]_{\mathsf{in}}) \\
&= [\![e_0^{\ell_0}]\!]_{\mathsf{out}} \mathbin{\hat{\oplus}} [\![e_1^{\ell_1}]\!]_{\mathsf{out}} && \text{(by IH, twice)} \\
&= [\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]_{\mathsf{out}} && \text{(by def. of } [\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]_{\mathsf{out}})
\end{aligned}$$

Statement soundness: Let s^ℓ be given. We proceed by structural induction on s^ℓ.

Case $\mathtt{skip}^\ell$:

$$\mathcal{A}[\![\mathtt{skip}^\ell]\!]([\![\mathtt{skip}^\ell]\!]_{\mathsf{in}}) = [\![\mathtt{skip}^\ell]\!]_{\mathsf{in}} = [\![\mathtt{skip}^\ell]\!]_{\mathsf{out}} \quad \text{(by def. of } \mathcal{A}, [\![\mathtt{skip}^\ell]\!]_{\mathsf{in}})$$

Case $\mathtt{x} :=^\ell e^{\ell_0}$:

$$\begin{aligned}
&\mathcal{A}[\![\mathtt{x} :=^\ell e^{\ell_0}]\!]([\![\mathtt{x} :=^\ell e^{\ell_0}]\!]_{\mathsf{in}}) \\
&= ([\![\mathtt{x} :=^\ell e^{\ell_0}]\!]_{\mathsf{in}})[\mathtt{x} \mapsto \mathcal{A}'[\![e^{\ell_0}]\!]([\![\mathtt{x} :=^\ell e^{\ell_0}]\!]_{\mathsf{in}})] && \text{(by def. of } \mathcal{A}) \\
&= ([\![\mathtt{x} :=^\ell e^{\ell_0}]\!]_{\mathsf{in}})[\mathtt{x} \mapsto \mathcal{A}'[\![e^{\ell_0}]\!]([\![e^{\ell_0}]\!]_{\mathsf{in}})] && \text{(by def. of } [\![e^{\ell_0}]\!]_{\mathsf{in}}) \\
&= ([\![\mathtt{x} :=^\ell e^{\ell_0}]\!]_{\mathsf{in}})[\mathtt{x} \mapsto [\![e^{\ell_0}]\!]_{\mathsf{out}}] && \text{(by first half of theorem)} \\
&= [\![\mathtt{x} :=^\ell e^{\ell_0}]\!]_{\mathsf{out}} && \text{(by def. of } [\![\mathtt{x} :=^\ell e^{\ell_0}]\!]_{\mathsf{out}})
\end{aligned}$$

Case $s_0^{\ell_0} ;^\ell s_1^{\ell_1}$:

$$\begin{aligned}
&\mathcal{A}[\![s_0^{\ell_0} ;^\ell s_1^{\ell_1}]\!]([\![s_0^{\ell_0} ;^\ell s_1^{\ell_1}]\!]_{\mathsf{in}}) \\
&= (\mathcal{A}[\![s_1^{\ell_1}]\!] \circ \mathcal{A}[\![s_0^{\ell_0}]\!])([\![s_0^{\ell_0} ;^\ell s_1^{\ell_1}]\!]_{\mathsf{in}}) && \text{(by def. of } \mathcal{A}) \\
&= (\mathcal{A}[\![s_1^{\ell_1}]\!] \circ \mathcal{A}[\![s_0^{\ell_0}]\!])([\![s_0^{\ell_0}]\!]_{\mathsf{in}}) && \text{(by def. of } [\![s_0^{\ell_0}]\!]_{\mathsf{in}}) \\
&\dot{\sqsubseteq} \mathcal{A}[\![s_1^{\ell_1}]\!]([\![s_0^{\ell_0}]\!]_{\mathsf{out}}) && \text{(by IH, } \mathcal{A} \text{ monotone)} \\
&= \mathcal{A}[\![s_1^{\ell_1}]\!]([\![s_1^{\ell_1}]\!]_{\mathsf{in}}) && \text{(by def. of } [\![s_1^{\ell_1}]\!]_{\mathsf{in}}) \\
&\dot{\sqsubseteq} [\![s_1^{\ell_1}]\!]_{\mathsf{out}} && \text{(by IH)} \\
&= [\![s_0^{\ell_0} ;^\ell s_1^{\ell_1}]\!]_{\mathsf{out}} && \text{(by def. of } [\![s_0^{\ell_0} ;^\ell s_1^{\ell_1}]\!]_{\mathsf{out}})
\end{aligned}$$

Case $\mathtt{if}^\ell\ e\ \mathtt{then}\ s_0^{\ell_0}\ \mathtt{else}\ s_1^{\ell_1}$:

$$\begin{aligned}
&\mathcal{A}[\![\mathtt{if}^\ell\ e\ \mathtt{then}\ s_0^{\ell_0}\ \mathtt{else}\ s_1^{\ell_1}]\!]([\![\mathtt{if}^\ell\ e\ \mathtt{then}\ s_0^{\ell_0}\ \mathtt{else}\ s_1^{\ell_1}]\!]_{\mathsf{in}}) \\
&= \mathcal{A}[\![s_0^{\ell_0}]\!]([\![\mathtt{if}^\ell\ e\ \mathtt{then}\ s_0^{\ell_0}\ \mathtt{else}\ s_1^{\ell_1}]\!]_{\mathsf{in}}) \\
&\quad \dot{\sqcup}\ \mathcal{A}[\![s_1^{\ell_1}]\!]([\![\mathtt{if}^\ell\ e\ \mathtt{then}\ s_0^{\ell_0}\ \mathtt{else}\ s_1^{\ell_1}]\!]_{\mathsf{in}}) && \text{(by def. of } \mathcal{A}) \\
&= \mathcal{A}[\![s_0^{\ell_0}]\!]([\![s_0^{\ell_0}]\!]_{\mathsf{in}}) \mathbin{\dot{\sqcup}} \mathcal{A}[\![s_1^{\ell_1}]\!]([\![s_1^{\ell_1}]\!]_{\mathsf{in}}) && \text{(by def. of } [\![s_0^{\ell_0}]\!]_{\mathsf{in}}, [\![s_1^{\ell_1}]\!]_{\mathsf{in}}) \\
&\dot{\sqsubseteq} [\![s_0^{\ell_0}]\!]_{\mathsf{out}} \mathbin{\dot{\sqcup}} [\![s_1^{\ell_1}]\!]_{\mathsf{out}} && \text{(by IH, twice)} \\
&= [\![\mathtt{if}^\ell\ e\ \mathtt{then}\ s_0^{\ell_0}\ \mathtt{else}\ s_1^{\ell_1}]\!]_{\mathsf{out}} && \text{(by def. of } [\![\mathtt{if}^\ell\ e\ \mathtt{then}\ s_0^{\ell_0}\ \mathtt{else}\ s_1^{\ell_1}]\!]_{\mathsf{out}})
\end{aligned}$$

Case $\mathtt{while}^\ell\ e\ \mathtt{do}\ s^{\ell_0}$: Recall the while equations:

$$[\![\mathtt{while}^\ell\ e\ \mathtt{do}\ s^{\ell_0}]\!]_{\mathsf{out}} = [\![s^{\ell_0}]\!]_{\mathsf{in}} \quad \text{(eq.1)}$$
$$[\![s^{\ell_0}]\!]_{\mathsf{in}} = [\![\mathtt{while}^\ell\ e\ \mathtt{do}\ s^{\ell_0}]\!]_{\mathsf{in}} \mathbin{\dot{\sqcup}} [\![s^{\ell_0}]\!]_{\mathsf{out}} \quad \text{(eq.2)}$$

We now prove by (inner) induction that for all $n \geqslant 0$

$$\ddot{\mathfrak{F}}^n(\ddot{\bot})([\![\mathtt{while}^\ell\ e\ \mathtt{do}\ s^{\ell_0}]\!]_{\mathsf{in}} \mathbin{\dot{\sqcup}} [\![s^{\ell_0}]\!]_{\mathsf{out}}) \dot{\sqsubseteq} [\![\mathtt{while}^\ell\ e\ \mathtt{do}\ s^{\ell_0}]\!]_{\mathsf{out}}$$

where $\mathfrak{F} = \lambda\Phi.\, \lambda a.\, a \mathbin{\dot{\sqcup}} \Phi(\mathcal{A}[\![s^{\ell_0}]\!]a)$. From here it follows that

$$\begin{aligned}
&\mathcal{A}[\![\mathtt{while}^\ell\ e\ \mathtt{do}\ s^{\ell_0}]\!]([\![\mathtt{while}^\ell\ e\ \mathtt{do}\ s^{\ell_0}]\!]_{\mathsf{in}}) \\
&= (\mathrm{lfp}\,\mathfrak{F})([\![\mathtt{while}^\ell\ e\ \mathtt{do}\ s^{\ell_0}]\!]_{\mathsf{in}}) && \text{(by def. of } \mathcal{A}) \\
&= (\ddot{\sqcup}_i \mathfrak{F}^i(\ddot{\bot}))([\![\mathtt{while}^\ell\ e\ \mathtt{do}\ s^{\ell_0}]\!]_{\mathsf{in}}) && \text{(by Kleene's fixed point theorem)} \\
&= (\lambda a.\, \dot{\sqcup}_i \mathfrak{F}^i(\ddot{\bot})a)([\![\mathtt{while}^\ell\ e\ \mathtt{do}\ s^{\ell_0}]\!]_{\mathsf{in}}) && \text{(by def. of } \ddot{\sqcup}) \\
&= \dot{\sqcup}_i \mathfrak{F}^i(\ddot{\bot})[\![\mathtt{while}^\ell\ e\ \mathtt{do}\ s^{\ell_0}]\!]_{\mathsf{in}} && (\beta\text{-reduction}) \\
&\dot{\sqsubseteq} \dot{\sqcup}_i \mathfrak{F}^i(\ddot{\bot})([\![\mathtt{while}^\ell\ e\ \mathtt{do}\ s^{\ell_0}]\!]_{\mathsf{in}} \mathbin{\dot{\sqcup}} [\![s^{\ell_0}]\!]_{\mathsf{out}}) && \text{(by monotonicity of } \mathfrak{F}^i(\ddot{\bot})) \\
&\dot{\sqsubseteq} [\![\mathtt{while}^\ell\ e\ \mathtt{do}\ s^{\ell_0}]\!]_{\mathsf{out}} && \text{(by above)}
\end{aligned}$$

Case $n = 0$**:**

$$\begin{aligned}
&\mathfrak{F}^0(\ddot{\bot})(\llbracket \texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0} \rrbracket_{\mathsf{in}} \dot{\sqcup} \llbracket s^{\ell_0} \rrbracket_{\mathsf{out}}) \\
&= \ddot{\bot}(\llbracket \texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0} \rrbracket_{\mathsf{in}} \dot{\sqcup} \llbracket s^{\ell_0} \rrbracket_{\mathsf{out}}) && \text{(by def. of } \mathfrak{F}^0\text{)} \\
&= \dot{\bot} && (\beta\text{-reduction}) \\
&\dot{\sqsubseteq} \llbracket \texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0} \rrbracket_{\mathsf{out}} && \text{(by def. of } \dot{\bot}\text{)}
\end{aligned}$$

Case $n = k + 1$**:**

Assume

$$\mathfrak{F}^k(\ddot{\bot})(\llbracket \texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0} \rrbracket_{\mathsf{in}} \dot{\sqcup} \llbracket s^{\ell_0} \rrbracket_{\mathsf{out}}) \dot{\sqsubseteq} \llbracket \texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0} \rrbracket_{\mathsf{out}}$$

We now reason as follows:

$$\begin{aligned}
&\mathfrak{F}^{k+1}(\ddot{\bot})(\llbracket \texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0} \rrbracket_{\mathsf{in}} \dot{\sqcup} \llbracket s^{\ell_0} \rrbracket_{\mathsf{out}}) \\
&= \mathfrak{F}^{k+1}(\ddot{\bot})(\llbracket s^{\ell_0} \rrbracket_{\mathsf{in}}) && \text{(by eq.2)} \\
&= \mathfrak{F}(\mathfrak{F}^k(\ddot{\bot}))(\llbracket s^{\ell_0} \rrbracket_{\mathsf{in}}) && \text{(by def. of } \mathfrak{F}^{k+1}\text{)} \\
&= (\lambda\Phi.\, \lambda a.\, a \dot{\sqcup} \Phi(\mathcal{A}\llbracket s^{\ell_0} \rrbracket a))(\mathfrak{F}^k(\ddot{\bot}))(\llbracket s^{\ell_0} \rrbracket_{\mathsf{in}}) && \text{(by def. of } \mathfrak{F}\text{)} \\
&= (\lambda a.\, a \dot{\sqcup} \mathfrak{F}^k(\ddot{\bot})(\mathcal{A}\llbracket s^{\ell_0} \rrbracket a))(\llbracket s^{\ell_0} \rrbracket_{\mathsf{in}}) && (\beta\text{-reduction}) \\
&= \llbracket s^{\ell_0} \rrbracket_{\mathsf{in}} \dot{\sqcup} \mathfrak{F}^k(\ddot{\bot})(\mathcal{A}\llbracket s^{\ell_0} \rrbracket(\llbracket s^{\ell_0} \rrbracket_{\mathsf{in}})) && (\beta\text{-reduction}) \\
&\dot{\sqsubseteq} \llbracket s^{\ell_0} \rrbracket_{\mathsf{in}} \dot{\sqcup} \mathfrak{F}^k(\ddot{\bot})(\llbracket s^{\ell_0} \rrbracket_{\mathsf{out}}) && \text{(by outer IH, monotonicity of } \mathfrak{F}^k(\ddot{\bot})\text{)} \\
&\dot{\sqsubseteq} \llbracket s^{\ell_0} \rrbracket_{\mathsf{in}} \dot{\sqcup} \mathfrak{F}^k(\ddot{\bot})(\llbracket \texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0} \rrbracket_{\mathsf{in}} \dot{\sqcup} \llbracket s^{\ell_0} \rrbracket_{\mathsf{out}}) && \text{(by monotonicity of } \mathfrak{F}^k(\ddot{\bot})\text{)} \\
&\dot{\sqsubseteq} \llbracket s^{\ell_0} \rrbracket_{\mathsf{in}} \dot{\sqcup} \llbracket \texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0} \rrbracket_{\mathsf{out}} && \text{(by inner IH)} \\
&= \llbracket \texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0} \rrbracket_{\mathsf{out}} && \text{(by eq.1)}
\end{aligned}$$

□

H. Lifting of the collecting semantics

Lemma 23 (Expression collecting semantics equivalence)**.**

$$\forall e.\ \overline{\mathcal{C}'}[\![e]\!] = \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}'[\![e]\!](\pi_k(\overline{c}))$$

Proof. Let e be given. We proceed by structural induction on e.

Case n**:**

$$\begin{aligned}
&\lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}'[\![n]\!](\pi_k(\overline{c})) \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} (\lambda c.\ \{n\})(\pi_k(\overline{c})) && \text{(by def. of } \mathcal{C}'\text{)} \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \{n\} && (\beta\text{-reduction}) \\
&= \overline{\mathcal{C}'}[\![n]\!] && \text{(by def. of } \overline{\mathcal{C}'}\text{)}
\end{aligned}$$

Case `x`**:**

$$\begin{aligned}
&\lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}'[\![\mathtt{x}]\!](\pi_k(\overline{c})) \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} (\lambda c.\ \{\sigma(\mathtt{x}) \mid \sigma \in c\})(\pi_k(\overline{c})) && \text{(by def. of } \mathcal{C}'\text{)} \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \{\sigma(\mathtt{x}) \mid \sigma \in \pi_k(\overline{c})\} && (\beta\text{-reduction}) \\
&= \overline{\mathcal{C}'}[\![\mathtt{x}]\!] && \text{(by def. of } \overline{\mathcal{C}'}\text{)}
\end{aligned}$$

Case $e_0 \oplus e_1$**:**

$$\begin{aligned}
&\lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}'[\![e_0 \oplus e_1]\!](\pi_k(\overline{c})) \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} (\lambda c.\ \{v \mid v \in \{v_0\} \dot{\oplus} \{v_1\} \ \wedge\ \sigma \in c \ \wedge \\
&\qquad v_0 \in \mathcal{C}'[\![e_0]\!]\{\sigma\} \ \wedge\ v_1 \in \mathcal{C}'[\![e_1]\!]\{\sigma\}\})(\pi_k(\overline{c})) \\
&\qquad\qquad \text{(by def. of } \mathcal{C}'\text{)} \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \{v \mid v \in \{v_0\} \dot{\oplus} \{v_1\} \ \wedge\ \sigma \in \pi_k(\overline{c}) \ \wedge \\
&\qquad v_0 \in \mathcal{C}'[\![e_0]\!]\{\sigma\} \ \wedge\ v_1 \in \mathcal{C}'[\![e_1]\!]\{\sigma\}\} \\
&\qquad\qquad (\beta\text{-reduction}) \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \{v \mid v \in \{v_0\} \dot{\oplus} \{v_1\} \ \wedge\ \sigma \in \pi_k(\overline{c}) \ \wedge \\
&\qquad v_0 \in \mathcal{C}'[\![e_0]\!](\pi_k(\prod_{k'\in\mathbb{K}} \{\sigma\})) \ \wedge \\
&\qquad v_1 \in \mathcal{C}'[\![e_1]\!](\pi_k(\prod_{k'\in\mathbb{K}} \{\sigma\}))\} \\
&\qquad\qquad \text{(proj.+constr. expansion)} \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \{v \mid v \in \{v_0\} \dot{\oplus} \{v_1\} \ \wedge\ \sigma \in \pi_k(\overline{c}) \ \wedge \\
&\qquad v_0 \in \pi_k(\overline{\mathcal{C}'}[\![e_0]\!](\prod_{k'\in\mathbb{K}} \{\sigma\})) \ \wedge \\
&\qquad v_1 \in \pi_k(\overline{\mathcal{C}'}[\![e_1]\!](\prod_{k'\in\mathbb{K}} \{\sigma\}))\} && \text{(IH, twice)} \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \{v \mid \sigma \in \pi_k(\overline{c}) \ \wedge \\
&\qquad v \in (\pi_k(\overline{\mathcal{C}'}[\![e_0]\!](\prod_{k'\in\mathbb{K}} \{\sigma\}))) \dot{\oplus} (\pi_k(\overline{\mathcal{C}'}[\![e_1]\!](\prod_{k'\in\mathbb{K}} \{\sigma\})))\} \\
&\qquad\qquad \text{(simplify)} \\
&= \overline{\mathcal{C}'}[\![e_0 \oplus e_1]\!] && \text{(by def. of } \overline{\mathcal{C}'}\text{)}
\end{aligned}$$

□

Theorem 24 (Statement collecting semantics equivalence)**.**

$$\forall s \in \overline{Stm}.\ \overline{\mathcal{C}}[\![s]\!] = \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}[\![P[\![s]\!]_k]\!](\pi_k(\overline{c}))$$

Proof. Let s be given. We proceed by structural induction on s.

Case `skip`**:**

$$\begin{aligned}
&\lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}[\![P[\![\mathtt{skip}]\!]_k]\!](\pi_k(\overline{c})) \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}[\![\mathtt{skip}]\!](\pi_k(\overline{c})) && \text{(by def. of } P\text{)} \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} (\lambda c.\ c)(\pi_k(\overline{c})) && \text{(by def. of } \mathcal{C}\text{)} \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \pi_k(\overline{c}) && (\beta\text{-reduction}) \\
&= \lambda\overline{c}.\ \overline{c} && \text{(shortcut proj.+constr.)} \\
&= \overline{\mathcal{C}}[\![\mathtt{skip}]\!] && \text{(by def. of } \overline{\mathcal{C}}\text{)}
\end{aligned}$$

Case `x := e`**:**

$$\begin{aligned}
&\lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}[\![P[\![\mathtt{x := e}]\!]_k]\!](\pi_k(\overline{c})) \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}[\![\mathtt{x := e}]\!](\pi_k(\overline{c})) && \text{(by def. of } P\text{)} \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} (\lambda c.\ \{\sigma[\mathtt{x} \mapsto v] \mid \sigma \in c \wedge v \in \mathcal{C}'[\![e]\!]\{\sigma\}\})(\pi_k(\overline{c})) \\
&\qquad\qquad \text{(by def. of } \mathcal{C}\text{)} \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \{\sigma[\mathtt{x} \mapsto v] \mid \sigma \in \pi_k(\overline{c}) \wedge v \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \\
&\qquad\qquad (\beta\text{-reduction}) \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \{\sigma[\mathtt{x} \mapsto v] \mid \sigma \in \pi_k(\overline{c}) \wedge v \in \mathcal{C}'[\![e]\!](\pi_k(\prod_{k'\in\mathbb{K}} \{\sigma\}))\} \\
&\qquad\qquad \text{(proj.+constr. expansion)} \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \{\sigma[\mathtt{x} \mapsto v] \mid \sigma \in \pi_k(\overline{c}) \wedge v \in \pi_k(\overline{\mathcal{C}'}[\![e]\!](\prod_{k'\in\mathbb{K}} \{\sigma\}))\} \\
&\qquad\qquad \text{(by Lemma 23)} \\
&= \overline{\mathcal{C}}[\![\mathtt{x := e}]\!] && \text{(by def. of } \overline{\mathcal{C}}\text{)}
\end{aligned}$$

Case s_0 `;` s_1**:**

$$\begin{aligned}
&\lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}[\![P[\![s_0 \ \mathtt{;}\ s_1]\!]_k]\!](\pi_k(\overline{c})) \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}[\![P[\![s_0]\!]_k \ \mathtt{;}\ P[\![s_1]\!]_k]\!](\pi_k(\overline{c})) && \text{(by def. of } P\text{)} \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} (\mathcal{C}[\![P[\![s_1]\!]_k]\!] \circ \mathcal{C}[\![P[\![s_0]\!]_k]\!])(\pi_k(\overline{c})) \\
&\qquad\qquad \text{(by def. of } \mathcal{C}\text{)} \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}[\![P[\![s_1]\!]_k]\!](\mathcal{C}[\![P[\![s_0]\!]_k]\!](\pi_k(\overline{c}))) && \text{(by def. of } \circ\text{)} \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}[\![P[\![s_1]\!]_k]\!](\pi_k(\overline{\mathcal{C}}[\![s_0]\!]\overline{c})) && \text{(by IH)} \\
&= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \pi_k(\overline{\mathcal{C}}[\![s_1]\!](\overline{\mathcal{C}}[\![s_0]\!]\overline{c})) && \text{(by IH)} \\
&= \lambda\overline{c}.\ \overline{\mathcal{C}}[\![s_1]\!](\overline{\mathcal{C}}[\![s_0]\!]\overline{c}) && \text{(shortcut proj.+constr.)} \\
&= \lambda\overline{c}.\ (\overline{\mathcal{C}}[\![s_1]\!] \circ \overline{\mathcal{C}}[\![s_0]\!])\overline{c} && \text{(by def. of } \circ\text{)} \\
&= \overline{\mathcal{C}}[\![s_1]\!] \circ \overline{\mathcal{C}}[\![s_0]\!] && (\eta\text{-reduction}) \\
&= \overline{\mathcal{C}}[\![s_0 \ \mathtt{;}\ s_1]\!] && \text{(by def. of } \overline{\mathcal{C}}\text{)}
\end{aligned}$$

Case `if` e `then` s_0 `else` s_1**:**

$$\lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}[\![P[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!]_k]\!](\pi_k(\overline{c}))$$

$$= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}[\![\texttt{if } e \texttt{ then } P[\![s_0]\!]_k \texttt{ else } P[\![s_1]\!]_k]\!](\pi_k(\overline{c}))$$

(by def. of P)

$$= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} (\lambda c.\, \mathcal{C}[\![P[\![s_0]\!]_k]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \mathcal{C}[\![P[\![s_1]\!]_k]\!]\{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\})(\pi_k(\overline{c}))$$

(by def. of $\mathcal{C}$)

$$= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}[\![P[\![s_0]\!]_k]\!]\{\sigma \in \pi_k(\overline{c}) \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \mathcal{C}[\![P[\![s_1]\!]_k]\!]\{\sigma \in \pi_k(\overline{c}) \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\}$$

(β-reduction)

$$= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}[\![P[\![s_0]\!]_k]\!]\{\sigma \in \pi_k(\overline{c}) \mid 0 \notin \mathcal{C}'[\![e]\!](\pi_k(\prod_{k'\in\mathbb{K}} \{\sigma\}))\} \cup \mathcal{C}[\![P[\![s_1]\!]_k]\!]\{\sigma \in \pi_k(\overline{c}) \mid 0 \in \mathcal{C}'[\![e]\!](\pi_k(\prod_{k'\in\mathbb{K}} \{\sigma\}))\}$$

(proj.+constr. expansion)

$$= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}[\![P[\![s_0]\!]_k]\!]\{\sigma \in \pi_k(\overline{c}) \mid 0 \notin \pi_k(\overline{\mathcal{C}'}[\![e]\!](\prod_{k'\in\mathbb{K}} \{\sigma\}))\} \cup \mathcal{C}[\![P[\![s_1]\!]_k]\!]\{\sigma \in \pi_k(\overline{c}) \mid 0 \in \pi_k(\mathcal{C}'[\![e]\!](\prod_{k'\in\mathbb{K}} \{\sigma\}))\}$$

(by Lemma 23)

$$= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \mathcal{C}[\![P[\![s_0]\!]_k]\!](\pi_k(\prod_{k''\in\mathbb{K}} \{\sigma\in\pi_k(\overline{c}) \mid 0\notin\pi_k(\overline{\mathcal{C}'}[\![e]\!](\prod_{k'\in\mathbb{K}}\{\sigma\}))\})) \cup \mathcal{C}[\![P[\![s_1]\!]_k]\!](\pi_k(\prod_{k''\in\mathbb{K}} \{\sigma\in\pi_k(\overline{c}) \mid 0\in\pi_k(\mathcal{C}'[\![e]\!](\prod_{k'\in\mathbb{K}}\{\sigma\}))\}))$$

(proj.+constr. expansion, twice)

$$= \lambda\overline{c}.\ \prod_{k\in\mathbb{K}} \pi_k(\overline{\mathcal{C}}[\![s_0]\!](\prod_{k''\in\mathbb{K}} \{\sigma \in \pi_k(\overline{c}) \mid 0 \notin \pi_k(\overline{\mathcal{C}'}[\![e]\!](\prod_{k'\in\mathbb{K}} \{\sigma\}))\})) \cup \pi_k(\overline{\mathcal{C}}[\![s_1]\!](\prod_{k''\in\mathbb{K}} \{\sigma \in \pi_k(\overline{c}) \mid 0 \in \pi_k(\mathcal{C}'[\![e]\!](\prod_{k'\in\mathbb{K}} \{\sigma\}))\}))$$

(by IH, twice)

$$= \overline{\mathcal{C}}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!] \qquad \text{(by def. of } \overline{\mathcal{C}})$$

Case `while` e `do` s**:** We first define α_k and γ_k as follows:

$$\alpha_k : \mathbb{C}^{\mathbb{K}} \to \mathbb{C}$$
$$\alpha_k(\overline{c}) = \pi_k(\overline{c})$$
$$\gamma_k : \mathbb{C} \to \mathbb{C}^{\mathbb{K}}$$
$$\gamma_k(c) = \prod_{k'\in\mathbb{K}} \begin{cases} c & k = k' \\ \mathit{Store} & k \neq k' \end{cases}$$

Together they constitute a k-specific abstraction from the lifted collecting semantics level to the single-program collecting semantics level:

$$\langle \mathbb{C}^{\mathbb{K}}, \dot{\subseteq} \rangle \underset{\alpha_k}{\overset{\gamma_k}{\leftrightarrows}} \langle \mathbb{C}, \subseteq \rangle$$

Projections such as this are well-known to be Galois connections. In particular it is a Galois insertion:

$$(\alpha_k \circ \gamma_k)(c)$$
$$= \pi_k(\prod_{k'\in\mathbb{K}} \begin{cases} c & k = k' \\ \mathit{Store} & k \neq k' \end{cases}) \qquad \text{(by def. of } \alpha_k, \gamma_k)$$
$$= c \qquad (\beta\text{-reduction})$$

We now lift this Galois connection to monotone transfer functions by a higher-order Galois connection:

$$\alpha_{\to} : (\mathbb{C}^{\mathbb{K}} \xrightarrow{m} \mathbb{C}^{\mathbb{K}}) \to \mathbb{C} \xrightarrow{m} \mathbb{C}$$
$$\alpha_{\to}(\overline{\Phi}) = \alpha_k \circ \overline{\Phi} \circ \gamma_k$$
$$\gamma_{\to} : (\mathbb{C} \xrightarrow{m} \mathbb{C}) \to \mathbb{C}^{\mathbb{K}} \xrightarrow{m} \mathbb{C}^{\mathbb{K}}$$
$$\gamma_{\to}(\Phi) = \gamma_k \circ \Phi \circ \alpha_k$$

$$\langle \mathbb{C}^{\mathbb{K}} \xrightarrow{m} \mathbb{C}^{\mathbb{K}}, \ddot{\subseteq} \rangle \underset{\alpha_{\to}}{\overset{\gamma_{\to}}{\leftrightarrows}} \langle \mathbb{C} \xrightarrow{m} \mathbb{C}, \ddot{\subseteq} \rangle$$

Now observe that α_k induces a complete abstraction over the body of the loop:

$$(\lambda\Phi.\ \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \Phi(\mathcal{C}[\![P[\![s]\!]_k]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})) \circ \alpha_{\to}$$
$$= (\lambda\Phi.\ \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \Phi(\mathcal{C}[\![P[\![s]\!]_k]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})) \circ (\lambda\overline{\Phi}.\ \alpha_k \circ \overline{\Phi} \circ \gamma_k)$$

(by def. of $\alpha_{\to}$)

$$= \lambda\overline{\Phi}.\ (\lambda\Phi.\ \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \Phi(\mathcal{C}[\![P[\![s]\!]_k]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}))(\alpha_k \circ \overline{\Phi} \circ \gamma_k)$$

(by def. of $\circ$)

$$= \lambda\overline{\Phi}.\ \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup (\alpha_k \circ \overline{\Phi} \circ \gamma_k)(\mathcal{C}[\![P[\![s]\!]_k]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})$$

(β-reduction)

$$= \lambda\overline{\Phi}.\ \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup (\pi_k \circ \overline{\Phi} \circ \gamma_k)(\mathcal{C}[\![P[\![s]\!]_k]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})$$

(by def. of α_k)

$$= \lambda\overline{\Phi}.\ \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \pi_k(\overline{\Phi}(\gamma_k(\mathcal{C}[\![P[\![s]\!]_k]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})))$$

(by def. of $\circ$)

$$= \lambda\overline{\Phi}.\ \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \pi_k(\overline{\Phi}(\prod_{k'\in\mathbb{K}} \begin{cases} (\mathcal{C}[\![P[\![s]\!]_k]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}) & k = k' \\ \mathit{Store} & k \neq k' \end{cases}))$$

(by def. of γ_k)

$$= \lambda\overline{\Phi}.\ \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \pi_k(\prod_{k'\in\mathbb{K}} \pi_{k'}(\overline{\Phi})(\begin{cases} (\mathcal{C}[\![P[\![s]\!]_k]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}) & k = k' \\ \mathit{Store} & k \neq k' \end{cases}))$$

(by def. of appl.)

$$= \lambda\overline{\Phi}.\ \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \pi_k(\prod_{k'\in\mathbb{K}} \pi_{k'}(\overline{\Phi})(\mathcal{C}[\![P[\![s]\!]_{k'}]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}))$$

(identical k entries)

$$= \lambda\overline{\Phi}.\ \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \pi_k(\overline{\Phi})(\mathcal{C}[\![P[\![s]\!]_k]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})$$

(by def. of π_k)

$$= \lambda\overline{\Phi}.\ \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \pi_k(\overline{\Phi})(\mathcal{C}[\![P[\![s]\!]_k]\!](\alpha_k(\gamma_k(\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}))))$$

(Galois insertion)

$$= \lambda\overline{\Phi}.\ \lambda c.\ \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \pi_k(\overline{\Phi})(\mathcal{C}[\![P[\![s]\!]_k]\!](\pi_k(\gamma_k(\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}))))$$

(by def. of α_k)

$$= \lambda\overline{\Phi}.\, \lambda c.\, \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \pi_k(\overline{\Phi})(\mathcal{C}[\![P[\![s]\!]_k]\!](\pi_k(\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\gamma_k(c)) \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}))) \quad \text{(identical } k \text{ entries)}$$

$$= \lambda\overline{\Phi}.\, \lambda c.\, \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \pi_k(\overline{\Phi}(\prod_{k' \in \mathbb{K}} \mathcal{C}[\![P[\![s]\!]_{k'}]\!](\pi_{k'}(\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\gamma_k(c)) \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})))) \quad \text{(by def. of appl.)}$$

$$= \lambda\overline{\Phi}.\, \lambda c.\, \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \pi_k(\overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\gamma_k(c)) \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}))) \quad \text{(by IH)}$$

$$= \lambda\overline{\Phi}.\, \lambda c.\, \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \alpha_k(\overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\gamma_k(c)) \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}))) \quad \text{(by def. of } \alpha_k)$$

$$= \lambda\overline{\Phi}.\, \lambda c.\, \alpha_k(\gamma_k(\{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\})) \cup \alpha_k(\overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\gamma_k(c)) \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}))) \quad \text{(Galois insertion)}$$

$$= \lambda\overline{\Phi}.\, \lambda c.\, \alpha_k(\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\gamma_k(c)) \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\}) \cup \alpha_k(\overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\gamma_k(c)) \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}))) \quad \text{(identical } k \text{ entries)}$$

$$= \lambda\overline{\Phi}.\, \lambda c.\, \alpha_k(\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\gamma_k(c)) \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\}) \dot{\cup}\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\gamma_k(c)) \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})) \quad (\alpha_k \text{ a CJM})$$

$$= \lambda\overline{\Phi}.\, \lambda c.\, (\lambda\overline{c}.\, \alpha_k(\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\}) \dot{\cup}\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})))(\gamma_k(c)) \quad (\beta \text{ expansion})$$

$$= \lambda\overline{\Phi}.\, \lambda c.\, (\alpha_k \circ (\lambda\overline{c}.\, \prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \dot{\cup}\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})))(\gamma_k(c)) \quad \text{(by def. of } \circ)$$

$$= \lambda\overline{\Phi}.\, \alpha_k \circ (\lambda\overline{c}.\, \prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \dot{\cup}\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})) \circ (\lambda c.\, \gamma_k(c)) \quad \text{(by def. of } \circ)$$

$$= \lambda\overline{\Phi}.\, \alpha_k \circ (\lambda\overline{c}.\, \prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \dot{\cup}\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})) \circ \gamma_k \quad (\eta\text{-reduction})$$

$$= \lambda\overline{\Phi}.\, \alpha_{\to}(\lambda\overline{c}.\, \prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \dot{\cup}\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}))) \quad \text{(by def. of } \alpha_{\to})$$

$$= \alpha_{\to} \circ (\lambda\overline{\Phi}.\, \lambda\overline{c}.\, \prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \dot{\cup}\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\}))) \quad \text{(by def. of } \circ)$$

$$= \alpha_{\to} \circ (\lambda\overline{\Phi}.\, \lambda\overline{c}.\, \prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \mathcal{C}'[\![e]\!](\pi_{k''}(\dot{\varnothing}[k'' \mapsto \{\sigma\}])\} \dot{\cup}\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \mathcal{C}'[\![e]\!](\pi_{k''}(\dot{\varnothing}[k'' \mapsto \{\sigma\}])\}))) \quad \text{(proj.+const. expansion)}$$

$$= \alpha_{\to} \circ (\lambda\overline{\Phi}.\, \lambda\overline{c}.\, \prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}])\} \dot{\cup}\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}])\}))) \quad \text{(by Lemma 23)}$$

As a consequence, we can now apply the stronger fixed point theorem:

$$\begin{aligned}
&\alpha_{\to}(\overline{\mathcal{C}}[\![\texttt{while } e \texttt{ do } s]\!]) \\
&= \alpha_{\to}(\mathrm{lfp}(\lambda\overline{\Phi}.\, \lambda\overline{c}.\, \prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}])\} \dot{\cup}\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}])\})))) && \text{(by def. of } \overline{\mathcal{C}}) \\
&= \mathrm{lfp}(\lambda\Phi.\, \lambda c.\, \{\sigma \in c \mid 0 \in \mathcal{C}'[\![e]\!]\{\sigma\}\} \cup \Phi(\mathcal{C}[\![P[\![s]\!]_k]\!]\{\sigma \in c \mid 0 \notin \mathcal{C}'[\![e]\!]\{\sigma\}\})) && \text{(by above)} \\
&= \mathcal{C}[\![\texttt{while } e \texttt{ do } P[\![s]\!]_k]\!] && \text{(by def. of } \mathcal{C}) \\
&= \mathcal{C}[\![P[\![\texttt{while } e \texttt{ do } s]\!]_k]\!] && \text{(by def. of } P)
\end{aligned}$$

Substituting equals for equals we now obtain:

$$\begin{aligned}
&\lambda\overline{c}.\, \prod_{k \in \mathbb{K}} \mathcal{C}[\![P[\![\texttt{while } e \texttt{ do } s]\!]_k]\!](\pi_k(\overline{c})) \\
&= \lambda\overline{c}.\, \prod_{k \in \mathbb{K}} \alpha_{\to}(\overline{\mathcal{C}}[\![\texttt{while } e \texttt{ do } s]\!])(\pi_k(\overline{c})) && \text{(by above equality)} \\
&= \lambda\overline{c}.\, \prod_{k \in \mathbb{K}} (\alpha_k \circ \overline{\mathcal{C}}[\![\texttt{while } e \texttt{ do } s]\!] \circ \gamma_k)(\pi_k(\overline{c})) && \text{(by def. of } \alpha_{\to}) \\
&= \lambda\overline{c}.\, \prod_{k \in \mathbb{K}} \alpha_k(\overline{\mathcal{C}}[\![\texttt{while } e \texttt{ do } s]\!](\gamma_k(\pi_k(\overline{c})))) && \text{(by def. of } \circ) \\
&= \lambda\overline{c}.\, \prod_{k \in \mathbb{K}} \alpha_k(\overline{\mathcal{C}}[\![\texttt{while } e \texttt{ do } s]\!](\prod_{k' \in \mathbb{K}} \begin{cases} \pi_k(\overline{c}) & k = k' \\ \mathit{Store} & k \neq k' \end{cases})) && \text{(by def. of } \gamma_k) \\
&= \lambda\overline{c}.\, \prod_{k \in \mathbb{K}} \alpha_k(\prod_{k' \in \mathbb{K}} (\pi_{k'}(\overline{\mathcal{C}}[\![\texttt{while } e \texttt{ do } s]\!]))(\begin{cases} \pi_k(\overline{c}) & k = k' \\ \mathit{Store} & k \neq k' \end{cases})) && \text{(by def. of appl.)} \\
&= \lambda\overline{c}.\, \prod_{k \in \mathbb{K}} (\pi_k(\overline{\mathcal{C}}[\![\texttt{while } e \texttt{ do } s]\!]))(\pi_k(\overline{c})) && \text{(by def. of } \alpha_k) \\
&= \lambda\overline{c}.\, \overline{\mathcal{C}}[\![\texttt{while } e \texttt{ do } s]\!]\overline{c} && \text{(by def. of appl.)} \\
&= \overline{\mathcal{C}}[\![\texttt{while } e \texttt{ do } s]\!] && (\eta\text{-reduce})
\end{aligned}$$

as we desired.

Case $\texttt{\#if}\ \varphi\ s$**:**

$$
\begin{aligned}
&\lambda\overline{c}.\prod_{k\in\mathbb{K}}\mathcal{C}[\![P[\![\texttt{\#if}\ \varphi\ s]\!]_k]\!](\pi_k(\overline{c})) \\
&= \lambda\overline{c}.\prod_{k\in\mathbb{K}}\begin{cases}\mathcal{C}[\![P[\![s]\!]_k]\!](\pi_k(\overline{c})) & k\vDash\varphi\\ \mathcal{C}[\![\texttt{skip}]\!](\pi_k(\overline{c})) & k\nvDash\varphi\end{cases} && \text{(by def. of } P\text{)}\\
&= \lambda\overline{c}.\prod_{k\in\mathbb{K}}\begin{cases}\mathcal{C}[\![P[\![s]\!]_k]\!](\pi_k(\overline{c})) & k\vDash\varphi\\ \pi_k(\overline{c}) & k\nvDash\varphi\end{cases} && \text{(by def. of } \mathcal{C}\text{)}\\
&= \lambda\overline{c}.\prod_{k\in\mathbb{K}}\begin{cases}\pi_k(\overline{\mathcal{C}}[\![s]\!]\overline{c}) & k\vDash\varphi\\ \pi_k(\overline{c}) & k\nvDash\varphi\end{cases} && \text{(by IH)}\\
&= \lambda\overline{c}.\prod_{k\in\mathbb{K}}\{\sigma\in\pi_k(\overline{\mathcal{C}}[\![s]\!]\overline{c})\mid k\vDash\varphi\}\cup\{\sigma\in\pi_k(\overline{c})\mid k\nvDash\varphi\} \\
& && \text{(by def. of } \cup\text{)}\\
&= \overline{\mathcal{C}}[\![\texttt{\#if}\ \varphi\ s]\!] && \text{(by def. of } \overline{\mathcal{C}}\text{)}
\end{aligned}
$$

□

I. Lifting of the approximate semantics

Lemma 25 (Approximate expression semantics lifting)**.**

$$\forall e \in \mathit{Exp}.\ \overline{\mathcal{B}'}[\![e]\!] = \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \mathcal{B}'[\![e]\!](\pi_k(\bar{b}))$$

Proof. Let e be given. We proceed by structural induction on e.

Case n**:**

$$\begin{aligned}
&\lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \mathcal{B}'[\![n]\!](\pi_k(\bar{b})) \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} (\lambda b.\ \{n\})(\pi_k(\bar{b})) && \text{(by def. of } \mathcal{B}') \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \{n\} && (\beta\text{-reduction}) \\
&= \overline{\mathcal{B}'}[\![n]\!] && \text{(by def. of } \overline{\mathcal{B}'})
\end{aligned}$$

Case `x`**:**

$$\begin{aligned}
&\lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \mathcal{B}'[\![\mathtt{x}]\!](\pi_k(\bar{b})) \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} (\lambda b.\ b(\mathtt{x}))(\pi_k(\bar{b})) && \text{(by def. of } \mathcal{B}') \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \pi_k(\bar{b})(\mathtt{x}) && (\beta\text{-reduction}) \\
&= \overline{\mathcal{B}'}[\![\mathtt{x}]\!] && \text{(by def. of } \overline{\mathcal{B}'})
\end{aligned}$$

Case $e_0 \oplus e_1$**:**

$$\begin{aligned}
&\lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \mathcal{B}'[\![e_0 \oplus e_1]\!](\pi_k(\bar{b})) \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} (\lambda b.\ \mathcal{B}'[\![e_0]\!]b \mathbin{\dot{\oplus}} \mathcal{B}'[\![e_1]\!]b)(\pi_k(\bar{b})) && \text{(by def. of } \mathcal{B}') \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \mathcal{B}'[\![e_0]\!](\pi_k(\bar{b})) \mathbin{\dot{\oplus}} \mathcal{B}'[\![e_1]\!](\pi_k(\bar{b})) && (\beta\text{-reduction}) \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \pi_k(\overline{\mathcal{B}'}[\![e_0]\!]\bar{b}) \mathbin{\dot{\oplus}} \pi_k(\overline{\mathcal{B}'}[\![e_1]\!]\bar{b}) && \text{(by IH, twice)} \\
&= \overline{\mathcal{B}'}[\![e_0 \oplus e_1]\!] && \text{(by def. of } \overline{\mathcal{B}'})
\end{aligned}$$

□

Theorem 26 (Approximate statement semantics lifting)**.**

$$\forall s \in \overline{\mathit{Stm}}.\ \overline{\mathcal{B}}[\![s]\!] = \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \mathcal{B}[\![P[\![s]\!]_k]\!](\pi_k(\bar{b}))$$

Proof. Let s be given. We proceed by structural induction on s.

Case `skip`**:**

$$\begin{aligned}
&\lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \mathcal{B}[\![P[\![\mathtt{skip}]\!]_k]\!](\pi_k(\bar{b})) \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \mathcal{B}[\![\mathtt{skip}]\!](\pi_k(\bar{b})) && \text{(by def. of } P) \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \pi_k(\bar{b}) && \text{(by def. of } \mathcal{B}) \\
&= \lambda \bar{b}.\ \bar{b} && \text{(shortcut proj.+constr.)} \\
&= \overline{\mathcal{B}}[\![\mathtt{skip}]\!] && \text{(by def. of } \overline{\mathcal{B}})
\end{aligned}$$

Case `x := e`**:**

$$\begin{aligned}
&\lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \mathcal{B}[\![P[\![\mathtt{x} := e]\!]_k]\!](\pi_k(\bar{b})) \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \mathcal{B}[\![\mathtt{x} := e]\!](\pi_k(\bar{b})) && \text{(by def. of } P) \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} (\lambda b.\ b[\mathtt{x} \mapsto \mathcal{B}'[\![e]\!]b])(\pi_k(\bar{b})) && \text{(by def. of } \mathcal{B}) \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} (\pi_k(\bar{b}))[\mathtt{x} \mapsto \mathcal{B}'[\![e]\!](\pi_k(\bar{b}))] && (\beta\text{-reduction}) \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} (\pi_k(\bar{b}))[\mathtt{x} \mapsto \pi_k(\overline{\mathcal{B}'}[\![e]\!]\bar{b})] && \text{(by Lemma 25)} \\
&= \overline{\mathcal{B}}[\![\mathtt{x} := e]\!] && \text{(by def. of } \overline{\mathcal{B}})
\end{aligned}$$

Case s_0 `;` s_1**:**

$$\begin{aligned}
&\lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \mathcal{B}[\![P[\![s_0 \ \mathtt{;}\ s_1]\!]_k]\!](\pi_k(\bar{b})) \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \mathcal{B}[\![P[\![s_0]\!]_k \ \mathtt{;}\ P[\![s_1]\!]_k]\!](\pi_k(\bar{b})) && \text{(by def. of } P) \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} (\mathcal{B}[\![P[\![s_1]\!]_k]\!] \circ \mathcal{B}[\![P[\![s_0]\!]_k]\!])(\pi_k(\bar{b})) \\
& && \text{(by def. of } \mathcal{B}) \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \mathcal{B}[\![P[\![s_1]\!]_k]\!](\mathcal{B}[\![P[\![s_0]\!]_k]\!](\pi_k(\bar{b}))) && \text{(by def. of } \circ) \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \mathcal{B}[\![P[\![s_1]\!]_k]\!](\pi_k(\overline{\mathcal{B}}[\![s_0]\!]\bar{b})) && \text{(by IH)} \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \pi_k(\overline{\mathcal{B}}[\![s_1]\!](\overline{\mathcal{B}}[\![s_0]\!]\bar{b})) && \text{(by IH)} \\
&= \lambda \bar{b}.\ \overline{\mathcal{B}}[\![s_1]\!](\overline{\mathcal{B}}[\![s_0]\!]\bar{b}) && \text{(shortcut proj.+constr.)} \\
&= \lambda \bar{b}.\ (\overline{\mathcal{B}}[\![s_1]\!] \circ \overline{\mathcal{B}}[\![s_0]\!])\bar{b} && \text{(by def. of } \circ) \\
&= \overline{\mathcal{B}}[\![s_1]\!] \circ \overline{\mathcal{B}}[\![s_0]\!] && (\eta\text{-reduction}) \\
&= \overline{\mathcal{B}}[\![s_0 \ \mathtt{;}\ s_1]\!] && \text{(by def. of } \overline{\mathcal{B}})
\end{aligned}$$

Case `if` e `then` s_0 `else` s_1**:**

$$\begin{aligned}
&\lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \mathcal{B}[\![P[\![\mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1]\!]_k]\!](\pi_k(\bar{b})) \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \mathcal{B}[\![\mathtt{if}\ e\ \mathtt{then}\ P[\![s_0]\!]_k\ \mathtt{else}\ P[\![s_1]\!]_k]\!](\pi_k(\bar{b})) \\
& && \text{(by def. of } P) \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \mathcal{B}[\![P[\![s_0]\!]_k]\!](\pi_k(\bar{b})) \mathbin{\dot{\cup}} \mathcal{B}[\![P[\![s_1]\!]_k]\!](\pi_k(\bar{b})) \\
& && \text{(by def. of } \mathcal{B}) \\
&= \lambda \bar{b}.\ \prod_{k \in \mathbb{K}} \pi_k(\overline{\mathcal{B}}[\![s_0]\!]\bar{b}) \mathbin{\dot{\cup}} \pi_k(\overline{\mathcal{B}}[\![s_1]\!]\bar{b}) && \text{(by IH, twice)} \\
&= \lambda \bar{b}.\ \overline{\mathcal{B}}[\![s_0]\!]\bar{b} \mathbin{\ddot{\cup}} \overline{\mathcal{B}}[\![s_1]\!]\bar{b} && \text{(by def. of } \ddot{\cup}) \\
&= \overline{\mathcal{B}}[\![\mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1]\!]\bar{b} && \text{(by def. of } \overline{\mathcal{B}})
\end{aligned}$$

Case `while` e `do` s**:** We first define α_k and γ_k as follows:

$$\begin{aligned}
&\alpha_k : \mathbb{B}^{\mathbb{K}} \to \mathbb{B} \\
&\alpha_k(\bar{b}) = \pi_k(\bar{b}) \\
&\gamma_k : \mathbb{B} \to \mathbb{B}^{\mathbb{K}} \\
&\gamma_k(b) = \prod_{k' \in \mathbb{K}} \begin{cases} b & k = k' \\ \lambda \mathtt{x}.\ \mathit{Val} & k \neq k' \end{cases}
\end{aligned}$$

Together they constitute a k-specific abstraction from the lifted approximate semantics level to the single-program approximate

semantics level:

$$\langle \mathbb{B}^{\mathbb{K}}, \ddot{\subseteq} \rangle \underset{\alpha_k}{\overset{\gamma_k}{\leftrightarrows}} \langle \mathbb{B}, \dot{\subseteq} \rangle$$

Projections such as this are well-known to be Galois connections. In particular it is a Galois insertion:

$$\begin{aligned}
&(\alpha_k \circ \gamma_k)(b) \\
&= \pi_k(\prod_{k' \in \mathbb{K}} \begin{cases} b & k = k' \\ \lambda \mathtt{x}.\ \mathit{Val} & k \neq k' \end{cases}) && \text{(by def. of } \alpha_k, \gamma_k) \\
&= b && (\beta\text{-reduction})
\end{aligned}$$

We now lift this Galois connection to monotone transfer functions by a higher-order Galois connection:

$$\begin{aligned}
&\alpha_{\rightarrow} : (\mathbb{B}^{\mathbb{K}} \xrightarrow{m} \mathbb{B}^{\mathbb{K}}) \rightarrow \mathbb{B} \xrightarrow{m} \mathbb{B} \\
&\alpha_{\rightarrow}(\overline{\Phi}) = \alpha_k \circ \overline{\Phi} \circ \gamma_k \\
&\gamma_{\rightarrow} : (\mathbb{B} \xrightarrow{m} \mathbb{B}) \rightarrow \mathbb{B}^{\mathbb{K}} \xrightarrow{m} \mathbb{B}^{\mathbb{K}} \\
&\gamma_{\rightarrow}(\Phi) = \gamma_k \circ \Phi \circ \alpha_k
\end{aligned}$$

$$\langle \mathbb{B}^{\mathbb{K}} \xrightarrow{m} \mathbb{B}^{\mathbb{K}}, \ddot{\subseteq} \rangle \underset{\alpha_{\rightarrow}}{\overset{\gamma_{\rightarrow}}{\leftrightarrows}} \langle \mathbb{B} \xrightarrow{m} \mathbb{B}, \ddot{\subseteq} \rangle$$

Now observe that α_k induces a complete abstraction over the body of the loop:

$$\begin{aligned}
&\alpha_{\rightarrow} \circ (\lambda \overline{\Phi}.\ \lambda \bar{b}.\ \bar{b} \ddot{\cup} \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\bar{b})) \\
&= \lambda \overline{\Phi}.\ \alpha_{\rightarrow}(\lambda \bar{b}.\ \bar{b} \ddot{\cup} \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\bar{b})) && \text{(by def. of } \circ) \\
&= \lambda \overline{\Phi}.\ \alpha_k \circ (\lambda \bar{b}.\ \bar{b} \ddot{\cup} \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\bar{b})) \circ \gamma_k && \text{(by def. of } \alpha_{\rightarrow}) \\
&= \lambda \overline{\Phi}.\ \alpha_k \circ (\lambda \bar{b}.\ \bar{b} \ddot{\cup} \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\bar{b})) \circ (\lambda b.\ \gamma_k(b)) \\
&&& (\eta\text{-expansion}) \\
&= \lambda \overline{\Phi}.\ \lambda b.\ (\alpha_k \circ (\lambda \bar{b}.\ \bar{b} \ddot{\cup} \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\bar{b})))(\gamma_k(b)) \\
&&& \text{(by def. of } \circ) \\
&= \lambda \overline{\Phi}.\ \lambda b.\ (\lambda \bar{b}.\ \alpha_k(\bar{b} \ddot{\cup} \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\bar{b})))(\gamma_k(b)) && \text{(by def. of } \circ) \\
&= \lambda \overline{\Phi}.\ \lambda b.\ \alpha_k(\gamma_k(b) \ddot{\cup} \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!](\gamma_k(b)))) && (\beta\text{-reduction}) \\
&= \lambda \overline{\Phi}.\ \lambda b.\ \alpha_k(\gamma_k(b)) \ \dot{\cup}\ \alpha_k(\overline{\Phi}(\overline{\mathcal{B}}[\![s]\!](\gamma_k(b)))) && (\alpha_k \text{ a CJM}) \\
&= \lambda \overline{\Phi}.\ \lambda b.\ b \ \dot{\cup}\ \alpha_k(\overline{\Phi}(\overline{\mathcal{B}}[\![s]\!](\gamma_k(b)))) && \text{(Galois insertion)} \\
&= \lambda \overline{\Phi}.\ \lambda b.\ b \ \dot{\cup}\ \alpha_k(\overline{\Phi}(\prod_{k' \in \mathbb{K}} \mathcal{B}[\![P[\![s]\!]_{k'}]\!](\pi_{k'}(\gamma_k(b))))) \\
&&& \text{(by IH)} \\
&= \lambda \overline{\Phi}.\ \lambda b.\ b \ \dot{\cup}\ \alpha_k(\prod_{k' \in \mathbb{K}} \pi_{k'}(\overline{\Phi})(\mathcal{B}[\![P[\![s]\!]_{k'}]\!](\pi_{k'}(\gamma_k(b))))) \\
&&& \text{(by def. of appl.)} \\
&= \lambda \overline{\Phi}.\ \lambda b.\ b \ \dot{\cup}\ \pi_k(\overline{\Phi})(\mathcal{B}[\![P[\![s]\!]_k]\!](\pi_k(\gamma_k(b)))) \\
&&& \text{(by def. of } \alpha_k) \\
&= \lambda \overline{\Phi}.\ \lambda b.\ b \ \dot{\cup}\ \pi_k(\overline{\Phi})(\mathcal{B}[\![P[\![s]\!]_k]\!](\alpha_k(\gamma_k(b)))) \\
&&& \text{(by def. of } \alpha_k) \\
&= \lambda \overline{\Phi}.\ \lambda b.\ b \ \dot{\cup}\ \pi_k(\overline{\Phi})(\mathcal{B}[\![P[\![s]\!]_k]\!]b) && \text{(Galois insertion)} \\
&= \lambda \overline{\Phi}.\ \lambda b.\ b \ \dot{\cup}\ \pi_k(\prod_{k' \in \mathbb{K}} \pi_{k'}(\overline{\Phi}) \begin{cases} (\mathcal{B}[\![P[\![s]\!]_k]\!]b) & k = k' \\ \lambda \mathtt{x}.\ \mathit{Val} & k \neq k' \end{cases}) \\
&&& \text{(identical } k \text{ entries)} \\
&= \lambda \overline{\Phi}.\ \lambda b.\ b \ \dot{\cup}\ \pi_k(\overline{\Phi}(\prod_{k' \in \mathbb{K}} \begin{cases} (\mathcal{B}[\![P[\![s]\!]_k]\!]b) & k = k' \\ \lambda \mathtt{x}.\ \mathit{Val} & k \neq k' \end{cases})) \\
&&& \text{(by def. of appl.)} \\
&= \lambda \overline{\Phi}.\ \lambda b.\ b \ \dot{\cup}\ \pi_k(\overline{\Phi}(\gamma_k(\mathcal{B}[\![P[\![s]\!]_k]\!]b))) && \text{(by def. of } \gamma_k) \\
&= \lambda \overline{\Phi}.\ \lambda b.\ b \ \dot{\cup}\ (\pi_k \circ \overline{\Phi} \circ \gamma_k)(\mathcal{B}[\![P[\![s]\!]_k]\!]b) && \text{(by def. of } \circ) \\
&= \lambda \overline{\Phi}.\ \lambda b.\ b \ \dot{\cup}\ (\alpha_k \circ \overline{\Phi} \circ \gamma_k)(\mathcal{B}[\![P[\![s]\!]_k]\!]b) \\
&&& \text{(by def. of } \alpha_k) \\
&= \lambda \overline{\Phi}.\ (\lambda \Phi.\ \lambda b.\ b \ \dot{\cup}\ \Phi(\mathcal{B}[\![P[\![s]\!]_k]\!]b))(\alpha_k \circ \overline{\Phi} \circ \gamma_k) \\
&&& (\beta\text{-expansion}) \\
&= (\lambda \Phi.\ \lambda b.\ b \ \dot{\cup}\ \Phi(\mathcal{B}[\![P[\![s]\!]_k]\!]b)) \circ (\lambda \overline{\Phi}.\ \alpha_k \circ \overline{\Phi} \circ \gamma_k) \\
&&& \text{(by def. of } \circ) \\
&= (\lambda \Phi.\ \lambda b.\ b \ \dot{\cup}\ \Phi(\mathcal{B}[\![P[\![s]\!]_k]\!]b)) \circ \alpha_{\rightarrow} && \text{(by def. of } \alpha_{\rightarrow})
\end{aligned}$$

As a consequence, we can now apply the stronger fixed point theorem:

$$\begin{aligned}
&\alpha_{\rightarrow}(\overline{\mathcal{B}}[\![\mathtt{while}\ e\ \mathtt{do}\ s]\!]) \\
&= \alpha_{\rightarrow}(\mathrm{lfp}\, \lambda \overline{\Phi}.\ \lambda \bar{b}.\ \bar{b} \ddot{\cup} \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\bar{b})) && \text{(by def. of } \overline{\mathcal{B}}) \\
&= \mathrm{lfp}\, \lambda \Phi.\ \lambda b.\ b \ \dot{\cup}\ \Phi(\mathcal{B}[\![P[\![s]\!]_k]\!]b) && \text{(by above)} \\
&= \mathcal{B}[\![\mathtt{while}\ e\ \mathtt{do}\ P[\![s]\!]_k]\!] && \text{(by def. of } \mathcal{B}) \\
&= \mathcal{B}[\![P[\![\mathtt{while}\ e\ \mathtt{do}\ s]\!]_k]\!] && \text{(by def. of } P)
\end{aligned}$$

Substituting equals for equals we now obtain:

$$\lambda\bar{b}.\prod_{k\in\mathbb{K}}\mathcal{B}[\![P[\![\texttt{while}\ e\ \texttt{do}\ s]\!]_k]\!](\pi_k(\bar{b}))$$

$$= \lambda\bar{b}.\prod_{k\in\mathbb{K}}\alpha_{\rightarrow}(\overline{\mathcal{B}}[\![\texttt{while}\ e\ \texttt{do}\ s]\!])(\pi_k(\bar{b}))$$

(by above equality)

$$= \lambda\bar{b}.\prod_{k\in\mathbb{K}}(\alpha_k\circ\overline{\mathcal{B}}[\![\texttt{while}\ e\ \texttt{do}\ s]\!]\circ\gamma_k)(\pi_k(\bar{b}))$$

(by def. of $\alpha_{\rightarrow}$)

$$= \lambda\bar{b}.\prod_{k\in\mathbb{K}}\alpha_k(\overline{\mathcal{B}}[\![\texttt{while}\ e\ \texttt{do}\ s]\!](\gamma_k(\pi_k(\bar{b})))) \quad \text{(by def. of } \circ)$$

$$= \lambda\bar{b}.\prod_{k\in\mathbb{K}}\alpha_k(\overline{\mathcal{B}}[\![\texttt{while}\ e\ \texttt{do}\ s]\!](\prod_{k'\in\mathbb{K}}\begin{cases}\pi_k(\bar{b}) & k=k'\\ \lambda\texttt{x}.\ Val & k\neq k'\end{cases}))$$

(by def. of γ_k)

$$= \lambda\bar{b}.\prod_{k\in\mathbb{K}}\alpha_k(\prod_{k'\in\mathbb{K}}(\pi_{k'}(\overline{\mathcal{B}}[\![\texttt{while}\ e\ \texttt{do}\ s]\!]))(\begin{cases}\pi_k(\bar{b}) & k=k'\\ \lambda\texttt{x}.\ Val & k\neq k'\end{cases}))$$

(by def. of appl.)

$$= \lambda\bar{b}.\prod_{k\in\mathbb{K}}(\pi_k(\overline{\mathcal{B}}[\![\texttt{while}\ e\ \texttt{do}\ s]\!]))(\pi_k(\bar{b})) \quad \text{(by def. of } \alpha_k)$$

$$= \lambda\bar{b}.\overline{\mathcal{B}}[\![\texttt{while}\ e\ \texttt{do}\ s]\!]\bar{b} \quad \text{(by def. of appl.)}$$

$$= \overline{\mathcal{B}}[\![\texttt{while}\ e\ \texttt{do}\ s]\!] \quad (\eta\text{-reduce})$$

as we desired.

Case $\texttt{\#if}\ \varphi\ s$**:**

$$\lambda\bar{b}.\prod_{k\in\mathbb{K}}\mathcal{B}[\![P[\![\texttt{\#if}\ \varphi\ s]\!]_k]\!](\pi_k(\bar{b}))$$

$$= \lambda\bar{b}.\prod_{k\in\mathbb{K}}\begin{cases}\mathcal{B}[\![P[\![s]\!]_k]\!](\pi_k(\bar{b})) & k\vDash\varphi\\ \mathcal{B}[\![\texttt{skip}]\!](\pi_k(\bar{b})) & k\nvDash\varphi\end{cases} \quad \text{(by def. of } P)$$

$$= \lambda\bar{b}.\prod_{k\in\mathbb{K}}\begin{cases}\mathcal{B}[\![P[\![s]\!]_k]\!](\pi_k(\bar{b})) & k\vDash\varphi\\ \pi_k(\bar{b}) & k\nvDash\varphi\end{cases} \quad \text{(by def. of } \mathcal{B})$$

$$= \lambda\bar{b}.\prod_{k\in\mathbb{K}}\begin{cases}\pi_k(\overline{\mathcal{B}}[\![s]\!]\bar{b}) & k\vDash\varphi\\ \pi_k(\bar{b}) & k\nvDash\varphi\end{cases} \quad \text{(by IH)}$$

$$= \overline{\mathcal{B}}[\![\texttt{\#if}\ \varphi\ s]\!] \quad \text{(by def. of } \overline{\mathcal{B}})$$

□

J. Soundness of family-based approximate semantics

Lemma 27 (Soundness of family-based approximate expression semantics)**.**

$$\forall e, \overline{b}.\ (\overline{\mathcal{C}'}[\![e]\!] \circ \mathit{lift}(\gamma_{\text{BC}}))\overline{b} \ddot{\sqsubseteq} \overline{\mathcal{B}'}[\![e]\!]\overline{b}$$

where

$$\begin{aligned}
\alpha_{\text{CB}}(c) &= \lambda \mathtt{x}.\ \{\sigma(\mathtt{x}) \mid \sigma \in c\} \\
\gamma_{\text{BC}}(b) &= \{\sigma \mid \forall \mathtt{x} : \sigma(\mathtt{x}) \in b(\mathtt{x})\} \\
\mathit{lift}(\alpha_{\text{CB}}) &= \lambda \overline{c}.\ \prod_{k \in \mathbb{K}} \alpha_{\text{CB}}(\pi_k(\overline{c})) \\
\mathit{lift}(\gamma_{\text{BC}}) &= \lambda \overline{b}.\ \prod_{k \in \mathbb{K}} \gamma_{\text{BC}}(\pi_k(\overline{b}))
\end{aligned}$$

Proof. Let e and $\overline{b}$ be given. We proceed by structural induction on e.

Case n:

$$\begin{aligned}
&(\overline{\mathcal{C}'}[\![n]\!] \circ \mathit{lift}(\gamma_{\text{BC}}))\overline{b} \\
&= \overline{\mathcal{C}'}[\![n]\!](\mathit{lift}(\gamma_{\text{BC}})\overline{b}) && \text{(by def. of } \circ) \\
&= (\lambda \overline{c}.\ \prod_{k \in \mathbb{K}} \{n\})(\mathit{lift}(\gamma_{\text{BC}})\overline{b}) && \text{(by def. of } \overline{\mathcal{C}'}) \\
&= \prod_{k \in \mathbb{K}} \{n\} && (\beta\text{-reduction}) \\
&= \overline{\mathcal{B}'}[\![n]\!]\overline{b} && \text{(by def. of } \overline{\mathcal{B}'})
\end{aligned}$$

Case $\mathtt{x}$:

$$\begin{aligned}
&(\overline{\mathcal{C}'}[\![\mathtt{x}]\!] \circ \mathit{lift}(\gamma_{\text{BC}}))\overline{b} \\
&= \overline{\mathcal{C}'}[\![\mathtt{x}]\!](\mathit{lift}(\gamma_{\text{BC}})\overline{b}) && \text{(by def. of } \circ) \\
&= (\lambda \overline{c}.\ \prod_{k \in \mathbb{K}} \{\sigma(\mathtt{x}) \mid \sigma \in \pi_k(\overline{c})\})(\mathit{lift}(\gamma_{\text{BC}})\overline{b}) && \text{(by def. of } \overline{\mathcal{C}'}) \\
&= \prod_{k \in \mathbb{K}} \{\sigma(\mathtt{x}) \mid \sigma \in \pi_k((\mathit{lift}(\gamma_{\text{BC}})\overline{b}))\} && (\beta\text{-reduction}) \\
&= \prod_{k \in \mathbb{K}} \{\sigma(\mathtt{x}) \mid \sigma \in \pi_k(\prod_{k' \in \mathbb{K}} \gamma_{\text{BC}}(\pi_{k'}(\overline{b})))\} \\
& && \text{(by def. of } \mathit{lift}(\gamma_{\text{BC}})) \\
&= \prod_{k \in \mathbb{K}} \{\sigma(\mathtt{x}) \mid \sigma \in \gamma_{\text{BC}}(\pi_k(\overline{b}))\} && \text{(by def. of } \pi_k) \\
&= \prod_{k \in \mathbb{K}} \{\sigma(\mathtt{x}) \mid \sigma \in \{\sigma' \mid \forall \mathtt{y} : \sigma'(\mathtt{y}) \in \pi_k(\overline{b})(\mathtt{y})\}\} \\
& && \text{(by def. of } \gamma_{\text{BC}}) \\
&= \prod_{k \in \mathbb{K}} \pi_k(\overline{b})(\mathtt{x}) && \text{(simplify)} \\
&= \overline{\mathcal{B}'}[\![\mathtt{x}]\!]\overline{b}
\end{aligned}$$

Case $e_0 \oplus e_1$:

$$\begin{aligned}
&(\overline{\mathcal{C}'}[\![e_0 \oplus e_1]\!] \circ \mathit{lift}(\gamma_{\text{BC}}))\overline{b} \\
&= \overline{\mathcal{C}'}[\![e_0 \oplus e_1]\!](\mathit{lift}(\gamma_{\text{BC}})\overline{b}) && \text{(by def. of } \circ) \\
&= (\lambda \overline{c}.\ \prod_{k \in \mathbb{K}} \{v \mid \sigma \in \pi_k(\overline{c})\ \wedge \\
&\qquad v \in (\pi_k(\overline{\mathcal{C}'}[\![e_0]\!](\prod_{k' \in \mathbb{K}} \{\sigma\}))) \mathbin{\dot{\oplus}} (\pi_k(\overline{\mathcal{C}'}[\![e_1]\!](\prod_{k' \in \mathbb{K}} \{\sigma\})))\}) \\
&\qquad (\mathit{lift}(\gamma_{\text{BC}})\overline{b}) \\
& && \text{(by def. of } \overline{\mathcal{C}'}) \\
&= \prod_{k \in \mathbb{K}} \{v \mid \sigma \in \pi_k(\mathit{lift}(\gamma_{\text{BC}})\overline{b})\ \wedge \\
&\qquad v \in (\pi_k(\overline{\mathcal{C}'}[\![e_0]\!](\prod_{k' \in \mathbb{K}} \{\sigma\}))) \mathbin{\dot{\oplus}} (\pi_k(\overline{\mathcal{C}'}[\![e_1]\!](\prod_{k' \in \mathbb{K}} \{\sigma\})))\} \\
& && (\beta\text{-reduction}) \\
&\ddot{\sqsubseteq} \prod_{k \in \mathbb{K}} \{v \mid \sigma \in \pi_k(\mathit{lift}(\gamma_{\text{BC}})\overline{b})\ \wedge \\
&\qquad v \in (\pi_k(\overline{\mathcal{C}'}[\![e_0]\!](\prod_{k' \in \mathbb{K}} \pi_k(\mathit{lift}(\gamma_{\text{BC}})\overline{b})))) \mathbin{\dot{\oplus}} (\pi_k(\overline{\mathcal{C}'}[\![e_1]\!](\prod_{k' \in \mathbb{K}} \{\sigma\})))\} \\
& && \text{(monotonicity of } \mathbin{\dot{\oplus}}, \pi_k, \overline{\mathcal{C}'}) \\
&\ddot{\sqsubseteq} \prod_{k \in \mathbb{K}} (\pi_k(\overline{\mathcal{C}'}[\![e_0]\!](\prod_{k' \in \mathbb{K}} \pi_k(\mathit{lift}(\gamma_{\text{BC}})\overline{b})))) \\
&\qquad \mathbin{\dot{\oplus}} (\pi_k(\overline{\mathcal{C}'}[\![e_1]\!](\prod_{k' \in \mathbb{K}} \pi_k(\mathit{lift}(\gamma_{\text{BC}})\overline{b})))) \\
& && \text{(monotonicity of } \mathbin{\dot{\oplus}}, \pi_k, \overline{\mathcal{C}'}) \\
&= \prod_{k \in \mathbb{K}} (\pi_k(\overline{\mathcal{C}'}[\![e_0]\!](\prod_{k' \in \mathbb{K}} \pi_{k'}(\mathit{lift}(\gamma_{\text{BC}})\overline{b})))) \\
&\qquad \mathbin{\dot{\oplus}} (\pi_k(\overline{\mathcal{C}'}[\![e_1]\!](\prod_{k' \in \mathbb{K}} \pi_{k'}(\mathit{lift}(\gamma_{\text{BC}})\overline{b})))) \\
& && \text{(identical } k \text{ entries)} \\
&= \prod_{k \in \mathbb{K}} (\pi_k(\overline{\mathcal{C}'}[\![e_0]\!](\mathit{lift}(\gamma_{\text{BC}})\overline{b}))) \mathbin{\dot{\oplus}} (\pi_k(\overline{\mathcal{C}'}[\![e_1]\!](\mathit{lift}(\gamma_{\text{BC}})\overline{b}))) \\
& && \text{(shortcut constr.+proj.)} \\
&\ddot{\sqsubseteq} \prod_{k \in \mathbb{K}} (\pi_k(\overline{\mathcal{B}'}[\![e_0]\!]\overline{b})) \mathbin{\dot{\oplus}} (\pi_k(\overline{\mathcal{C}'}[\![e_1]\!](\mathit{lift}(\gamma_{\text{BC}})\overline{b}))) \\
& && \text{(IH, monotonicity of } \mathbin{\dot{\oplus}}, \pi_k, \overline{\mathcal{C}'}) \\
&\ddot{\sqsubseteq} \prod_{k \in \mathbb{K}} (\pi_k(\overline{\mathcal{B}'}[\![e_0]\!]\overline{b})) \mathbin{\dot{\oplus}} (\pi_k(\overline{\mathcal{B}'}[\![e_1]\!]\overline{b})) \\
& && \text{(IH, monotonicity of } \mathbin{\dot{\oplus}}, \pi_k, \overline{\mathcal{C}'}) \\
&= \overline{\mathcal{B}'}[\![e_0 \oplus e_1]\!]\overline{b} && \text{(by def. of } \overline{\mathcal{B}'})
\end{aligned}$$

□

Theorem 28 (Soundness of family-based approximate statement semantics)**.**

$$\forall s, \overline{b}.\ (\mathit{lift}(\alpha_{\text{CB}}) \circ \overline{\mathcal{C}}[\![s]\!] \circ \mathit{lift}(\gamma_{\text{BC}}))\overline{b} \ddot{\sqsubseteq} \overline{\mathcal{B}}[\![s]\!]\overline{b}$$

Proof. Let s and $\overline{b}$ be given. We proceed by structural induction on s.

Case $\mathtt{skip}$:

$$\begin{aligned}
&(\mathit{lift}(\alpha_{\text{CB}}) \circ \overline{\mathcal{C}}[\![\mathtt{skip}]\!] \circ \mathit{lift}(\gamma_{\text{BC}}))\overline{b} \\
&= (\mathit{lift}(\alpha_{\text{CB}}) \circ (\lambda \overline{c}.\ \overline{c}) \circ \mathit{lift}(\gamma_{\text{BC}}))\overline{b} && \text{(by def. of } \overline{\mathcal{C}}) \\
&= (\mathit{lift}(\alpha_{\text{CB}}) \circ \mathit{lift}(\gamma_{\text{BC}}))\overline{b} && \text{(simplify)} \\
&\ddot{\sqsubseteq} \overline{b} && (\mathit{lift}(\alpha_{\text{CB}}) \circ \mathit{lift}(\gamma_{\text{BC}}) \text{ reductive}) \\
&= \overline{\mathcal{B}}[\![\mathtt{skip}]\!]\overline{b} && \text{(by def. of } \overline{\mathcal{B}})
\end{aligned}$$

Case $\texttt{x := } e$**:**

$$
\begin{aligned}
&(\mathit{lift}(\alpha_{\text{CB}}) \circ \overline{\mathcal{C}}[\![\texttt{x := } e]\!] \circ \mathit{lift}(\gamma_{\text{BC}}))\bar{b} \\
&= \mathit{lift}(\alpha_{\text{CB}})(\overline{\mathcal{C}}[\![\texttt{x := } e]\!](\mathit{lift}(\gamma_{\text{BC}})\bar{b})) && \text{(by def. of } \circ) \\
&= \mathit{lift}(\alpha_{\text{CB}})(\prod_{k\in\mathbb{K}}\{\sigma[\texttt{x}\mapsto v] \mid \sigma \in \pi_k(\mathit{lift}(\gamma_{\text{BC}})\bar{b}) \wedge \\
&\qquad\qquad v \in \pi_k(\overline{\mathcal{C}'}[\![e]\!](\prod_{k'\in\mathbb{K}}\{\sigma\}))\}) \\
&\qquad\qquad \text{(by def. of } \overline{\mathcal{B}}) \\
&\ddot{\subseteq}\ \mathit{lift}(\alpha_{\text{CB}})(\prod_{k\in\mathbb{K}}\{\sigma[\texttt{x}\mapsto v] \mid \sigma \in \pi_k(\mathit{lift}(\gamma_{\text{BC}})\bar{b}) \wedge \\
&\qquad\qquad v \in \pi_k(\overline{\mathcal{C}'}[\![e]\!](\prod_{k'\in\mathbb{K}}\pi_k(\mathit{lift}(\gamma_{\text{BC}})\bar{b}))))\}) \\
&\qquad\qquad \text{(monotonicity of } \mathit{lift}(\alpha_{\text{CB}}), \pi_k, \overline{\mathcal{C}'}) \\
&= \mathit{lift}(\alpha_{\text{CB}})(\prod_{k\in\mathbb{K}}\{\sigma[\texttt{x}\mapsto v] \mid \sigma \in \pi_k(\mathit{lift}(\gamma_{\text{BC}})\bar{b}) \wedge \\
&\qquad\qquad v \in \pi_k(\overline{\mathcal{C}'}[\![e]\!](\prod_{k'\in\mathbb{K}}\pi_{k'}(\mathit{lift}(\gamma_{\text{BC}})\bar{b}))))\}) \\
&\qquad\qquad \text{(identical } k \text{ entries)} \\
&= \mathit{lift}(\alpha_{\text{CB}})(\prod_{k\in\mathbb{K}}\{\sigma[\texttt{x}\mapsto v] \mid \sigma \in \pi_k(\mathit{lift}(\gamma_{\text{BC}})\bar{b}) \wedge \\
&\qquad\qquad v \in \pi_k(\overline{\mathcal{C}'}[\![e]\!](\mathit{lift}(\gamma_{\text{BC}})\bar{b}))\}) \\
&\qquad\qquad \text{(shortcut constr.+proj.)} \\
&\ddot{\subseteq}\ \mathit{lift}(\alpha_{\text{CB}})(\prod_{k\in\mathbb{K}}\{\sigma[\texttt{x}\mapsto v] \mid \sigma \in \pi_k(\mathit{lift}(\gamma_{\text{BC}})\bar{b}) \wedge v \in \pi_k(\overline{\mathcal{B}'}[\![e]\!]\bar{b})\}) \\
&\qquad\qquad \text{(by Lemma 27)} \\
&= \prod_{k\in\mathbb{K}}\alpha_{\text{CB}}(\pi_k(\prod_{k'\in\mathbb{K}}\{\sigma[\texttt{x}\mapsto v] \mid \sigma \in \pi_{k'}(\mathit{lift}(\gamma_{\text{BC}})\bar{b}) \\
&\qquad\qquad \wedge v \in \pi_{k'}(\overline{\mathcal{B}'}[\![e]\!]\bar{b})\})) \\
&\qquad\qquad \text{(by def. of } \mathit{lift}(\alpha_{\text{CB}})) \\
&= \prod_{k\in\mathbb{K}}\alpha_{\text{CB}}(\{\sigma[\texttt{x}\mapsto v] \mid \sigma \in \pi_k(\mathit{lift}(\gamma_{\text{BC}})\bar{b}) \wedge v \in \pi_k(\overline{\mathcal{B}'}[\![e]\!]\bar{b})\}) \\
&\qquad\qquad \text{(shortcut constr.+proj.)} \\
&= \prod_{k\in\mathbb{K}}\lambda\texttt{y}.\{\sigma'(\texttt{y}) \mid \sigma' \in \{\sigma[\texttt{x}\mapsto v] \mid \sigma \in \pi_k(\mathit{lift}(\gamma_{\text{BC}})\bar{b}) \\
&\qquad\qquad \wedge v \in \pi_k(\overline{\mathcal{B}'}[\![e]\!]\bar{b})\}\} \\
&\qquad\qquad \text{(by def. of } \alpha_{\text{CB}}) \\
&= \prod_{k\in\mathbb{K}}\lambda\texttt{y}.\{\sigma[\texttt{x}\mapsto v](\texttt{y}) \mid \sigma \in \pi_k(\mathit{lift}(\gamma_{\text{BC}})\bar{b}) \wedge v \in \pi_k(\overline{\mathcal{B}'}[\![e]\!]\bar{b})\} \\
&\qquad\qquad \text{(simplify)} \\
&= \prod_{k\in\mathbb{K}}\lambda\texttt{y}.\begin{cases}\{v \mid \sigma \in \pi_k(\mathit{lift}(\gamma_{\text{BC}})\bar{b}) \wedge v \in \pi_k(\overline{\mathcal{B}'}[\![e]\!]\bar{b})\} & \texttt{x} = \texttt{y} \\ \{\sigma(\texttt{y}) \mid \sigma \in \pi_k(\mathit{lift}(\gamma_{\text{BC}})\bar{b}) \wedge v \in \pi_k(\overline{\mathcal{B}'}[\![e]\!]\bar{b})\} & \texttt{x} \neq \texttt{y}\end{cases} \\
&\qquad\qquad \text{(by def. of } \alpha_{\text{CB}}) \\
&\ddot{\subseteq}\ \prod_{k\in\mathbb{K}}\lambda\texttt{y}.\begin{cases}\pi_k(\overline{\mathcal{B}'}[\![e]\!]\bar{b}) & \texttt{x} = \texttt{y} \\ \{\sigma(\texttt{y}) \mid \sigma \in \pi_k(\mathit{lift}(\gamma_{\text{BC}})\bar{b})\} & \texttt{x} \neq \texttt{y}\end{cases} \\
&\qquad\qquad \text{(by def. of } \ddot{\subseteq}) \\
&= \prod_{k\in\mathbb{K}}\lambda\texttt{y}.\begin{cases}\pi_k(\overline{\mathcal{B}'}[\![e]\!]\bar{b}) & \texttt{x} = \texttt{y} \\ \{\sigma(\texttt{y}) \mid \sigma \in \pi_k(\prod_{k'\in\mathbb{K}}\gamma_{\text{BC}}(\pi_{k'}(\bar{b})))\} & \texttt{x} \neq \texttt{y}\end{cases} \\
&\qquad\qquad \text{(by def. of } \mathit{lift}(\gamma_{\text{BC}})) \\
&= \prod_{k\in\mathbb{K}}\lambda\texttt{y}.\begin{cases}\pi_k(\overline{\mathcal{B}'}[\![e]\!]\bar{b}) & \texttt{x} = \texttt{y} \\ \{\sigma(\texttt{y}) \mid \sigma \in \gamma_{\text{BC}}(\pi_k(\bar{b}))\} & \texttt{x} \neq \texttt{y}\end{cases} \\
&\qquad\qquad \text{(shortcut constr.+proj.)} \\
&= \prod_{k\in\mathbb{K}}\lambda\texttt{y}.\begin{cases}\pi_k(\overline{\mathcal{B}'}[\![e]\!]\bar{b}) & \texttt{x} = \texttt{y} \\ \{\sigma(\texttt{y}) \mid \sigma \in \{\sigma' \mid \forall \texttt{z} : \sigma'(\texttt{z}) \in \pi_k(\bar{b})(\texttt{z})\}\} & \texttt{x} \neq \texttt{y}\end{cases} \\
&\qquad\qquad \text{(by def. of } \gamma_{\text{BC}}) \\
&= \prod_{k\in\mathbb{K}}\lambda\texttt{y}.\begin{cases}\pi_k(\overline{\mathcal{B}'}[\![e]\!]\bar{b}) & \texttt{x} = \texttt{y} \\ \{\sigma(\texttt{y}) \mid \forall \texttt{z} : \sigma(\texttt{z}) \in \pi_k(\bar{b})(\texttt{z})\} & \texttt{x} \neq \texttt{y}\end{cases} \\
&\qquad\qquad \text{(simplify)} \\
&= \prod_{k\in\mathbb{K}}\lambda\texttt{y}.\begin{cases}\pi_k(\overline{\mathcal{B}'}[\![e]\!]\bar{b}) & \texttt{x} = \texttt{y} \\ \pi_k(\bar{b})(\texttt{y}) & \texttt{x} \neq \texttt{y}\end{cases} \\
&\qquad\qquad \text{(simplify)} \\
&= \prod_{k\in\mathbb{K}}(\pi_k(\bar{b}))[\texttt{x}\mapsto \pi_k(\overline{\mathcal{B}'}[\![e]\!]\bar{b})] && \text{(by def. of } [\mapsto]) \\
&= \overline{\mathcal{B}}[\![\texttt{x := } e]\!]\bar{b} && \text{(by def. of } \overline{\mathcal{B}})
\end{aligned}
$$

Case $s_0\ \texttt{;}\ s_1$**:**

$$
\begin{aligned}
&(\mathit{lift}(\alpha_{\text{CB}}) \circ \overline{\mathcal{C}}[\![s_0\ \texttt{;}\ s_1]\!] \circ \mathit{lift}(\gamma_{\text{BC}}))\bar{b} \\
&= (\mathit{lift}(\alpha_{\text{CB}}) \circ \overline{\mathcal{C}}[\![s_1]\!] \circ \overline{\mathcal{C}}[\![s_0]\!] \circ \mathit{lift}(\gamma_{\text{BC}}))\bar{b} && \text{(by def. of } \overline{\mathcal{C}}) \\
&\ddot{\subseteq}\ (\mathit{lift}(\alpha_{\text{CB}}) \circ \overline{\mathcal{C}}[\![s_1]\!] \circ \mathit{lift}(\gamma_{\text{BC}}) \circ \mathit{lift}(\alpha_{\text{CB}}) \circ \overline{\mathcal{C}}[\![s_0]\!] \circ \mathit{lift}(\gamma_{\text{BC}}))\bar{b} \\
&\qquad\qquad (\mathit{lift}(\gamma_{\text{BC}}) \circ \mathit{lift}(\alpha_{\text{CB}}) \text{ extensive}) \\
&\ddot{\subseteq}\ (\overline{\mathcal{B}}[\![s_1]\!] \circ \overline{\mathcal{B}}[\![s_0]\!])\bar{b} && \text{(by IH, twice)} \\
&= \overline{\mathcal{B}}[\![s_0\ \texttt{;}\ s_1]\!]\bar{b} && \text{(by def. of } \overline{\mathcal{B}})
\end{aligned}
$$

Case $\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1$**:**

$$
\begin{aligned}
&(\mathit{lift}(\alpha_{\text{CB}}) \circ \overline{\mathcal{C}}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!] \circ \mathit{lift}(\gamma_{\text{BC}}))\bar{b} \\
&= \mathit{lift}(\alpha_{\text{CB}})(\overline{\mathcal{C}}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!](\mathit{lift}(\gamma_{\text{BC}})\bar{b})) \\
&\qquad\qquad \text{(by def. of } \circ) \\
&= \mathit{lift}(\alpha_{\text{CB}}) \\
&\quad (\prod_{k\in\mathbb{K}}\pi_k(\overline{\mathcal{C}}[\![s_0]\!](\prod_{k''\in\mathbb{K}}\{\sigma \in \pi_k(\mathit{lift}(\gamma_{\text{BC}})\bar{b}) \mid 0 \notin \pi_k(\overline{\mathcal{C}'}[\![e]\!](\prod_{k'\in\mathbb{K}}\{\sigma\}))\})) \\
&\quad \cup \pi_k(\overline{\mathcal{C}}[\![s_1]\!](\prod_{k''\in\mathbb{K}}\{\sigma \in \pi_k(\mathit{lift}(\gamma_{\text{BC}})\bar{b}) \mid 0 \in \pi_k(\mathcal{C}'[\![e]\!](\prod_{k'\in\mathbb{K}}\{\sigma\}))\}))) \\
&\qquad\qquad \text{(by def. of } \overline{\mathcal{C}}) \\
&\ddot{\subseteq}\ \mathit{lift}(\alpha_{\text{CB}})(\prod_{k\in\mathbb{K}}\pi_k(\overline{\mathcal{C}}[\![s_0]\!](\prod_{k''\in\mathbb{K}}\pi_k(\mathit{lift}(\gamma_{\text{BC}})\bar{b}))) \\
&\qquad\qquad \cup \pi_k(\overline{\mathcal{C}}[\![s_1]\!](\prod_{k''\in\mathbb{K}}\pi_k(\mathit{lift}(\gamma_{\text{BC}})\bar{b})))) \\
&\qquad\qquad \text{(monotonicity of } \mathit{lift}(\alpha_{\text{CB}}), \pi_k, \overline{\mathcal{C}}) \\
&= \mathit{lift}(\alpha_{\text{CB}})(\prod_{k\in\mathbb{K}}\pi_k(\overline{\mathcal{C}}[\![s_0]\!](\prod_{k''\in\mathbb{K}}\pi_{k''}(\mathit{lift}(\gamma_{\text{BC}})\bar{b}))) \\
&\qquad\qquad \cup \pi_k(\overline{\mathcal{C}}[\![s_1]\!](\prod_{k''\in\mathbb{K}}\pi_{k''}(\mathit{lift}(\gamma_{\text{BC}})\bar{b})))) \\
&\qquad\qquad \text{(identical } k \text{ entries)} \\
&= \mathit{lift}(\alpha_{\text{CB}})(\prod_{k\in\mathbb{K}}\pi_k(\overline{\mathcal{C}}[\![s_0]\!](\mathit{lift}(\gamma_{\text{BC}})\bar{b})) \cup \pi_k(\overline{\mathcal{C}}[\![s_1]\!](\mathit{lift}(\gamma_{\text{BC}})\bar{b}))) \\
&\qquad\qquad \text{(simplify)} \\
&= \mathit{lift}(\alpha_{\text{CB}})(\prod_{k\in\mathbb{K}}\pi_k(\overline{\mathcal{C}}[\![s_0]\!](\mathit{lift}(\gamma_{\text{BC}})\bar{b})) \dot{\cup} \prod_{k\in\mathbb{K}}\pi_k(\overline{\mathcal{C}}[\![s_1]\!](\mathit{lift}(\gamma_{\text{BC}})\bar{b}))) \\
&\qquad\qquad \text{(by def. of } \dot{\cup}) \\
&= \mathit{lift}(\alpha_{\text{CB}})(\overline{\mathcal{C}}[\![s_0]\!](\mathit{lift}(\gamma_{\text{BC}})\bar{b}) \dot{\cup} \overline{\mathcal{C}}[\![s_1]\!](\mathit{lift}(\gamma_{\text{BC}})\bar{b})) \\
&\qquad\qquad \text{(simplify)} \\
&= \mathit{lift}(\alpha_{\text{CB}})(\overline{\mathcal{C}}[\![s_0]\!](\mathit{lift}(\gamma_{\text{BC}})\bar{b})) \ddot{\cup} \mathit{lift}(\alpha_{\text{CB}})(\overline{\mathcal{C}}[\![s_1]\!](\mathit{lift}(\gamma_{\text{BC}})\bar{b})) \\
&\qquad\qquad (\mathit{lift}(\alpha_{\text{CB}}) \text{ a CJM}) \\
&\ddot{\subseteq}\ \overline{\mathcal{B}}[\![s_0]\!]\bar{b} \ddot{\cup} \overline{\mathcal{B}}[\![s_1]\!]\bar{b} && \text{(by IH, twice)} \\
&= \overline{\mathcal{B}}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!]\bar{b} && \text{(by def. of } \overline{\mathcal{B}})
\end{aligned}
$$

Case `while` e `do` s**:** In this case our higher-order Galois connection reads:

$$\alpha_{\to}(\overline{\Phi}) = \mathit{lift}(\alpha_{\text{CB}}) \circ \overline{\Phi} \circ \mathit{lift}(\gamma_{\text{BC}})$$
$$\gamma_{\to}(\overline{\Phi}) = \mathit{lift}(\gamma_{\text{BC}}) \circ \overline{\Phi} \circ \mathit{lift}(\alpha_{\text{CB}})$$

First observe that for any given monotone $\overline{\Phi}$

$$\begin{aligned}
&(\alpha_{\to} \circ (\lambda\overline{\Phi}.\, \lambda\overline{c}.\, \prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}])\} \\
&\quad \dot\cup\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}])\})))) \\
&\quad \circ \gamma_{\to})\overline{\Phi} \\
&= \alpha_{\to}((\lambda\overline{\Phi}.\, \lambda\overline{c}.\, \prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}])\} \\
&\quad \dot\cup\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}])\})))) \\
&\quad (\gamma_{\to}(\overline{\Phi}))) && \text{(by def. of } \circ) \\
&= \alpha_{\to}(\lambda\overline{c}.\, \prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}])\} \\
&\dot\cup\, \gamma_{\to}(\overline{\Phi})(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}])\}))) \\
&&& (\beta\text{-reduction}) \\
&\dddot\subseteq \alpha_{\to}(\lambda\overline{c}.\, \prod_{k'' \in \mathbb{K}} \pi_{k''}(\overline{c}) \\
&\dot\cup\, \gamma_{\to}(\overline{\Phi})(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}])\})))) \\
&&& \text{(monotonicity of } \alpha_{\to}) \\
&\dddot\subseteq \alpha_{\to}(\lambda\overline{c}.\, \prod_{k'' \in \mathbb{K}} \pi_{k''}(\overline{c}) \,\dot\cup\, \gamma_{\to}(\overline{\Phi})(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \pi_{k''}(\overline{c})))) \\
&&& \text{(monotonicity of } \alpha_{\to}, \gamma_{\to}, \overline{\Phi}, \overline{\mathcal{C}}) \\
&= \alpha_{\to}(\lambda\overline{c}.\, \overline{c} \,\dot\cup\, \gamma_{\to}(\overline{\Phi})(\overline{\mathcal{C}}[\![s]\!](\overline{c}))) \\
&&& \text{(shortcut constr.+proj., twice)} \\
&= \mathit{lift}(\alpha_{\text{CB}}) \circ (\lambda\overline{c}.\, \overline{c} \,\dot\cup\, \gamma_{\to}(\overline{\Phi})(\overline{\mathcal{C}}[\![s]\!](\overline{c}))) \circ \mathit{lift}(\gamma_{\text{BC}}) \\
&&& \text{(by def. of } \alpha_{\to}) \\
&= \lambda\overline{b}.\, (\mathit{lift}(\alpha_{\text{CB}}) \circ (\lambda\overline{c}.\, \overline{c} \,\dot\cup\, \gamma_{\to}(\overline{\Phi})(\overline{\mathcal{C}}[\![s]\!](\overline{c}))) \circ \mathit{lift}(\gamma_{\text{BC}}))\overline{b} \\
&&& (\eta\text{-expansion}) \\
&= \lambda\overline{b}.\, \mathit{lift}(\alpha_{\text{CB}})((\lambda\overline{c}.\, \overline{c} \,\dot\cup\, \gamma_{\to}(\overline{\Phi})(\overline{\mathcal{C}}[\![s]\!](\overline{c})))(\mathit{lift}(\gamma_{\text{BC}})\overline{b})) \\
&&& \text{(by def. of } \circ) \\
&= \lambda\overline{b}.\, \mathit{lift}(\alpha_{\text{CB}})(\mathit{lift}(\gamma_{\text{BC}})\overline{b} \,\dot\cup\, \gamma_{\to}(\overline{\Phi})(\overline{\mathcal{C}}[\![s]\!](\mathit{lift}(\gamma_{\text{BC}})\overline{b}))) \\
&&& (\beta\text{-reduction}) \\
&= \lambda\overline{b}.\, \mathit{lift}(\alpha_{\text{CB}})(\mathit{lift}(\gamma_{\text{BC}})\overline{b}) \,\ddot\cup\, \mathit{lift}(\alpha_{\text{CB}})(\gamma_{\to}(\overline{\Phi})(\overline{\mathcal{C}}[\![s]\!](\mathit{lift}(\gamma_{\text{BC}})\overline{b}))) \\
&&& (\mathit{lift}(\alpha_{\text{CB}}) \text{ a CJM}) \\
&\dddot\subseteq \lambda\overline{b}.\, \overline{b} \,\ddot\cup\, \mathit{lift}(\alpha_{\text{CB}})(\gamma_{\to}(\overline{\Phi})(\overline{\mathcal{C}}[\![s]\!](\mathit{lift}(\gamma_{\text{BC}})\overline{b}))) \\
&&& (\mathit{lift}(\alpha_{\text{CB}}) \circ \mathit{lift}(\gamma_{\text{BC}}) \text{ reductive}) \\
&= \lambda\overline{b}.\, \overline{b} \,\ddot\cup\, \mathit{lift}(\alpha_{\text{CB}})((\mathit{lift}(\gamma_{\text{BC}}) \circ \overline{\Phi} \circ \mathit{lift}(\alpha_{\text{CB}}))(\overline{\mathcal{C}}[\![s]\!](\mathit{lift}(\gamma_{\text{BC}})\overline{b}))) \\
&&& \text{(by def. of } \gamma_{\to}) \\
&\ddot\subseteq \lambda\overline{b}.\, \overline{b} \,\ddot\cup\, (\overline{\Phi} \circ \mathit{lift}(\alpha_{\text{CB}}))(\overline{\mathcal{C}}[\![s]\!](\mathit{lift}(\gamma_{\text{BC}})\overline{b})) \\
&&& (\mathit{lift}(\alpha_{\text{CB}}) \circ \mathit{lift}(\gamma_{\text{BC}}) \text{ reductive}) \\
&\ddot\subseteq \lambda\overline{b}.\, \overline{b} \,\ddot\cup\, \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\overline{b}) && \text{(IH, } \overline{\Phi} \text{ monotone)} \\
&= (\lambda\overline{\Phi}.\, \lambda\overline{b}.\, \overline{b} \,\ddot\cup\, \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\overline{b}))\overline{\Phi} && (\eta\text{-expand})
\end{aligned}$$

We can now utilize that observation in the following calculation.

$$\begin{aligned}
&(\mathit{lift}(\alpha_{\text{CB}}) \circ \overline{\mathcal{C}}[\![\texttt{while } e \texttt{ do } s]\!] \circ \mathit{lift}(\gamma_{\text{BC}}))\overline{b} \\
&= \alpha_{\to}(\overline{\mathcal{C}}[\![\texttt{while } e \texttt{ do } s]\!])\overline{b} && \text{(by def. of } \alpha_{\to}) \\
&= \alpha_{\to} \\
&\quad (\text{lfp}(\lambda\overline{\Phi}.\, \lambda\overline{c}.\, \prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}])\} \\
&\quad \dot\cup\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}])\}))))) \\
&\quad \overline{b} && \text{(by def. of } \overline{\mathcal{C}}) \\
&\ddot\subseteq (\text{lfp}\, \alpha_{\to} \circ \\
&\quad (\lambda\overline{\Phi}.\, \lambda\overline{c}.\, \prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}])\} \\
&\quad \dot\cup\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k'' \in \mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}])\})))) \\
&\quad \circ \gamma_{\to})\overline{b} && \text{(by fixed point transfer theorem)} \\
&\ddot\subseteq (\text{lfp}\, \lambda\overline{\Phi}.\, \lambda\overline{b}.\, \overline{b} \,\ddot\cup\, \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\overline{b}))\overline{b} \\
&&& \text{(by fixed point transfer theorem + above)} \\
&= \overline{\mathcal{B}}[\![\texttt{while } e \texttt{ do } e]\!]\overline{b} && \text{(by def. of } \overline{\mathcal{B}})
\end{aligned}$$

Case `#if` φ s**:**

$$\begin{aligned}
&(\mathit{lift}(\alpha_{\text{CB}}) \circ \overline{\mathcal{C}}[\![\texttt{\#if } \varphi\ s]\!] \circ \mathit{lift}(\gamma_{\text{BC}}))\overline{b} \\
&= \mathit{lift}(\alpha_{\text{CB}})(\overline{\mathcal{C}}[\![\texttt{\#if } \varphi\ s]\!](\mathit{lift}(\gamma_{\text{BC}})\overline{b})) && \text{(by def. of } \circ) \\
&= \mathit{lift}(\alpha_{\text{CB}})(\prod_{k \in \mathbb{K}} \begin{cases} \pi_k(\overline{\mathcal{C}}[\![s]\!](\mathit{lift}(\gamma_{\text{BC}})\overline{b})) & k \vDash \varphi \\ \pi_k(\mathit{lift}(\gamma_{\text{BC}})\overline{b}) & k \nvDash \varphi \end{cases}) \\
&&& \text{(by def. of } \overline{\mathcal{C}}) \\
&= \mathit{lift}(\alpha_{\text{CB}})(\prod_{k \in \mathbb{K}} \begin{cases} \pi_k(\overline{\mathcal{C}}[\![s]\!](\mathit{lift}(\gamma_{\text{BC}})\overline{b})) & k \vDash \varphi \\ \pi_k(\prod_{k' \in \mathbb{K}} \gamma_{\text{BC}}(\pi_{k'}(\overline{b}))) & k \nvDash \varphi \end{cases}) \\
&&& \text{(by def. of } \mathit{lift}(\gamma_{\text{BC}})) \\
&= \mathit{lift}(\alpha_{\text{CB}})(\prod_{k \in \mathbb{K}} \begin{cases} \pi_k(\overline{\mathcal{C}}[\![s]\!](\mathit{lift}(\gamma_{\text{BC}})\overline{b})) & k \vDash \varphi \\ \gamma_{\text{BC}}(\pi_k(\overline{b})) & k \nvDash \varphi \end{cases}) \\
&&& \text{(shortcut constr.+proj.)} \\
&= \prod_{k \in \mathbb{K}} \begin{cases} \alpha_{\text{CB}}(\pi_k(\overline{\mathcal{C}}[\![s]\!](\mathit{lift}(\gamma_{\text{BC}})\overline{b}))) & k \vDash \varphi \\ \alpha_{\text{CB}}(\gamma_{\text{BC}}(\pi_k(\overline{b}))) & k \nvDash \varphi \end{cases} \\
&&& \text{(by def. of } \mathit{lift}(\alpha_{\text{CB}})) \\
&\dddot\subseteq \prod_{k \in \mathbb{K}} \begin{cases} \alpha_{\text{CB}}(\pi_k(\overline{\mathcal{C}}[\![s]\!](\mathit{lift}(\gamma_{\text{BC}})\overline{b}))) & k \vDash \varphi \\ \pi_k(\overline{b}) & k \nvDash \varphi \end{cases} \\
&&& (\alpha_{\text{CB}} \circ \gamma_{\text{BC}} \text{ reductive}) \\
&\dddot\subseteq \prod_{k \in \mathbb{K}} \begin{cases} \alpha_{\text{CB}}(\pi_k(\mathit{lift}(\gamma_{\text{BC}}) \circ \mathit{lift}(\alpha_{\text{CB}}) \circ \overline{\mathcal{C}}[\![s]\!](\mathit{lift}(\gamma_{\text{BC}})\overline{b}))) & k \vDash \varphi \\ \pi_k(\overline{b}) & k \nvDash \varphi \end{cases} \\
&&& (\mathit{lift}(\gamma_{\text{BC}}) \circ \mathit{lift}(\alpha_{\text{CB}}) \text{ extensive}) \\
&\dddot\subseteq \prod_{k \in \mathbb{K}} \begin{cases} \alpha_{\text{CB}}(\pi_k(\mathit{lift}(\gamma_{\text{BC}})(\overline{\mathcal{B}}[\![s]\!]\overline{b}))) & k \vDash \varphi \\ \pi_k(\overline{b}) & k \nvDash \varphi \end{cases} \\
&&& \text{(by IH, } \alpha_{\text{CB}}, \mathit{lift}(\gamma_{\text{BC}}) \text{ monotone)} \\
&= \prod_{k \in \mathbb{K}} \begin{cases} \alpha_{\text{CB}}(\pi_k(\prod_{k' \in \mathbb{K}} \gamma_{\text{BC}}(\pi_{k'}(\overline{\mathcal{B}}[\![s]\!]\overline{b})))) & k \vDash \varphi \\ \pi_k(\overline{b}) & k \nvDash \varphi \end{cases} \\
&&& \text{(by def. of } \mathit{lift}(\gamma_{\text{BC}}))
\end{aligned}$$

$$= \prod_{k \in \mathbb{K}} \begin{cases} \alpha_{\text{CB}}(\gamma_{\text{BC}}(\pi_k(\overline{\mathcal{B}}[\![s]\!]\bar{b}))) & k \vDash \varphi \\ \pi_k(\bar{b}) & k \nvDash \varphi \end{cases}$$
(shortcut constr.+proj.)

$$\ddot{\subseteq} \prod_{k \in \mathbb{K}} \begin{cases} \pi_k(\overline{\mathcal{B}}[\![s]\!]\bar{b}) & k \vDash \varphi \\ \pi_k(\bar{b}) & k \nvDash \varphi \end{cases}$$
($\alpha_{\text{CB}} \circ \gamma_{\text{BC}}$ reductive)

$= \overline{\mathcal{B}}[\![\texttt{\#if}\ \varphi\ s]\!]\bar{b}$ (by def. of $\overline{\mathcal{B}}$)

□

K. Lifting of the constant propagation analyses

Lemma 29 (Lifting of expression analysis)**.**

$$\forall e \in \mathit{Exp}.\ \overline{\mathcal{A}'}[\![e]\!] = \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}'[\![e]\!](\pi_k(\overline{a}))$$

Proof. Let e be given. We proceed by structural induction on e.

Case n**:**

$$\begin{aligned}
&\lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}'[\![n]\!](\pi_k(\overline{a})) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} (\lambda a.\, \mathtt{n})(\pi_k(\overline{a})) && \text{(by def. of } \mathcal{A}') \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathtt{n} && (\beta\text{-reduction}) \\
&= \overline{\mathcal{A}'}[\![n]\!] && \text{(by def. of } \overline{\mathcal{A}'})
\end{aligned}$$

Case `x`**:**

$$\begin{aligned}
&\lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}'[\![\mathtt{x}]\!](\pi_k(\overline{a})) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} (\lambda a.\, a(\mathtt{x}))(\pi_k(\overline{a})) && \text{(by def. of } \mathcal{A}') \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \pi_k(\overline{a})(\mathtt{x}) && (\beta\text{-reduction}) \\
&= \overline{\mathcal{A}'}[\![\mathtt{x}]\!] && \text{(by def. of } \overline{\mathcal{A}'})
\end{aligned}$$

Case $e_0 \oplus e_1$**:**

$$\begin{aligned}
&\lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}'[\![e_0 \oplus e_1]\!](\pi_k(\overline{a})) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} (\lambda a.\, \mathcal{A}'[\![e_0]\!]a \mathbin{\hat{\oplus}} \mathcal{A}'[\![e_1]\!]a)(\pi_k(\overline{a})) \\
& && \text{(by def. of } \mathcal{A}') \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}'[\![e_0]\!](\pi_k(\overline{a})) \mathbin{\hat{\oplus}} \mathcal{A}'[\![e_1]\!](\pi_k(\overline{a})) && (\beta\text{-reduction}) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \pi_k(\overline{\mathcal{A}'}[\![e_0]\!]\overline{a}) \mathbin{\hat{\oplus}} \pi_k(\overline{\mathcal{A}'}[\![e_1]\!]\overline{a}) && \text{(by IH, twice)} \\
&= \overline{\mathcal{A}'}[\![e_0 \oplus e_1]\!] && \text{(by def. of } \overline{\mathcal{A}'})
\end{aligned}$$

□

K.1 Proof of Theorem 6

$$\forall s \in \overline{\mathit{Stm}}.\ \overline{\mathcal{A}}[\![s]\!] = \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}[\![P[\![s]\!]_k]\!](\pi_k(\overline{a}))$$

Proof. Let s be given. We proceed by structural induction on s.

Case `skip`**:**

$$\begin{aligned}
&\lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}[\![P[\![\mathtt{skip}]\!]_k]\!](\pi_k(\overline{a})) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}[\![\mathtt{skip}]\!](\pi_k(\overline{a})) && \text{(by def. of } P) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} (\lambda a.\, a)(\pi_k(\overline{a})) && \text{(by def. of } \mathcal{A}) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \pi_k(\overline{a}) && (\beta\text{-reduction}) \\
&= \lambda\overline{a}.\ \overline{a} && \text{(shortcut proj. + constr.)} \\
&= \overline{\mathcal{A}}[\![\mathtt{skip}]\!] && \text{(by def. of } \overline{\mathcal{A}})
\end{aligned}$$

Case `x := e`**:**

$$\begin{aligned}
&\lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}[\![P[\![\mathtt{x} := e]\!]_k]\!](\pi_k(\overline{a})) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}[\![\mathtt{x} := e]\!](\pi_k(\overline{a})) && \text{(by def. of } P) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} (\lambda a.\, a[\mathtt{x} \mapsto \mathcal{A}'[\![e]\!]a])(\pi_k(\overline{a})) && \text{(by def. of } \mathcal{A}) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} (\pi_k(\overline{a}))[\mathtt{x} \mapsto \mathcal{A}'[\![e]\!](\pi_k(\overline{a}))] && (\beta\text{-reduction}) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} (\pi_k(\overline{a}))[\mathtt{x} \mapsto \pi_k(\overline{\mathcal{A}'}[\![e]\!]\overline{a})] && \text{(by Lemma 29)} \\
&= \overline{\mathcal{A}}[\![\mathtt{x} := e]\!] && \text{(by def. of } \overline{\mathcal{A}})
\end{aligned}$$

Case s_0 `;` s_1**:**

$$\begin{aligned}
&\lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}[\![P[\![s_0\ \mathtt{;}\ s_1]\!]_k]\!](\pi_k(\overline{a})) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}[\![P[\![s_0]\!]_k\ \mathtt{;}\ P[\![s_1]\!]_k]\!](\pi_k(\overline{a})) && \text{(by def. of } P) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} (\mathcal{A}[\![P[\![s_1]\!]_k]\!] \circ \mathcal{A}[\![P[\![s_0]\!]_k]\!])(\pi_k(\overline{a})) \\
& && \text{(by def. of } \mathcal{A}) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}[\![P[\![s_1]\!]_k]\!](\mathcal{A}[\![P[\![s_0]\!]_k]\!](\pi_k(\overline{a}))) \\
& && \text{(by def. of } \circ) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}[\![P[\![s_1]\!]_k]\!](\pi_k(\overline{\mathcal{A}}[\![s_0]\!]\overline{a})) && \text{(by IH)} \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \pi_k(\overline{\mathcal{A}}[\![s_1]\!](\overline{\mathcal{A}}[\![s_0]\!]\overline{a})) && \text{(by IH)} \\
&= \lambda\overline{a}.\ \overline{\mathcal{A}}[\![s_1]\!](\overline{\mathcal{A}}[\![s_0]\!]\overline{a}) && \text{(shortcut proj. + constr.)} \\
&= \lambda\overline{a}.\ (\overline{\mathcal{A}}[\![s_1]\!] \circ \overline{\mathcal{A}}[\![s_0]\!])\overline{a} && \text{(by def. of } \circ) \\
&= \overline{\mathcal{A}}[\![s_1]\!] \circ \overline{\mathcal{A}}[\![s_0]\!] && (\eta\text{-reduce}) \\
&= \overline{\mathcal{A}}[\![s_0\ \mathtt{;}\ s_1]\!] && \text{(by def. of } \overline{\mathcal{A}})
\end{aligned}$$

Case `if` e `then` s_0 `else` s_1**:**

$$\begin{aligned}
&\lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}[\![P[\![\mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1]\!]_k]\!](\pi_k(\overline{a})) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} \mathcal{A}[\![\mathtt{if}\ e\ \mathtt{then}\ P[\![s_0]\!]_k\ \mathtt{else}\ P[\![s_1]\!]_k]\!](\pi_k(\overline{a})) \\
& && \text{(by def. of } P) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} (\mathcal{A}[\![P[\![s_0]\!]_k]\!] \mathbin{\ddot{\sqcup}} \mathcal{A}[\![P[\![s_1]\!]_k]\!])(\pi_k(\overline{a})) \\
& && \text{(by def. of } \mathcal{A}) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} (\mathcal{A}[\![P[\![s_0]\!]_k]\!](\pi_k(\overline{a}))) \mathbin{\dot{\sqcup}} (\mathcal{A}[\![P[\![s_1]\!]_k]\!](\pi_k(\overline{a}))) \\
& && \text{(by def. of } \ddot{\sqcup}) \\
&= \lambda\overline{a}.\ \prod_{k\in\mathbb{K}} (\pi_k(\overline{\mathcal{A}}[\![s_0]\!]\overline{a})) \mathbin{\dot{\sqcup}} (\pi_k(\overline{\mathcal{A}}[\![s_1]\!]\overline{a})) && \text{(by IH, twice)} \\
&= \lambda\overline{a}.\ \overline{\mathcal{A}}[\![s_0]\!]\overline{a} \mathbin{\ddot{\sqcup}} \overline{\mathcal{A}}[\![s_1]\!]\overline{a} && \text{(by def. of } \ddot{\sqcup}) \\
&= \overline{\mathcal{A}}[\![s_0]\!] \mathbin{\ddot{\sqcup}} \overline{\mathcal{A}}[\![s_1]\!] && (\eta\text{-reduce}) \\
&= \overline{\mathcal{A}}[\![\mathtt{if}\ e\ \mathtt{then}\ s_0\ \mathtt{else}\ s_1]\!] && \text{(by def. of } \overline{\mathcal{A}})
\end{aligned}$$

Case `while` e `do` s**:** We first define α_k and γ_k as follows:

$$\begin{aligned}
&\alpha_k : \mathbb{A}^{\mathbb{K}} \to \mathbb{A} \\
&\alpha_k(\overline{a}) = \pi_k(\overline{a}) \\
&\gamma_k : \mathbb{A} \to \mathbb{A}^{\mathbb{K}} \\
&\gamma_k(a) = \prod_{k' \in \mathbb{K}} \begin{cases} a & k = k' \\ \dot{\top} & k \neq k' \end{cases}
\end{aligned}$$

Together they constitute a k-specific abstraction from the lifted analysis level to the single-program analysis level:

$$\langle \mathbb{A}^{\mathbb{K}}, \ddot{\sqsubseteq} \rangle \underset{\alpha_k}{\overset{\gamma_k}{\leftrightarrows}} \langle \mathbb{A}, \dot{\sqsubseteq} \rangle$$

Projections such as this are well-known to be Galois connections. In particular it is a Galois insertion:

$$\begin{aligned}
&(\alpha_k \circ \gamma_k)(a) \\
&= \pi_k(\prod_{k' \in \mathbb{K}} \begin{cases} a & k = k' \\ \dot{\top} & k \neq k' \end{cases}) && \text{(by def. of } \alpha_k, \gamma_k) \\
&= a && (\beta\text{-reduction})
\end{aligned}$$

We now lift this Galois connection to monotone transfer functions by a higher-order Galois connection:

$$\begin{aligned}
&\alpha_{\to} : (\mathbb{A}^{\mathbb{K}} \xrightarrow{m} \mathbb{A}^{\mathbb{K}}) \to \mathbb{A} \xrightarrow{m} \mathbb{A} \\
&\alpha_{\to}(\overline{\Phi}) = \alpha_k \circ \overline{\Phi} \circ \gamma_k \\
&\gamma_{\to} : (\mathbb{A} \xrightarrow{m} \mathbb{A}) \to \mathbb{A}^{\mathbb{K}} \xrightarrow{m} \mathbb{A}^{\mathbb{K}} \\
&\gamma_{\to}(\Phi) = \gamma_k \circ \Phi \circ \alpha_k
\end{aligned}$$

$$\langle \mathbb{A}^{\mathbb{K}} \xrightarrow{m} \mathbb{A}^{\mathbb{K}}, \ddot{\sqsubseteq} \rangle \underset{\alpha_{\to}}{\overset{\gamma_{\to}}{\leftrightarrows}} \langle \mathbb{A} \xrightarrow{m} \mathbb{A}, \ddot{\sqsubseteq} \rangle$$

Now observe that α_k induces a complete abstraction over the body of the loop:

$$\begin{aligned}
&\alpha_{\to} \circ (\lambda\overline{\Phi}.\, \lambda\overline{a}.\, \overline{a} \ddot{\sqcup} \overline{\Phi}(\overline{\mathcal{A}}[\![s]\!]\overline{a})) \\
&= \lambda\overline{\Phi}.\, \alpha_{\to}(\lambda\overline{a}.\, \overline{a} \ddot{\sqcup} \overline{\Phi}(\overline{\mathcal{A}}[\![s]\!]\overline{a})) && \text{(by def. of } \circ) \\
&= \lambda\overline{\Phi}.\, \alpha_k \circ (\lambda\overline{a}.\, \overline{a} \ddot{\sqcup} \overline{\Phi}(\overline{\mathcal{A}}[\![s]\!]\overline{a})) \circ \gamma_k && \text{(by def. of } \alpha_{\to}) \\
&= \lambda\overline{\Phi}.\, \alpha_k \circ (\lambda\overline{a}.\, \overline{a} \ddot{\sqcup} \overline{\Phi}(\overline{\mathcal{A}}[\![s]\!]\overline{a})) \circ (\lambda a.\, \gamma_k(a)) \\
& && (\eta\text{-expansion}) \\
&= \lambda\overline{\Phi}.\, \lambda a.\, (\alpha_k \circ (\lambda\overline{a}.\, \overline{a} \ddot{\sqcup} \overline{\Phi}(\overline{\mathcal{A}}[\![s]\!]\overline{a})))(\gamma_k(a)) \\
& && \text{(by def. of } \circ) \\
&= \lambda\overline{\Phi}.\, \lambda a.\, (\lambda\overline{a}.\, \alpha_k(\overline{a} \ddot{\sqcup} \overline{\Phi}(\overline{\mathcal{A}}[\![s]\!]\overline{a})))(\gamma_k(a)) && \text{(by def. of } \circ) \\
&= \lambda\overline{\Phi}.\, \lambda a.\, \alpha_k(\gamma_k(a) \ddot{\sqcup} \overline{\Phi}(\overline{\mathcal{A}}[\![s]\!](\gamma_k(a)))) && (\beta\text{-reduction}) \\
&= \lambda\overline{\Phi}.\, \lambda a.\, \alpha_k(\gamma_k(a)) \dot{\sqcup} \alpha_k(\overline{\Phi}(\overline{\mathcal{A}}[\![s]\!](\gamma_k(a)))) \\
& && (\alpha_k \text{ a CJM}) \\
&= \lambda\overline{\Phi}.\, \lambda a.\, a \dot{\sqcup} \alpha_k(\overline{\Phi}(\overline{\mathcal{A}}[\![s]\!](\gamma_k(a)))) && \text{(Galois insertion)} \\
&= \lambda\overline{\Phi}.\, \lambda a.\, a \dot{\sqcup} \alpha_k(\overline{\Phi}(\prod_{k' \in \mathbb{K}} \mathcal{A}[\![P[\![s]\!]_{k'}]\!](\pi_{k'}(\gamma_k(a))))) \\
& && \text{(by IH)} \\
&= \lambda\overline{\Phi}.\, \lambda a.\, a \dot{\sqcup} \alpha_k(\prod_{k' \in \mathbb{K}} \pi_{k'}(\overline{\Phi})(\mathcal{A}[\![P[\![s]\!]_{k'}]\!](\pi_{k'}(\gamma_k(a))))) \\
& && \text{(by def. of appl.)} \\
&= \lambda\overline{\Phi}.\, \lambda a.\, a \dot{\sqcup} \pi_k(\overline{\Phi})(\mathcal{A}[\![P[\![s]\!]_k]\!](\pi_k(\gamma_k(a)))) \\
& && \text{(by def. of } \alpha_k) \\
&= \lambda\overline{\Phi}.\, \lambda a.\, a \dot{\sqcup} \pi_k(\overline{\Phi})(\mathcal{A}[\![P[\![s]\!]_k]\!](\alpha_k(\gamma_k(a)))) \\
& && \text{(by def. of } \alpha_k) \\
&= \lambda\overline{\Phi}.\, \lambda a.\, a \dot{\sqcup} \pi_k(\overline{\Phi})(\mathcal{A}[\![P[\![s]\!]_k]\!]a) && \text{(Galois insertion)} \\
&= \lambda\overline{\Phi}.\, \lambda a.\, a \dot{\sqcup} \pi_k(\prod_{k' \in \mathbb{K}} \pi_{k'}(\overline{\Phi}) \begin{cases} (\mathcal{A}[\![P[\![s]\!]_k]\!]a) & k = k' \\ \dot{\top} & k \neq k' \end{cases}) \\
& && \text{(identical } k \text{ entries)} \\
&= \lambda\overline{\Phi}.\, \lambda a.\, a \dot{\sqcup} \pi_k(\overline{\Phi}(\prod_{k' \in \mathbb{K}} \begin{cases} (\mathcal{A}[\![P[\![s]\!]_k]\!]a) & k = k' \\ \dot{\top} & k \neq k' \end{cases})) \\
& && \text{(by def. of appl.)} \\
&= \lambda\overline{\Phi}.\, \lambda a.\, a \dot{\sqcup} \pi_k(\overline{\Phi}(\gamma_k(\mathcal{A}[\![P[\![s]\!]_k]\!]a))) && \text{(by def. of } \gamma_k) \\
&= \lambda\overline{\Phi}.\, \lambda a.\, a \dot{\sqcup} (\pi_k \circ \overline{\Phi} \circ \gamma_k)(\mathcal{A}[\![P[\![s]\!]_k]\!]a) && \text{(by def. of } \circ) \\
&= \lambda\overline{\Phi}.\, \lambda a.\, a \dot{\sqcup} (\alpha_k \circ \overline{\Phi} \circ \gamma_k)(\mathcal{A}[\![P[\![s]\!]_k]\!]a) \\
& && \text{(by def. of } \alpha_k) \\
&= \lambda\overline{\Phi}.\, (\lambda\Phi.\, \lambda a.\, a \dot{\sqcup} \Phi(\mathcal{A}[\![P[\![s]\!]_k]\!]a))(\alpha_k \circ \overline{\Phi} \circ \gamma_k) \\
& && (\beta\text{-expansion}) \\
&= (\lambda\Phi.\, \lambda a.\, a \dot{\sqcup} \Phi(\mathcal{A}[\![P[\![s]\!]_k]\!]a)) \circ (\lambda\overline{\Phi}.\, \alpha_k \circ \overline{\Phi} \circ \gamma_k) \\
& && \text{(by def. of } \circ) \\
&= (\lambda\Phi.\, \lambda a.\, a \dot{\sqcup} \Phi(\mathcal{A}[\![P[\![s]\!]_k]\!]a)) \circ \alpha_{\to} && \text{(by def. of } \alpha_{\to})
\end{aligned}$$

As a consequence, we can now apply the stronger fixed point theorem:

$$\begin{aligned}
&\alpha_{\to}(\overline{\mathcal{A}}[\![\texttt{while } e \texttt{ do } s]\!]) \\
&= \alpha_{\to}(\mathrm{lfp}\, \lambda\overline{\Phi}.\, \lambda\overline{a}.\, \overline{a} \ddot{\sqcup} \overline{\Phi}(\overline{\mathcal{A}}[\![s]\!]\overline{a})) && \text{(by def. of } \overline{\mathcal{A}}) \\
&= \mathrm{lfp}\, \lambda\Phi.\, \lambda a.\, a \dot{\sqcup} \Phi(\mathcal{A}[\![P[\![s]\!]_k]\!]a) && \text{(by above)} \\
&= \mathcal{A}[\![\texttt{while } e \texttt{ do } P[\![s]\!]_k]\!] && \text{(by def. of } \mathcal{A}) \\
&= \mathcal{A}[\![P[\![\texttt{while } e \texttt{ do } s]\!]_k]\!] && \text{(by def. of } P)
\end{aligned}$$

Substituting equals for equals we now obtain:

$$
\begin{aligned}
&\lambda\overline{a}.\prod_{k\in\mathbb{K}}\mathcal{A}[\![P[\![\texttt{while}\ e\ \texttt{do}\ s]\!]_k]\!](\pi_k(\overline{a}))\\
&=\lambda\overline{a}.\prod_{k\in\mathbb{K}}\alpha_{\to}(\overline{\mathcal{A}}[\![\texttt{while}\ e\ \texttt{do}\ s]\!])(\pi_k(\overline{a}))\\
&\qquad\qquad\text{(by above equality)}\\
&=\lambda\overline{a}.\prod_{k\in\mathbb{K}}(\alpha_k\circ\overline{\mathcal{A}}[\![\texttt{while}\ e\ \texttt{do}\ s]\!]\circ\gamma_k)(\pi_k(\overline{a}))\\
&\qquad\qquad\text{(by def. of }\alpha_{\to}\text{)}\\
&=\lambda\overline{a}.\prod_{k\in\mathbb{K}}\alpha_k(\overline{\mathcal{A}}[\![\texttt{while}\ e\ \texttt{do}\ s]\!](\gamma_k(\pi_k(\overline{a}))))\\
&\qquad\qquad\text{(by def. of }\circ\text{)}\\
&=\lambda\overline{a}.\prod_{k\in\mathbb{K}}\alpha_k\left(\overline{\mathcal{A}}[\![\texttt{while}\ e\ \texttt{do}\ s]\!]\left(\prod_{k'\in\mathbb{K}}\begin{cases}\pi_k(\overline{a}) & k=k'\\ \dagger & k\neq k'\end{cases}\right)\right)\\
&\qquad\qquad\text{(by def. of }\gamma_k\text{)}\\
&=\lambda\overline{a}.\prod_{k\in\mathbb{K}}\alpha_k\left(\prod_{k'\in\mathbb{K}}(\pi_{k'}(\overline{\mathcal{A}}[\![\texttt{while}\ e\ \texttt{do}\ s]\!]))\left(\begin{cases}\pi_k(\overline{a}) & k=k'\\ \dagger & k\neq k'\end{cases}\right)\right)\\
&\qquad\qquad\text{(by def. of appl.)}\\
&=\lambda\overline{a}.\prod_{k\in\mathbb{K}}(\pi_k(\overline{\mathcal{A}}[\![\texttt{while}\ e\ \texttt{do}\ s]\!]))(\pi_k(\overline{a}))\quad\text{(by def. of }\alpha_k\text{)}\\
&=\lambda\overline{a}.\overline{\mathcal{A}}[\![\texttt{while}\ e\ \texttt{do}\ s]\!]\overline{a}\qquad\text{(by def. of appl.)}\\
&=\overline{\mathcal{A}}[\![\texttt{while}\ e\ \texttt{do}\ s]\!]\qquad\text{(}\eta\text{-reduce)}
\end{aligned}
$$

as we desired.

Case `#if` $\varphi\ s$**:**

$$
\begin{aligned}
&\lambda\overline{a}.\prod_{k\in\mathbb{K}}\mathcal{A}[\![P[\![\texttt{\#if}\ \varphi\ s]\!]_k]\!](\pi_k(\overline{a}))\\
&=\lambda\overline{a}.\prod_{k\in\mathbb{K}}\begin{cases}\mathcal{A}[\![P[\![s]\!]_k]\!](\pi_k(\overline{a})) & k\vDash\varphi\\ \mathcal{A}[\![\texttt{skip}]\!](\pi_k(\overline{a})) & k\nvDash\varphi\end{cases}\qquad\text{(by def. of }P\text{)}\\
&=\lambda\overline{a}.\prod_{k\in\mathbb{K}}\begin{cases}\mathcal{A}[\![P[\![s]\!]_k]\!](\pi_k(\overline{a})) & k\vDash\varphi\\ \pi_k(\overline{a}) & k\nvDash\varphi\end{cases}\qquad\text{(by def. of }\mathcal{A}\text{)}\\
&=\lambda\overline{a}.\prod_{k\in\mathbb{K}}\begin{cases}\pi_k(\overline{\mathcal{A}}[\![s]\!]\overline{a}) & k\vDash\varphi\\ \pi_k(\overline{a}) & k\nvDash\varphi\end{cases}\qquad\text{(by IH)}\\
&=\overline{\mathcal{A}}[\![\texttt{\#if}\ \varphi\ s]\!]\qquad\text{(by def. of }\overline{\mathcal{A}}\text{)}
\end{aligned}
$$

□

L. Soundness of family-based analyses

Lemma 30 (Soundness of family-based expression analysis)**.**

$$\forall e, \overline{a}.\ (\mathit{lift}(\hat{\alpha}) \circ \overline{\mathcal{B}'}[\![e]\!] \circ \mathit{lift}(\gamma_{\text{AB}}))\overline{a} \mathrel{\dot{\sqsubseteq}} \overline{\mathcal{A}'}[\![e]\!]\overline{a}$$

where

$$\begin{aligned}
\alpha_{\text{BA}}(b) &= \lambda \boldsymbol{x}.\, \hat{\alpha}(b(\boldsymbol{x})) \\
\gamma_{\text{AB}}(a) &= \lambda \boldsymbol{x}.\, \hat{\gamma}(a(\boldsymbol{x})) \\
\mathit{lift}(\hat{\alpha}) &= \lambda \overline{v}.\, \prod_{k \in \mathbb{K}} \hat{\alpha}(\pi_k(\overline{v})) \\
\mathit{lift}(\hat{\gamma}) &= \lambda \overline{v}.\, \prod_{k \in \mathbb{K}} \hat{\gamma}(\pi_k(\overline{v})) \\
\mathit{lift}(\alpha_{\text{BA}}) &= \lambda \overline{b}.\, \prod_{k \in \mathbb{K}} \alpha_{\text{BA}}(\pi_k(\overline{b})) \\
\mathit{lift}(\gamma_{\text{AB}}) &= \lambda \overline{a}.\, \prod_{k \in \mathbb{K}} \gamma_{\text{AB}}(\pi_k(\overline{a}))
\end{aligned}$$

Proof. Let e, $\overline{a}$ be given. We proceed by structural induction on e.

Case n:

$$\begin{aligned}
&(\mathit{lift}(\hat{\alpha}) \circ \overline{\mathcal{B}'}[\![n]\!] \circ \mathit{lift}(\gamma_{\text{AB}}))\overline{a} \\
&= \mathit{lift}(\hat{\alpha})(\overline{\mathcal{B}'}[\![n]\!](\mathit{lift}(\gamma_{\text{AB}})\overline{a})) && \text{(by def. of } \circ) \\
&= \mathit{lift}(\hat{\alpha})(\prod_{k \in \mathbb{K}} \{n\}) && \text{(by def. of } \overline{\mathcal{B}'}) \\
&= \prod_{k \in \mathbb{K}} \hat{\alpha}(\{n\}) && \text{(by def. of } \mathit{lift}(\alpha_{\text{BA}})) \\
&= \prod_{k \in \mathbb{K}} \mathtt{n} && \text{(by def. of } \hat{\alpha}) \\
&= \overline{\mathcal{A}'}[\![n]\!]\overline{a} && \text{(by def. of } \overline{\mathcal{A}'})
\end{aligned}$$

Case `x`:

$$\begin{aligned}
&(\mathit{lift}(\hat{\alpha}) \circ \overline{\mathcal{B}'}[\![\mathtt{x}]\!] \circ \mathit{lift}(\gamma_{\text{AB}}))\overline{a} \\
&= \mathit{lift}(\hat{\alpha})(\overline{\mathcal{B}'}[\![\mathtt{x}]\!](\mathit{lift}(\gamma_{\text{AB}})\overline{a})) && \text{(by def. of } \circ) \\
&= \mathit{lift}(\hat{\alpha})(\prod_{k \in \mathbb{K}} \pi_k(\mathit{lift}(\gamma_{\text{AB}})\overline{a})(\mathtt{x})) && \text{(by def. of } \overline{\mathcal{B}'}) \\
&= \mathit{lift}(\hat{\alpha})(\prod_{k \in \mathbb{K}} \pi_k(\prod_{k' \in \mathbb{K}} \gamma_{\text{AB}}(\pi_{k'}(\overline{a})))(\mathtt{x})) \\
& && \text{(by def. of } \mathit{lift}(\gamma_{\text{AB}})) \\
&= \mathit{lift}(\hat{\alpha})(\prod_{k \in \mathbb{K}} \gamma_{\text{AB}}(\pi_k(\overline{a}))(\mathtt{x})) && \text{(shortcut constr.+proj.)} \\
&= \mathit{lift}(\hat{\alpha})(\prod_{k \in \mathbb{K}} \hat{\gamma}(\pi_k(\overline{a})(\mathtt{x}))) && \text{(by def. of } \gamma_{\text{AB}}) \\
&= \prod_{k \in \mathbb{K}} \hat{\alpha}(\hat{\gamma}(\pi_k(\overline{a})(\mathtt{x}))) && \text{(by def. of } \mathit{lift}(\alpha_{\text{BA}})) \\
&\mathrel{\dot{\sqsubseteq}} \prod_{k \in \mathbb{K}} \pi_k(\overline{a})(\mathtt{x}) && (\hat{\alpha} \circ \hat{\gamma} \text{ reductive}) \\
&= \overline{\mathcal{A}'}[\![\mathtt{x}]\!]\overline{a} && \text{(by def. of } \overline{\mathcal{A}'})
\end{aligned}$$

Case $e_0 \oplus e_1$:

$$\begin{aligned}
&(\mathit{lift}(\hat{\alpha}) \circ \overline{\mathcal{B}'}[\![e_0 \oplus e_1]\!] \circ \mathit{lift}(\gamma_{\text{AB}}))\overline{a} \\
&= \mathit{lift}(\hat{\alpha})(\overline{\mathcal{B}'}[\![e_0 \oplus e_1]\!](\mathit{lift}(\gamma_{\text{AB}})\overline{a})) && \text{(by def. of } \circ) \\
&= \mathit{lift}(\hat{\alpha})(\prod_{k \in \mathbb{K}} (\pi_k(\overline{\mathcal{B}'}[\![e_0]\!](\mathit{lift}(\gamma_{\text{AB}})\overline{a}))) \\
&\qquad \dot{\oplus}\ (\pi_k(\overline{\mathcal{B}'}[\![e_1]\!](\mathit{lift}(\gamma_{\text{AB}})\overline{a})))) && \text{(by def. of } \overline{\mathcal{B}'}) \\
&\mathrel{\dot{\sqsubseteq}} \mathit{lift}(\hat{\alpha})(\prod_{k \in \mathbb{K}} (\pi_k(\mathit{lift}(\hat{\gamma}) \circ \mathit{lift}(\hat{\alpha}) \circ \overline{\mathcal{B}'}[\![e_0]\!](\mathit{lift}(\gamma_{\text{AB}})\overline{a}))) \\
&\qquad \dot{\oplus}\ (\pi_k(\mathit{lift}(\hat{\gamma}) \circ \mathit{lift}(\hat{\alpha}) \circ \overline{\mathcal{B}'}[\![e_1]\!](\mathit{lift}(\gamma_{\text{AB}})\overline{a})))) \\
&\qquad (\mathit{lift}(\hat{\gamma}) \circ \mathit{lift}(\hat{\alpha}) \text{ extensive, } \mathit{lift}(\hat{\alpha}) \text{ monotone}) \\
&\mathrel{\dot{\sqsubseteq}} \mathit{lift}(\hat{\alpha})(\prod_{k \in \mathbb{K}} (\pi_k(\mathit{lift}(\hat{\gamma}) \circ \overline{\mathcal{A}'}[\![e_0]\!](\overline{a}))) \\
&\qquad \dot{\oplus}\ (\pi_k(\mathit{lift}(\hat{\gamma}) \circ \overline{\mathcal{A}'}[\![e_1]\!](\overline{a})))) \\
&\qquad \text{(by IH, } \mathit{lift}(\hat{\gamma}), \mathit{lift}(\hat{\alpha}) \text{ monotone}) \\
&= \mathit{lift}(\hat{\alpha})(\prod_{k \in \mathbb{K}} \hat{\gamma}(\pi_k(\overline{\mathcal{A}'}[\![e_0]\!](\overline{a}))) \mathbin{\dot{\oplus}} \hat{\gamma}(\pi_k(\overline{\mathcal{A}'}[\![e_1]\!](\overline{a})))) \\
&\qquad \text{(by def. of } \mathit{lift}(\hat{\gamma})) \\
&= \prod_{k \in \mathbb{K}} \hat{\alpha}(\hat{\gamma}(\pi_k(\overline{\mathcal{A}'}[\![e_0]\!](\overline{a}))) \mathbin{\dot{\oplus}} \hat{\gamma}(\pi_k(\overline{\mathcal{A}'}[\![e_1]\!](\overline{a})))) \\
&\qquad \text{(by def. of } \mathit{lift}(\hat{\alpha})) \\
&\mathrel{\dot{\sqsubseteq}} \prod_{k \in \mathbb{K}} \pi_k(\overline{\mathcal{A}'}[\![e_0]\!](\overline{a})) \mathbin{\hat{\oplus}} \pi_k(\overline{\mathcal{A}'}[\![e_1]\!](\overline{a})) && \text{(by def. of } \hat{\oplus}) \\
&= \overline{\mathcal{A}'}[\![e_0 \oplus e_1]\!]\overline{a} && \text{(by def. of } \overline{\mathcal{A}'})
\end{aligned}$$

□

Theorem 31 (Soundness of family-based statement analysis)**.**

$$\forall s, \overline{a}.\ (\mathit{lift}(\alpha_{\text{BA}}) \circ \overline{\mathcal{B}}[\![s]\!] \circ \mathit{lift}(\gamma_{\text{AB}}))\overline{a} \mathrel{\ddot{\sqsubseteq}} \overline{\mathcal{A}}[\![s]\!]\overline{a}$$

Proof. Let s, $\overline{a}$ be given. We proceed by structural induction on s.

Case `skip`:

$$\begin{aligned}
&(\mathit{lift}(\alpha_{\text{BA}}) \circ \overline{\mathcal{B}}[\![\mathtt{skip}]\!] \circ \mathit{lift}(\gamma_{\text{AB}}))\overline{a} \\
&= (\mathit{lift}(\alpha_{\text{BA}}) \circ (\lambda \overline{b}.\, \overline{b}) \circ \mathit{lift}(\gamma_{\text{AB}}))\overline{a} && \text{(by def. of } \overline{\mathcal{B}}) \\
&= (\mathit{lift}(\alpha_{\text{BA}}) \circ \mathit{lift}(\gamma_{\text{AB}}))\overline{a} && \text{(simplify)} \\
&\mathrel{\ddot{\sqsubseteq}} \overline{a} && (\mathit{lift}(\alpha_{\text{BA}}) \circ \mathit{lift}(\gamma_{\text{AB}}) \text{ is reductive}) \\
&= \overline{\mathcal{A}}[\![\mathtt{skip}]\!]\overline{a} && \text{(by def. of } \overline{\mathcal{A}})
\end{aligned}$$

Case $\texttt{x := } e$**:**

$$
\begin{aligned}
&(\mathit{lift}(\alpha_{\mathrm{BA}}) \circ \overline{\mathcal{B}}[\![\texttt{x := } e]\!] \circ \mathit{lift}(\gamma_{\mathrm{AB}}))\overline{a} \\
&= \mathit{lift}(\alpha_{\mathrm{BA}})(\overline{\mathcal{B}}[\![\texttt{x := } e]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a})) && \text{(by def. of } \circ) \\
&= \mathit{lift}(\alpha_{\mathrm{BA}})(\prod_{k\in\mathbb{K}}(\pi_k(\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a}))[\texttt{x} \mapsto \pi_k(\overline{\mathcal{B}'}[\![e]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a}))]) \\
& && \text{(by def. of } \overline{\mathcal{B}}) \\
&= \prod_{k\in\mathbb{K}} \alpha_{\mathrm{BA}}((\pi_k(\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a}))[\texttt{x} \mapsto \pi_k(\overline{\mathcal{B}'}[\![e]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a}))]) \\
& && \text{(by def. of } \mathit{lift}(\alpha_{\mathrm{BA}})) \\
&= \prod_{k\in\mathbb{K}} \alpha_{\mathrm{BA}}((\gamma_{\mathrm{AB}}(\pi_k(\overline{a})))[\texttt{x} \mapsto \pi_k(\overline{\mathcal{B}'}[\![e]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a}))]) \\
& && \text{(by def. of } \mathit{lift}(\gamma_{\mathrm{AB}})) \\
&= \prod_{k\in\mathbb{K}} ((\alpha_{\mathrm{BA}}(\gamma_{\mathrm{AB}}(\pi_k(\overline{a}))))[\texttt{x} \mapsto \hat{\alpha}(\pi_k(\overline{\mathcal{B}'}[\![e]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a})))]) \\
& && \text{(by def. of } \alpha_{\mathrm{BA}}) \\
&\ddot{\sqsubseteq} \prod_{k\in\mathbb{K}} (\pi_k(\overline{a})[\texttt{x} \mapsto \hat{\alpha}(\pi_k(\overline{\mathcal{B}'}[\![e]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a})))]) \\
& && (\alpha_{\mathrm{BA}} \circ \gamma_{\mathrm{AB}} \text{ reductive}) \\
&= \prod_{k\in\mathbb{K}} (\pi_k(\overline{a})[\texttt{x} \mapsto \pi_k(\mathit{lift}(\hat{\alpha})(\overline{\mathcal{B}'}[\![e]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a})))]) \\
& && \text{(identical } k \text{ entries)} \\
&\ddot{\sqsubseteq} \prod_{k\in\mathbb{K}} (\pi_k(\overline{a})[\texttt{x} \mapsto \pi_k(\overline{\mathcal{A}'}[\![e]\!]\overline{a})]) && \text{(By Lemma 30)} \\
&= \overline{\mathcal{A}}[\![\texttt{x := } e]\!]\overline{a} && \text{(by def. of } \overline{\mathcal{A}})
\end{aligned}
$$

Case $s_0 \texttt{ ; } s_1$**:**

$$
\begin{aligned}
&(\mathit{lift}(\alpha_{\mathrm{BA}}) \circ \overline{\mathcal{B}}[\![s_0 \texttt{ ; } s_1]\!] \circ \mathit{lift}(\gamma_{\mathrm{AB}}))\overline{a} \\
&= (\mathit{lift}(\alpha_{\mathrm{BA}}) \circ \overline{\mathcal{B}}[\![s_1]\!] \circ \overline{\mathcal{B}}[\![s_0]\!] \circ \mathit{lift}(\gamma_{\mathrm{AB}}))\overline{a} && \text{(by def. of } \overline{\mathcal{B}}) \\
&\ddot{\sqsubseteq} (\mathit{lift}(\alpha_{\mathrm{BA}}) \circ \overline{\mathcal{B}}[\![s_1]\!] \circ \mathit{lift}(\gamma_{\mathrm{AB}}) \circ \mathit{lift}(\alpha_{\mathrm{BA}}) \circ \overline{\mathcal{B}}[\![s_0]\!] \circ \mathit{lift}(\gamma_{\mathrm{AB}}))\overline{a} \\
& && (\mathit{lift}(\gamma_{\mathrm{AB}}) \circ \mathit{lift}(\alpha_{\mathrm{BA}}) \text{ extensive}) \\
&\ddot{\sqsubseteq} (\overline{\mathcal{A}}[\![s_1]\!] \circ \overline{\mathcal{A}}[\![s_0]\!])\overline{a} && \text{(by IH, twice)} \\
&= \overline{\mathcal{A}}[\![s_0 \texttt{ ; } s_1]\!]\overline{a} && \text{(by def. of } \overline{\mathcal{A}})
\end{aligned}
$$

Case $\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1$**:**

$$
\begin{aligned}
&(\mathit{lift}(\alpha_{\mathrm{BA}}) \circ \overline{\mathcal{B}}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!] \circ \mathit{lift}(\gamma_{\mathrm{AB}}))\overline{a} \\
&= \mathit{lift}(\alpha_{\mathrm{BA}})(\overline{\mathcal{B}}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a})) \\
& && \text{(by def. of } \circ) \\
&= \mathit{lift}(\alpha_{\mathrm{BA}})(\overline{\mathcal{B}}[\![s_0]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a}) \mathbin{\ddot{\cup}} \overline{\mathcal{B}}[\![s_1]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a})) \\
& && \text{(by def. of } \overline{\mathcal{B}}) \\
&= \mathit{lift}(\alpha_{\mathrm{BA}})(\overline{\mathcal{B}}[\![s_0]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a})) \mathbin{\ddot{\sqcup}} \mathit{lift}(\alpha_{\mathrm{BA}})(\overline{\mathcal{B}}[\![s_1]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a})) \\
& && (\mathit{lift}(\alpha_{\mathrm{BA}}) \text{ a CJM}) \\
&\ddot{\sqsubseteq} \overline{\mathcal{A}}[\![s_0]\!]\overline{a} \mathbin{\ddot{\sqcup}} \overline{\mathcal{A}}[\![s_1]\!]\overline{a} && \text{(by IH, twice)} \\
&= \overline{\mathcal{A}}[\![\texttt{if } e \texttt{ then } s_0 \texttt{ else } s_1]\!]\overline{a} && \text{(by def. of } \overline{\mathcal{A}})
\end{aligned}
$$

Case $\texttt{while } e \texttt{ do } s$**:** In this case our higher-order Galois connection reads:

$$
\begin{aligned}
\alpha_{\rightarrow}(\overline{\Phi}) &= \mathit{lift}(\alpha_{\mathrm{BA}}) \circ \overline{\Phi} \circ \mathit{lift}(\gamma_{\mathrm{AB}}) \\
\gamma_{\rightarrow}(\overline{\Phi}) &= \mathit{lift}(\gamma_{\mathrm{AB}}) \circ \overline{\Phi} \circ \mathit{lift}(\alpha_{\mathrm{BA}})
\end{aligned}
$$

First observe that for any given monotone $\overline{\Phi}$

$$
\begin{aligned}
&(\alpha_{\rightarrow} \circ (\lambda\overline{\Phi}.\, \lambda\overline{b}.\, \overline{b} \mathbin{\ddot{\cup}} \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\overline{b})) \circ \gamma_{\rightarrow})\overline{\Phi} \\
&= \alpha_{\rightarrow}((\lambda\overline{\Phi}.\, \lambda\overline{b}.\, \overline{b} \mathbin{\ddot{\cup}} \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\overline{b}))(\gamma_{\rightarrow}(\overline{\Phi}))) && \text{(by def. of } \circ) \\
&= \alpha_{\rightarrow}(\lambda\overline{b}.\, \overline{b} \mathbin{\ddot{\cup}} \gamma_{\rightarrow}(\overline{\Phi})(\overline{\mathcal{B}}[\![s]\!]\overline{b})) && (\beta\text{-reduction}) \\
&= \mathit{lift}(\alpha_{\mathrm{BA}}) \circ (\lambda\overline{b}.\, \overline{b} \mathbin{\ddot{\cup}} \gamma_{\rightarrow}(\overline{\Phi})(\overline{\mathcal{B}}[\![s]\!]\overline{b})) \circ \mathit{lift}(\gamma_{\mathrm{AB}}) \\
& && \text{(by def. of } \alpha_{\rightarrow}) \\
&= \lambda\overline{a}.\, (\mathit{lift}(\alpha_{\mathrm{BA}}) \circ (\lambda\overline{b}.\, \overline{b} \mathbin{\ddot{\cup}} \gamma_{\rightarrow}(\overline{\Phi})(\overline{\mathcal{B}}[\![s]\!]\overline{b})) \circ \mathit{lift}(\gamma_{\mathrm{AB}}))\overline{a} \\
& && (\eta\text{-expansion}) \\
&= \lambda\overline{a}.\, \mathit{lift}(\alpha_{\mathrm{BA}})(\lambda\overline{b}.\, \overline{b} \mathbin{\ddot{\cup}} \gamma_{\rightarrow}(\overline{\Phi})(\overline{\mathcal{B}}[\![s]\!]\overline{b}))(\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a}) \\
& && \text{(by def. of } \circ) \\
&= \lambda\overline{a}.\, \mathit{lift}(\alpha_{\mathrm{BA}})(\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a} \mathbin{\ddot{\cup}} \gamma_{\rightarrow}(\overline{\Phi})(\overline{\mathcal{B}}[\![s]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a}))) \\
& && (\beta\text{-reduction}) \\
&= \lambda\overline{a}.\, \mathit{lift}(\alpha_{\mathrm{BA}})(\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a}) \mathbin{\ddot{\sqcup}} \mathit{lift}(\alpha_{\mathrm{BA}})(\gamma_{\rightarrow}(\overline{\Phi})(\overline{\mathcal{B}}[\![s]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a}))) \\
& && (\mathit{lift}(\alpha_{\mathrm{BA}}) \text{ a CJM}) \\
&\ddot{\sqsubseteq} \lambda\overline{a}.\, \overline{a} \mathbin{\ddot{\sqcup}} \mathit{lift}(\alpha_{\mathrm{BA}})(\gamma_{\rightarrow}(\overline{\Phi})(\overline{\mathcal{B}}[\![s]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a}))) \\
& && (\mathit{lift}(\alpha_{\mathrm{BA}}) \circ \mathit{lift}(\gamma_{\mathrm{AB}}) \text{ reductive}) \\
&= \lambda\overline{a}.\, \overline{a} \mathbin{\ddot{\sqcup}} \mathit{lift}(\alpha_{\mathrm{BA}})((\mathit{lift}(\gamma_{\mathrm{AB}}) \circ \overline{\Phi} \circ \mathit{lift}(\alpha_{\mathrm{BA}}))(\overline{\mathcal{B}}[\![s]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a}))) \\
& && \text{(by def. of } \gamma_{\rightarrow}) \\
&\ddot{\sqsubseteq} \lambda\overline{a}.\, \overline{a} \mathbin{\ddot{\sqcup}} (\overline{\Phi} \circ \mathit{lift}(\alpha_{\mathrm{BA}}))(\overline{\mathcal{B}}[\![s]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a})) \\
& && (\mathit{lift}(\alpha_{\mathrm{BA}}) \circ \mathit{lift}(\gamma_{\mathrm{AB}}) \text{ reductive}) \\
&\ddot{\sqsubseteq} \lambda\overline{a}.\, \overline{a} \mathbin{\ddot{\sqcup}} \overline{\Phi}(\overline{\mathcal{A}}[\![s]\!]\overline{a}) && \text{(by IH, } \overline{\Phi} \text{ monotone)}
\end{aligned}
$$

Now we can utilize that observation

$$
\begin{aligned}
&(\mathit{lift}(\alpha_{\mathrm{BA}}) \circ \overline{\mathcal{B}}[\![\texttt{while } e \texttt{ do } s]\!] \circ \mathit{lift}(\gamma_{\mathrm{AB}}))\overline{a} \\
&= \alpha_{\rightarrow}(\overline{\mathcal{B}}[\![\texttt{while } e \texttt{ do } s]\!])\overline{a} && \text{(by def. of } \alpha_{\rightarrow}) \\
&= \alpha_{\rightarrow}(\mathrm{lfp}\, \lambda\overline{\Phi}.\, \lambda\overline{b}.\, \overline{b} \mathbin{\ddot{\cup}} \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\overline{b}))\overline{a} && \text{(by def. of } \overline{\mathcal{B}}) \\
&\ddot{\sqsubseteq} (\mathrm{lfp}\, \alpha_{\rightarrow} \circ (\lambda\overline{\Phi}.\, \lambda\overline{b}.\, \overline{b} \mathbin{\ddot{\cup}} \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\overline{b})) \circ \gamma_{\rightarrow})\overline{a} \\
& && \text{(by the fixed point transfer theorem)} \\
&\ddot{\sqsubseteq} (\mathrm{lfp}(\lambda\overline{a}.\, \overline{a} \mathbin{\ddot{\sqcup}} \overline{\Phi}(\overline{\mathcal{A}}[\![s]\!]\overline{a})))\overline{a} \\
& && \text{(by the fixed point transfer theorem, above)} \\
&= \overline{\mathcal{A}}[\![\texttt{while } e \texttt{ do } s]\!]\overline{a} && \text{(by def. of } \overline{\mathcal{A}})
\end{aligned}
$$

Case `#if` $\varphi\ s$**:**

$$
\begin{aligned}
&(\mathit{lift}(\alpha_{\mathrm{BA}}) \circ \overline{\mathcal{B}}[\![\texttt{\#if}\ \varphi\ s]\!] \circ \mathit{lift}(\gamma_{\mathrm{AB}}))\overline{a} \\
&= \mathit{lift}(\alpha_{\mathrm{BA}})(\overline{\mathcal{B}}[\![\texttt{\#if}\ \varphi\ s]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a})) && \text{(by def. of } \circ) \\
&= \mathit{lift}(\alpha_{\mathrm{BA}})(\prod_{k\in\mathbb{K}} \begin{cases} \pi_k(\overline{\mathcal{B}}[\![s]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a})) & k \vDash \varphi \\ \pi_k(\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a}) & k \nvDash \varphi \end{cases}) \\
& && \text{(by def. of } \overline{\mathcal{B}}) \\
&= \prod_{k\in\mathbb{K}} \begin{cases} \alpha_{\mathrm{BA}}(\pi_k(\overline{\mathcal{B}}[\![s]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a}))) & k \vDash \varphi \\ \alpha_{\mathrm{BA}}(\pi_k(\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a})) & k \nvDash \varphi \end{cases} \\
& && \text{(by def. of } \mathit{lift}(\alpha_{\mathrm{BA}})) \\
&= \prod_{k\in\mathbb{K}} \begin{cases} \alpha_{\mathrm{BA}}(\pi_k(\overline{\mathcal{B}}[\![s]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a}))) & k \vDash \varphi \\ \alpha_{\mathrm{BA}}(\gamma_{\mathrm{AB}}(\pi_k(\overline{a}))) & k \nvDash \varphi \end{cases} \\
& && \text{(by def. of } \mathit{lift}(\gamma_{\mathrm{AB}})) \\
&\ddot{\sqsubseteq} \prod_{k\in\mathbb{K}} \begin{cases} \alpha_{\mathrm{BA}}(\pi_k(\overline{\mathcal{B}}[\![s]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a}))) & k \vDash \varphi \\ \pi_k(\overline{a}) & k \nvDash \varphi \end{cases} \\
& && (\alpha_{\mathrm{BA}} \circ \gamma_{\mathrm{AB}} \text{ reductive}) \\
&= \prod_{k\in\mathbb{K}} \begin{cases} \pi_k(\mathit{lift}(\alpha_{\mathrm{BA}})(\overline{\mathcal{B}}[\![s]\!](\mathit{lift}(\gamma_{\mathrm{AB}})\overline{a}))) & k \vDash \varphi \\ \pi_k(\overline{a}) & k \nvDash \varphi \end{cases} \\
& && \text{(identical } k \text{ entries)} \\
&\ddot{\sqsubseteq} \prod_{k\in\mathbb{K}} \begin{cases} \pi_k(\overline{\mathcal{A}}[\![s]\!]\overline{a}) & k \vDash \varphi \\ \pi_k(\overline{a}) & k \nvDash \varphi \end{cases} && \text{(by IH)} \\
&= \overline{\mathcal{A}}[\![\texttt{\#if}\ \varphi\ s]\!]\overline{a} && \text{(by def. of } \overline{\mathcal{A}})
\end{aligned}
$$

□

M. Family-based monotonicity proofs

Lemma 32 ($\overline{\mathcal{C}'}$ monotone)**.** $\forall e, \overline{c}, \overline{c}' \,.\, \overline{c} \dot{\subseteq} \overline{c}' \implies \overline{\mathcal{C}'}[\![e]\!]\overline{c} \dot{\subseteq} \overline{\mathcal{C}'}[\![e]\!]\overline{c}'$

Proof. Let $e, \overline{c} \dot{\subseteq} \overline{c}'$ be given.

$$
\begin{aligned}
&\overline{\mathcal{C}'}[\![e]\!]\overline{c} \\
&= \prod_{k\in\mathbb{K}} \mathcal{C}'[\![e]\!](\pi_k(\overline{c})) && \text{(by def. of } \mathcal{C}') \\
&\dot{\subseteq} \prod_{k\in\mathbb{K}} \mathcal{C}'[\![e]\!](\pi_k(\overline{c}')) && \text{(by assumption, monotonicity of } \mathcal{C}') \\
&= \overline{\mathcal{C}'}[\![e]\!]\overline{c}' && \text{(by def. of } \mathcal{C}')
\end{aligned}
$$

□

Theorem 33 ($\overline{\mathcal{C}}$ monotone)**.** $\forall s, \overline{c}, \overline{c}' \,.\, \overline{c} \dot{\subseteq} \overline{c}' \implies \overline{\mathcal{C}}[\![s]\!]\overline{c} \dot{\subseteq} \overline{\mathcal{C}}[\![s]\!]\overline{c}'$

Proof. Let $s, \overline{c} \dot{\subseteq} \overline{c}'$ be given.

$$
\begin{aligned}
&\overline{\mathcal{C}}[\![s]\!]\overline{c} \\
&= \prod_{k\in\mathbb{K}} \mathcal{C}[\![P[\![s]\!]_k]\!](\pi_k(\overline{c})) && \text{(by def. of } \overline{\mathcal{C}}) \\
&\dot{\subseteq} \prod_{k\in\mathbb{K}} \mathcal{C}[\![P[\![s]\!]_k]\!](\pi_k(\overline{c}')) && \text{(by assumption, monotonicity of } \mathcal{C}) \\
&= \overline{\mathcal{C}}[\![s]\!]\overline{c}' && \text{(by def. of } \overline{\mathcal{C}})
\end{aligned}
$$

We consider in particular the fixed point definition of while:

$$
\begin{aligned}
&\overline{\mathcal{C}}[\![\texttt{while } e \texttt{ do } s]\!] = \\
&\quad \mathrm{lfp}(\lambda\overline{\Phi}.\, \lambda\overline{c}.\ \prod_{k''\in\mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}]))\} \\
&\qquad \dot{\cup}\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k''\in\mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}]))\})))
\end{aligned}
$$

For shorthand notation, we let

$$
\begin{aligned}
F = \lambda\overline{\Phi}.\, \lambda\overline{c}.\ &\prod_{k''\in\mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}]))\} \\
&\dot{\cup}\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k''\in\mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}]))\}))
\end{aligned}
$$

.

For a monotone $\overline{\Phi}$, $F\overline{\Phi}$ is monotone: Let $\overline{c} \dot{\subseteq} \overline{c}'$ be given:

$$
\begin{aligned}
&F\overline{\Phi}\overline{c} \\
&= \prod_{k''\in\mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}]))\} \\
&\quad \dot{\cup}\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k''\in\mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}]))\})) \\
&\qquad\qquad \text{(by def. of } F) \\
&\dot{\subseteq} \prod_{k''\in\mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}') \mid 0 \in \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}]))\} \\
&\quad \dot{\cup}\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k''\in\mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}]))\})) \\
&\qquad\qquad \text{(by assumption, def. of } \cup) \\
&\dot{\subseteq} \prod_{k''\in\mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}') \mid 0 \in \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}]))\} \\
&\quad \dot{\cup}\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k''\in\mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}') \mid 0 \notin \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}]))\})) \\
&\qquad\qquad \text{(monotonicity of } \overline{\mathcal{C}} \text{ and } \overline{\Phi}) \\
&= F\overline{\Phi}\overline{c}' && \text{(by def. of } F)
\end{aligned}
$$

As a result, F can be seen as an operator over the domain of monotone functions.

Now we argue that F is itself monotone. Let monotone functions $\overline{\Phi} \ddot{\subseteq} \overline{\Phi}'$ be given.

$$
\begin{aligned}
&F\overline{\Phi} \\
&= \lambda\overline{c}.\ \prod_{k''\in\mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}]))\} \\
&\quad \dot{\cup}\, \overline{\Phi}(\overline{\mathcal{C}}[\![s]\!](\prod_{k''\in\mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}]))\})) \\
&\qquad\qquad \text{(by def. of } F) \\
&\ddot{\subseteq} \lambda\overline{c}.\ \prod_{k''\in\mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \in \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}]))\} \\
&\quad \dot{\cup}\, \overline{\Phi}'(\overline{\mathcal{C}}[\![s]\!](\prod_{k''\in\mathbb{K}} \{\sigma \in \pi_{k''}(\overline{c}) \mid 0 \notin \pi_{k''}(\overline{\mathcal{C}'}[\![e]\!](\dot{\varnothing}[k'' \mapsto \{\sigma\}]))\})) \\
&\qquad\qquad \text{(by assumption)} \\
&= F\overline{\Phi}' && \text{(by def. of } F)
\end{aligned}
$$

This ensures that the while-case is well defined by Tarski's fixed point theorem (a monotone operator over the complete lattice of monotone functions). Since the fixed point is itself a member of the lattice of monotone functions, we thereby confirm monotonicity of the while rule.

□

Lemma 34 ($\overline{\mathcal{B}'}$ monotone)**.**

$$\forall e, \overline{b}, \overline{b}' \,.\, \overline{b} \ddot{\subseteq} \overline{b}' \implies \overline{\mathcal{B}'}[\![e]\!]\overline{b} \ddot{\subseteq} \overline{\mathcal{B}'}[\![e]\!]\overline{b}'$$

Proof. Let e and $\overline{b} \ddot{\subseteq} \overline{b}'$ be given.

$$
\begin{aligned}
&\overline{\mathcal{B}'}[\![e]\!]\overline{b} \\
&= \prod_{k\in\mathbb{K}} \mathcal{B}'[\![e]\!](\pi_k(\overline{b})) && \text{(by def. of } \overline{\mathcal{B}'}) \\
&\ddot{\subseteq} \prod_{k\in\mathbb{K}} \mathcal{B}'[\![e]\!](\pi_k(\overline{b}')) && \text{(by assumption, monotonicity of } \mathcal{B}') \\
&= \overline{\mathcal{B}'}[\![e]\!]\overline{b}' && \text{(by def. of } \overline{\mathcal{B}'})
\end{aligned}
$$

□

Theorem 35 ($\overline{\mathcal{B}}$ monotone)**.**

$$\forall s, \overline{b}, \overline{b}' \,.\, \overline{b} \ddot{\subseteq} \overline{b}' \implies \overline{\mathcal{B}}[\![s]\!]\overline{b} \ddot{\subseteq} \overline{\mathcal{B}}[\![s]\!]\overline{b}'$$

Proof. Let s and $\overline{b} \ddot{\subseteq} \overline{b}'$ be given.

$$
\begin{aligned}
&\overline{\mathcal{B}}[\![s]\!]\overline{b} \\
&= \prod_{k\in\mathbb{K}} \mathcal{B}[\![P[\![s]\!]_k]\!](\pi_k(\overline{b})) && \text{(by def. of } \overline{\mathcal{B}}) \\
&\ddot{\subseteq} \prod_{k\in\mathbb{K}} \mathcal{B}[\![P[\![s]\!]_k]\!](\pi_k(\overline{b}')) && \text{(by Theorem 20)} \\
&= \overline{\mathcal{B}}[\![s]\!]\overline{b}' && \text{(by def. of } \overline{\mathcal{B}})
\end{aligned}
$$

Again we consider in particular the fixed point definition of while:

$$\overline{\mathcal{B}}[\![\texttt{while } e \texttt{ do } s]\!] = \mathrm{lfp}\, \lambda\overline{\Phi}.\, \lambda\overline{b}.\, \overline{b} \ddot{\cup} \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\overline{b})$$

For shorthand notation, we let $F = \lambda\overline{\Phi}.\, \lambda\overline{b}.\, \overline{b} \ddot{\cup} \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\overline{b})$.
For a monotone $\overline{\Phi}$, $F\overline{\Phi}$ is monotone: Let $\overline{b} \ddot{\subseteq} \overline{b}'$ be given:

$$
\begin{aligned}
&F\overline{\Phi}\overline{b} \\
&= \overline{b} \ddot{\cup} \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\overline{b}) && \text{(by def. of } F) \\
&\ddot{\subseteq} \overline{b}' \ddot{\cup} \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\overline{b}) && \text{(by def. of } \ddot{\cup}) \\
&\ddot{\subseteq} \overline{b}' \ddot{\cup} \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\overline{b}') && \text{(monotonicity of } \overline{\mathcal{B}} \text{ and } \overline{\Phi}) \\
&= F\overline{\Phi}\overline{b}' && \text{(by def. of } F)
\end{aligned}
$$

As a result, F can be seen as an operator over the domain of monotone functions.

Now we argue that F is itself monotone. Let monotone functions $\overline{\Phi} \mathrel{\ddot{\sqsubseteq}} \overline{\Phi}'$ be given.

$$\begin{aligned}
& F\overline{\Phi} \\
&= \lambda \overline{b}.\, \overline{b} \mathbin{\ddot{\cup}} \overline{\Phi}(\overline{\mathcal{B}}[\![s]\!]\overline{b}) && \text{(by def. of } F\text{)} \\
&\mathrel{\ddot{\sqsubseteq}} \lambda \overline{b}.\, \overline{b} \mathbin{\ddot{\cup}} \overline{\Phi}'(\overline{\mathcal{B}}[\![s]\!]\overline{b}) && \text{(by assumption)} \\
&= F\overline{\Phi}' && \text{(by def. of } F\text{)}
\end{aligned}$$

Again this ensures that the while-case is well defined by Tarski's fixed point theorem (a monotone operator over the complete lattice of monotone functions). Since the fixed point is itself a member of the lattice of monotone functions, we thereby confirm monotonicity of the while rule.

□

Lemma 36 ($\overline{\mathcal{A}'}$ monotone)**.** $\forall e, \overline{a}, \overline{a}'.\, \overline{a} \mathrel{\ddot{\sqsubseteq}} \overline{a}' \implies \overline{\mathcal{A}'}[\![e]\!]\overline{a} \mathrel{\ddot{\sqsubseteq}} \overline{\mathcal{A}'}[\![e]\!]\overline{a}'$

Proof. Let $e, \overline{a} \mathrel{\ddot{\sqsubseteq}} \overline{a}'$ be given.

$$\begin{aligned}
& \overline{\mathcal{A}'}[\![e]\!]\overline{a} \\
&= \prod_{k \in \mathbb{K}} \mathcal{A}'[\![e]\!](\pi_k(\overline{a})) && \text{(by def. of } \overline{\mathcal{A}'}\text{)} \\
&\mathrel{\ddot{\sqsubseteq}} \prod_{k \in \mathbb{K}} \mathcal{A}'[\![e]\!](\pi_k(\overline{a}')) && \text{(by assumption, monotonicity of } \mathcal{A}'\text{)} \\
&= \overline{\mathcal{A}'}[\![e]\!]\overline{a}' && \text{(by def. of } \overline{\mathcal{A}'}\text{)}
\end{aligned}$$

□

Theorem 37 ($\overline{\mathcal{A}}$ monotone)**.** $\forall s, \overline{a}, \overline{a}'.\, \overline{a} \mathrel{\ddot{\sqsubseteq}} \overline{a}' \implies \overline{\mathcal{A}}[\![s]\!]\overline{a} \mathrel{\ddot{\sqsubseteq}} \overline{\mathcal{A}}[\![s]\!]\overline{a}'$

Proof. Let $s, \overline{a} \mathrel{\ddot{\sqsubseteq}} \overline{a}'$ be given.

$$\begin{aligned}
& \overline{\mathcal{A}}[\![s]\!]\overline{a} \\
&= \prod_{k \in \mathbb{K}} \mathcal{A}[\![P[\![s]\!]_k]\!](\pi_k(\overline{a})) && \text{(by def. of } \overline{\mathcal{A}}\text{)} \\
&\mathrel{\ddot{\sqsubseteq}} \prod_{k \in \mathbb{K}} \mathcal{A}[\![P[\![s]\!]_k]\!](\pi_k(\overline{a}')) && \text{(by Theorem 22, def. of } \mathrel{\ddot{\sqsubseteq}}\text{)} \\
&= \overline{\mathcal{A}}[\![s]\!]\overline{a}' && \text{(by def. of } \overline{\mathcal{A}}\text{)}
\end{aligned}$$

Yet again we consider in particular the fixed point definition of while:

$$\overline{\mathcal{A}}[\![\texttt{while}\ e\ \texttt{do}\ s]\!] = \mathrm{lfp}\, \lambda\overline{\Phi}.\, \lambda\overline{a}.\, \overline{a} \mathbin{\ddot{\cup}} \overline{\Phi}(\overline{\mathcal{A}}[\![s]\!]\overline{a})$$

For shorthand notation, we let $F = \lambda\overline{\Phi}.\, \lambda\overline{a}.\, \overline{a} \mathbin{\ddot{\cup}} \overline{\Phi}(\overline{\mathcal{A}}[\![s]\!]\overline{a})$.
For a monotone $\overline{\Phi}$, $F\overline{\Phi}$ is monotone: Let $\overline{a} \mathrel{\ddot{\sqsubseteq}} \overline{a}'$ be given:

$$\begin{aligned}
& F\overline{\Phi}\overline{a} \\
&= \overline{a} \mathbin{\ddot{\cup}} \overline{\Phi}(\overline{\mathcal{A}}[\![s]\!]\overline{a}) && \text{(by def. of } F\text{)} \\
&\mathrel{\ddot{\sqsubseteq}} \overline{a}' \mathbin{\ddot{\cup}} \overline{\Phi}(\overline{\mathcal{A}}[\![s]\!]\overline{a}) && \text{(by def. of } \mathbin{\ddot{\cup}}\text{)} \\
&\mathrel{\ddot{\sqsubseteq}} \overline{a}' \mathbin{\ddot{\cup}} \overline{\Phi}(\overline{\mathcal{A}}[\![s]\!]\overline{a}') && \text{(monotonicity of } \overline{\mathcal{A}} \text{ and } \overline{\Phi}\text{)} \\
&= F\overline{\Phi}\overline{a}' && \text{(by def. of } F\text{)}
\end{aligned}$$

As a result, F can be seen as an operator over the domain of monotone functions.

Now we argue that F is itself monotone. Let monotone functions $\overline{\Phi} \mathrel{\ddot{\sqsubseteq}} \overline{\Phi}'$ be given.

$$\begin{aligned}
& F\overline{\Phi} \\
&= \lambda\overline{a}.\, \overline{a} \mathbin{\ddot{\cup}} \overline{\Phi}(\overline{\mathcal{A}}[\![s]\!]\overline{a}) && \text{(by def. of } F\text{)} \\
&\mathrel{\ddot{\sqsubseteq}} \lambda\overline{a}.\, \overline{a} \mathbin{\ddot{\cup}} \overline{\Phi}'(\overline{\mathcal{A}}[\![s]\!]\overline{a}) && \text{(by assumption)} \\
&= F\overline{\Phi}' && \text{(by def. of } F\text{)}
\end{aligned}$$

Yet again this ensures that the while-case is well defined by Tarski's fixed point theorem (a monotone operator over the complete lattice of monotone functions). Since the fixed point is itself a member of the lattice of monotone functions, we thereby confirm monotonicity of the while rule. □

N. Proof of Theorem 7: Lifted data-flow equation soundness

Note: the main paper version of the data-flow equations are purely statement based, whereas the below development also formulates data-flow equations for (labelled) expressions. Because of the below equality, analyzing expressions with $\overline{\mathcal{A}'}$ or with data-flow equations is equivalent.

We prove that a solution $[\![-]\!]_{\overline{\mathsf{in}}}$, $[\![-]\!]_{\overline{\mathsf{out}}}$ to the lifted data-flow constraints is sound wrt. to the lifted analysis:

$$\overline{\mathcal{A}'}[\![e^\ell]\!]([\![e^\ell]\!]_{\overline{\mathsf{in}}}) = [\![e^\ell]\!]_{\overline{\mathsf{out}}}$$
$$\overline{\mathcal{A}}[\![s^\ell]\!]([\![s^\ell]\!]_{\overline{\mathsf{in}}}) \ddot{\sqsubseteq} [\![s^\ell]\!]_{\overline{\mathsf{out}}}$$

for the following expression related equations:

$$\forall k \in \mathbb{K}.\ \pi_k([\![n^\ell]\!]_{\overline{\mathsf{out}}}) = \mathtt{n}$$
$$\forall k \in \mathbb{K}.\ \pi_k([\![\mathtt{x}^\ell]\!]_{\overline{\mathsf{out}}}) = \pi_k([\![\mathtt{x}^\ell]\!]_{\overline{\mathsf{in}}})(\mathtt{x})$$
$$[\![e_0^{\ell_0}]\!]_{\overline{\mathsf{in}}} = [\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]_{\overline{\mathsf{in}}}$$
$$[\![e_1^{\ell_1}]\!]_{\overline{\mathsf{in}}} = [\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]_{\overline{\mathsf{in}}}$$
$$\forall k \in \mathbb{K}.\ \pi_k([\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]_{\overline{\mathsf{out}}}) = \pi_k([\![e_0^{\ell_0}]\!]_{\overline{\mathsf{out}}}) \mathbin{\widehat{\oplus}} \pi_k([\![e_1^{\ell_1}]\!]_{\overline{\mathsf{out}}})$$

and for the following statement related equations:

$$[\![\mathtt{skip}^\ell]\!]_{\overline{\mathsf{in}}} = [\![\mathtt{skip}^\ell]\!]_{\overline{\mathsf{out}}}$$
$$[\![e^{\ell_0}]\!]_{\overline{\mathsf{in}}} = [\![\mathtt{x} :=^\ell e^{\ell_0}]\!]_{\overline{\mathsf{in}}}$$
$$\forall k \in \mathbb{K}.\ \pi_k([\![\mathtt{x} :=^\ell e^{\ell_0}]\!]_{\overline{\mathsf{out}}}) = \pi_k([\![\mathtt{x} :=^\ell e^{\ell_0}]\!]_{\overline{\mathsf{in}}})[\mathtt{x} \mapsto \pi_k([\![e^{\ell_0}]\!]_{\overline{\mathsf{out}}})]$$
$$[\![s_0^{\ell_0}]\!]_{\overline{\mathsf{in}}} = [\![s_0^{\ell_0} ;^\ell s_1^{\ell_1}]\!]_{\overline{\mathsf{in}}}$$
$$[\![s_1^{\ell_1}]\!]_{\overline{\mathsf{in}}} = [\![s_0^{\ell_0}]\!]_{\overline{\mathsf{out}}}$$
$$[\![s_0^{\ell_0} ;^\ell s_1^{\ell_1}]\!]_{\overline{\mathsf{out}}} = [\![s_1^{\ell_1}]\!]_{\overline{\mathsf{out}}}$$
$$[\![\mathtt{if}^\ell\ e\ \mathtt{then}\ s_0^{\ell_0}\ \mathtt{else}\ s_1^{\ell_1}]\!]_{\overline{\mathsf{out}}} = [\![s_0^{\ell_0}]\!]_{\overline{\mathsf{out}}} \ddot{\sqcup} [\![s_1^{\ell_1}]\!]_{\overline{\mathsf{out}}}$$
$$[\![s_0^{\ell_0}]\!]_{\overline{\mathsf{in}}} = [\![\mathtt{if}^\ell\ e\ \mathtt{then}\ s_0^{\ell_0}\ \mathtt{else}\ s_1^{\ell_1}]\!]_{\overline{\mathsf{in}}}$$
$$[\![s_1^{\ell_1}]\!]_{\overline{\mathsf{in}}} = [\![\mathtt{if}^\ell\ e\ \mathtt{then}\ s_0^{\ell_0}\ \mathtt{else}\ s_1^{\ell_1}]\!]_{\overline{\mathsf{in}}}$$
$$[\![\mathtt{while}^\ell\ e\ \mathtt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{out}}} = [\![s^{\ell_0}]\!]_{\overline{\mathsf{in}}}$$
$$[\![s^{\ell_0}]\!]_{\overline{\mathsf{in}}} = [\![\mathtt{while}^\ell\ e\ \mathtt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}} \ddot{\sqcup} [\![s^{\ell_0}]\!]_{\overline{\mathsf{out}}}$$
$$\forall k \in \mathbb{K}.\ \pi_k([\![\#\mathtt{if}^\ell\ \varphi\ s^{\ell_0}]\!]_{\overline{\mathsf{out}}}) = \pi_k([\![s^{\ell_0}]\!]_{\overline{\mathsf{out}}}) \quad \text{if} \quad k \vDash \varphi$$
$$\forall k \in \mathbb{K}.\ \pi_k([\![\#\mathtt{if}^\ell\ \varphi\ s^{\ell_0}]\!]_{\overline{\mathsf{out}}}) = \pi_k([\![\#\mathtt{if}^\ell\ \varphi\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) \quad \text{if} \quad k \nvDash \varphi$$
$$\forall k \in \mathbb{K}.\ \pi_k([\![s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) = \pi_k([\![\#\mathtt{if}^\ell\ \varphi\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) \quad \text{if} \quad k \vDash \varphi$$

Proof. **Expression soundness:** Let e^ℓ be given. We proceed by structural induction on e^ℓ.

Case n^ℓ:

$$\overline{\mathcal{A}'}[\![n^\ell]\!]([\![n^\ell]\!]_{\overline{\mathsf{in}}}) = \prod_{k\in\mathbb{K}} \mathtt{n} = \prod_{k\in\mathbb{K}} \pi_k([\![n^\ell]\!]_{\overline{\mathsf{out}}}) = [\![n^\ell]\!]_{\overline{\mathsf{out}}}$$
(by def. of $\overline{\mathcal{A}'}$, $[\![n^\ell]\!]_{\overline{\mathsf{out}}}$)

Case $\mathtt{x}^\ell$:

$$\overline{\mathcal{A}'}[\![\mathtt{x}^\ell]\!]([\![\mathtt{x}^\ell]\!]_{\overline{\mathsf{in}}}) = \prod_{k\in\mathbb{K}} \pi_k([\![\mathtt{x}^\ell]\!]_{\overline{\mathsf{in}}})(\mathtt{x}) = \prod_{k\in\mathbb{K}} \pi_k([\![\mathtt{x}^\ell]\!]_{\overline{\mathsf{out}}}) = [\![\mathtt{x}^\ell]\!]_{\overline{\mathsf{out}}}$$
(by def. of $\overline{\mathcal{A}'}$, $[\![\mathtt{x}^\ell]\!]_{\overline{\mathsf{out}}}$)

Case $e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}$:

$$\begin{aligned}
&\overline{\mathcal{A}'}[\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]([\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]_{\overline{\mathsf{in}}}) \\
&= \prod_{k\in\mathbb{K}} \pi_k(\overline{\mathcal{A}'}[\![e_0^{\ell_0}]\!]([\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]_{\overline{\mathsf{in}}})) \\
&\qquad \mathbin{\widehat{\oplus}} \pi_k(\overline{\mathcal{A}'}[\![e_1^{\ell_1}]\!]([\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]_{\overline{\mathsf{in}}})) \quad \text{(by def. of } \overline{\mathcal{A}'}) \\
&= \prod_{k\in\mathbb{K}} \pi_k(\overline{\mathcal{A}'}[\![e_0^{\ell_0}]\!]([\![e_0^{\ell_0}]\!]_{\overline{\mathsf{in}}})) \mathbin{\widehat{\oplus}} \pi_k(\overline{\mathcal{A}'}[\![e_1^{\ell_1}]\!]([\![e_1^{\ell_1}]\!]_{\overline{\mathsf{in}}})) \\
&\qquad \text{(by def. of } [\![e_0^{\ell_0}]\!]_{\overline{\mathsf{in}}}, [\![e_1^{\ell_1}]\!]_{\overline{\mathsf{in}}}) \\
&= \prod_{k\in\mathbb{K}} \pi_k([\![e_0^{\ell_0}]\!]_{\overline{\mathsf{out}}}) \mathbin{\widehat{\oplus}} \pi_k([\![e_1^{\ell_1}]\!]_{\overline{\mathsf{out}}}) \quad \text{(by IH, twice)} \\
&= \prod_{k\in\mathbb{K}} \pi_k([\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]_{\overline{\mathsf{out}}}) \quad \text{(by def. of } [\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]_{\overline{\mathsf{out}}}) \\
&= [\![e_0^{\ell_0} \oplus^\ell e_1^{\ell_1}]\!]_{\overline{\mathsf{out}}} \quad \text{(simplify)}
\end{aligned}$$

Statement soundness: Let s^ℓ be given. We proceed by structural induction on s^ℓ.

Case $\mathtt{skip}^\ell$:

$$\overline{\mathcal{A}}[\![\mathtt{skip}^\ell]\!]([\![\mathtt{skip}^\ell]\!]_{\overline{\mathsf{in}}}) = [\![\mathtt{skip}^\ell]\!]_{\overline{\mathsf{in}}} = [\![\mathtt{skip}^\ell]\!]_{\overline{\mathsf{out}}}$$
(by def. of $\overline{\mathcal{A}}$,$[\![\mathtt{skip}^\ell]\!]_{\overline{\mathsf{in}}}$)

Case $\mathtt{x} :=^\ell e^{\ell_0}$:

$$\begin{aligned}
&\overline{\mathcal{A}}[\![\mathtt{x} :=^\ell e^{\ell_0}]\!]([\![\mathtt{x} :=^\ell e^{\ell_0}]\!]_{\overline{\mathsf{in}}}) \\
&= \prod_{k\in\mathbb{K}} (\pi_k([\![\mathtt{x} :=^\ell e^{\ell_0}]\!]_{\overline{\mathsf{in}}}))[\mathtt{x} \mapsto \pi_k(\overline{\mathcal{A}'}[\![e^{\ell_0}]\!]([\![\mathtt{x} :=^\ell e^{\ell_0}]\!]_{\overline{\mathsf{in}}}))] \\
&\qquad \text{(by def. of } \overline{\mathcal{A}}) \\
&= \prod_{k\in\mathbb{K}} (\pi_k([\![\mathtt{x} :=^\ell e^{\ell_0}]\!]_{\overline{\mathsf{in}}}))[\mathtt{x} \mapsto \pi_k(\overline{\mathcal{A}'}[\![e^{\ell_0}]\!]([\![e^{\ell_0}]\!]_{\overline{\mathsf{in}}}))] \\
&\qquad \text{(by def. of } [\![e^{\ell_0}]\!]_{\overline{\mathsf{in}}}) \\
&= \prod_{k\in\mathbb{K}} (\pi_k([\![\mathtt{x} :=^\ell e^{\ell_0}]\!]_{\overline{\mathsf{in}}}))[\mathtt{x} \mapsto \pi_k([\![e^{\ell_0}]\!]_{\overline{\mathsf{out}}})] \\
&\qquad \text{(by first half of theorem)} \\
&= \prod_{k\in\mathbb{K}} \pi_k([\![\mathtt{x} :=^\ell e^{\ell_0}]\!]_{\overline{\mathsf{out}}}) \quad \text{(by def. of } [\![\mathtt{x} :=^\ell e^{\ell_0}]\!]_{\overline{\mathsf{out}}}) \\
&= [\![\mathtt{x} :=^\ell e^{\ell_0}]\!]_{\overline{\mathsf{out}}} \quad \text{(simplify)}
\end{aligned}$$

Case $s_0^{\ell_0} ;^\ell s_1^{\ell_1}$:

$$\begin{aligned}
&\overline{\mathcal{A}}[\![s_0^{\ell_0} ;^\ell s_1^{\ell_1}]\!]([\![s_0^{\ell_0} ;^\ell s_1^{\ell_1}]\!]_{\overline{\mathsf{in}}}) \\
&= (\overline{\mathcal{A}}[\![s_1^{\ell_1}]\!] \circ \overline{\mathcal{A}}[\![s_0^{\ell_0}]\!])([\![s_0^{\ell_0} ;^\ell s_1^{\ell_1}]\!]_{\overline{\mathsf{in}}}) \quad \text{(by def. of } \overline{\mathcal{A}}) \\
&= (\overline{\mathcal{A}}[\![s_1^{\ell_1}]\!] \circ \overline{\mathcal{A}}[\![s_0^{\ell_0}]\!])([\![s_0^{\ell_0}]\!]_{\overline{\mathsf{in}}}) \quad \text{(by def. of } [\![s_0^{\ell_0}]\!]_{\mathsf{in}}) \\
&= \overline{\mathcal{A}}[\![s_1^{\ell_1}]\!](\overline{\mathcal{A}}[\![s_0^{\ell_0}]\!][\![s_0^{\ell_0}]\!]_{\overline{\mathsf{in}}}) \quad \text{(by def. of } \circ) \\
&\ddot{\sqsubseteq} \overline{\mathcal{A}}[\![s_1^{\ell_1}]\!]([\![s_0^{\ell_0}]\!]_{\overline{\mathsf{out}}}) \quad \text{(by IH, } \overline{\mathcal{A}} \text{ monotone)} \\
&= \overline{\mathcal{A}}[\![s_1^{\ell_1}]\!]([\![s_1^{\ell_1}]\!]_{\overline{\mathsf{in}}}) \quad \text{(by def. of } [\![s_1^{\ell_1}]\!]_{\overline{\mathsf{in}}}) \\
&\ddot{\sqsubseteq} [\![s_1^{\ell_1}]\!]_{\overline{\mathsf{out}}} \quad \text{(by IH)} \\
&= [\![s_0^{\ell_0} ;^\ell s_1^{\ell_1}]\!]_{\overline{\mathsf{out}}} \quad \text{(by def. of } [\![s_0^{\ell_0} ;^\ell s_1^{\ell_1}]\!]_{\overline{\mathsf{out}}})
\end{aligned}$$

Case $\texttt{if}^{\ell}\ e\ \texttt{then}\ s_0^{\ell_0}\ \texttt{else}\ s_1^{\ell_1}$**:**

$$
\begin{aligned}
&\overline{\mathcal{A}}[\![\texttt{if}^{\ell}\ e\ \texttt{then}\ s_0^{\ell_0}\ \texttt{else}\ s_1^{\ell_1}]\!]([\![\texttt{if}^{\ell}\ e\ \texttt{then}\ s_0^{\ell_0}\ \texttt{else}\ s_1^{\ell_1}]\!]_{\overline{\mathsf{in}}}) \\
&= (\overline{\mathcal{A}}[\![s_0^{\ell_0}]\!] \ddot{\sqcup} \overline{\mathcal{A}}[\![s_1^{\ell_1}]\!])([\![\texttt{if}^{\ell}\ e\ \texttt{then}\ s_0^{\ell_0}\ \texttt{else}\ s_1^{\ell_1}]\!]_{\overline{\mathsf{in}}}) && \text{(by def. of } \overline{\mathcal{A}}) \\
&= \overline{\mathcal{A}}[\![s_0^{\ell_0}]\!]([\![\texttt{if}^{\ell}\ e\ \texttt{then}\ s_0^{\ell_0}\ \texttt{else}\ s_1^{\ell_1}]\!]_{\overline{\mathsf{in}}}) \\
&\quad \ddot{\sqcup}\ \overline{\mathcal{A}}[\![s_1^{\ell_1}]\!]([\![\texttt{if}^{\ell}\ e\ \texttt{then}\ s_0^{\ell_0}\ \texttt{else}\ s_1^{\ell_1}]\!]_{\overline{\mathsf{in}}}) && \text{(by def. of } \ddot{\sqcup}) \\
&= \overline{\mathcal{A}}[\![s_0^{\ell_0}]\!]([\![s_0^{\ell_0}]\!]_{\overline{\mathsf{in}}}) \ddot{\sqcup} \overline{\mathcal{A}}[\![s_1^{\ell_1}]\!]([\![s_1^{\ell_1}]\!]_{\overline{\mathsf{in}}}) && \text{(by def. of } [\![s_0^{\ell_0}]\!]_{\overline{\mathsf{in}}}, [\![s_1^{\ell_1}]\!]_{\overline{\mathsf{in}}}) \\
&\ddot{\sqsubseteq} [\![s_0^{\ell_0}]\!]_{\overline{\mathsf{out}}} \ddot{\sqcup} [\![s_1^{\ell_1}]\!]_{\overline{\mathsf{out}}} && \text{(by IH, twice)} \\
&= [\![\texttt{if}^{\ell}\ e\ \texttt{then}\ s_0^{\ell_0}\ \texttt{else}\ s_1^{\ell_1}]\!]_{\overline{\mathsf{out}}} && \text{(by def. of } [\![\texttt{if}^{\ell}\ e\ \texttt{then}\ s_0^{\ell_0}\ \texttt{else}\ s_1^{\ell_1}]\!]_{\overline{\mathsf{out}}})
\end{aligned}
$$

Case $\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}$**:** Recall the while equations:

$$[\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{out}}} = [\![s^{\ell_0}]\!]_{\overline{\mathsf{in}}} \qquad \text{(eq.1)}$$

$$[\![s^{\ell_0}]\!]_{\overline{\mathsf{in}}} = [\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}} \ddot{\sqcup} [\![s^{\ell_0}]\!]_{\overline{\mathsf{out}}} \qquad \text{(eq.2)}$$

We now prove by (inner) induction that for all $n \geqslant 0$

$$\overline{\mathfrak{F}}^n(\ddot{\bot})([\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}} \ddot{\sqcup} [\![s^{\ell_0}]\!]_{\overline{\mathsf{out}}}) \ddot{\sqsubseteq} [\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{out}}}$$

where $\overline{\mathfrak{F}} = \lambda\overline{\Phi}.\, \lambda\overline{a}.\, \overline{a} \ddot{\sqcup} \overline{\Phi}(\overline{\mathcal{A}}[\![s^{\ell_0}]\!]\overline{a})$ and $\ddot{\bot} = \lambda\overline{a}.\, \prod_{k\in\mathbb{K}} \dot{\bot}$. From here it follows that

$$
\begin{aligned}
&\overline{\mathcal{A}}[\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]([\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) \\
&= (\mathrm{lfp}\,\overline{\mathfrak{F}})([\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) && \text{(by def. of } \overline{\mathcal{A}}) \\
&= (\ddot{\sqcup}_i \overline{\mathfrak{F}}^i(\ddot{\bot}))([\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) && \text{(by Kleene's fixed point theorem)} \\
&= (\lambda\overline{a}.\, \ddot{\sqcup}_i \overline{\mathfrak{F}}^i(\ddot{\bot})\overline{a})([\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) && \text{(by def. of } \ddot{\sqcup}) \\
&= \ddot{\sqcup}_i \overline{\mathfrak{F}}^i(\ddot{\bot})[\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}} && (\beta\text{-reduction}) \\
&\ddot{\sqsubseteq} \ddot{\sqcup}_i \overline{\mathfrak{F}}^i(\ddot{\bot})([\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}} \ddot{\sqcup} [\![s^{\ell_0}]\!]_{\overline{\mathsf{out}}}) && \text{(by monotonicity of } \overline{\mathfrak{F}}^i(\ddot{\bot})) \\
&\ddot{\sqsubseteq} [\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{out}}} && \text{(by above)}
\end{aligned}
$$

Case $n = 0$**:**

$$
\begin{aligned}
&\overline{\mathfrak{F}}^0(\ddot{\bot})([\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}} \ddot{\sqcup} [\![s^{\ell_0}]\!]_{\overline{\mathsf{out}}}) \\
&= \ddot{\bot}([\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}} \ddot{\sqcup} [\![s^{\ell_0}]\!]_{\overline{\mathsf{out}}}) && \text{(by def. of } \overline{\mathfrak{F}}^0) \\
&= \prod_{k\in\mathbb{K}} \dot{\bot} && \text{(by def. of } \ddot{\bot}) \\
&\ddot{\sqsubseteq} [\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{out}}} && \text{(by def. of } \ddot{\sqsubseteq} \text{ and } \dot{\bot})
\end{aligned}
$$

Case $n = k + 1$**:**
Assume $\overline{\mathfrak{F}}^k(\ddot{\bot})([\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}} \ddot{\sqcup} [\![s^{\ell_0}]\!]_{\overline{\mathsf{out}}}) \ddot{\sqsubseteq} [\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{out}}}$.

We now reason as follows:

$$
\begin{aligned}
&\overline{\mathfrak{F}}^{k+1}(\ddot{\bot})([\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}} \ddot{\sqcup} [\![s^{\ell_0}]\!]_{\overline{\mathsf{out}}}) \\
&= \overline{\mathfrak{F}}^{k+1}(\ddot{\bot})([\![s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) && \text{(by eq.2)} \\
&= \overline{\mathfrak{F}}(\overline{\mathfrak{F}}^k(\ddot{\bot}))([\![s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) && \text{(by def. of } \overline{\mathfrak{F}}^{k+1}) \\
&= (\lambda\overline{\Phi}.\, \lambda\overline{a}.\, \overline{a} \ddot{\sqcup} \overline{\Phi}(\overline{\mathcal{A}}[\![s^{\ell_0}]\!]\overline{a}))(\overline{\mathfrak{F}}^k(\ddot{\bot}))([\![s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) && \text{(by def. of } \overline{\mathfrak{F}}) \\
&= (\lambda\overline{a}.\, \overline{a} \ddot{\sqcup} \overline{\mathfrak{F}}^k(\ddot{\bot})(\overline{\mathcal{A}}[\![s^{\ell_0}]\!]\overline{a}))([\![s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) && (\beta\text{-reduction}) \\
&= [\![s^{\ell_0}]\!]_{\overline{\mathsf{in}}} \ddot{\sqcup} \overline{\mathfrak{F}}^k(\ddot{\bot})(\overline{\mathcal{A}}[\![s^{\ell_0}]\!]([\![s^{\ell_0}]\!]_{\overline{\mathsf{in}}})) && (\beta\text{-reduction}) \\
&\ddot{\sqsubseteq} [\![s^{\ell_0}]\!]_{\overline{\mathsf{in}}} \ddot{\sqcup} \overline{\mathfrak{F}}^k(\ddot{\bot})([\![s^{\ell_0}]\!]_{\overline{\mathsf{out}}}) && \text{(by outer IH, monotonicity of } \overline{\mathfrak{F}}^k(\ddot{\bot})) \\
&\ddot{\sqsubseteq} [\![s^{\ell_0}]\!]_{\overline{\mathsf{in}}} \ddot{\sqcup} \overline{\mathfrak{F}}^k(\ddot{\bot})([\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}} \ddot{\sqcup} [\![s^{\ell_0}]\!]_{\overline{\mathsf{out}}}) && \text{(by monotonicity of } \overline{\mathfrak{F}}^k(\ddot{\bot})) \\
&\ddot{\sqsubseteq} [\![s^{\ell_0}]\!]_{\overline{\mathsf{in}}} \ddot{\sqcup} [\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{out}}} && \text{(by inner IH)} \\
&= [\![\texttt{while}^{\ell}\ e\ \texttt{do}\ s^{\ell_0}]\!]_{\overline{\mathsf{out}}} && \text{(by eq.1)}
\end{aligned}
$$

Case $\texttt{\#if}^{\ell}\ \varphi\ s^{\ell_0}$**:**

$$
\begin{aligned}
&\overline{\mathcal{A}}[\![\texttt{\#if}^{\ell}\ \varphi\ s^{\ell_0}]\!]([\![\texttt{\#if}^{\ell}\ \varphi\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) \\
&= (\lambda\overline{a}.\, \prod_{k\in\mathbb{K}} \begin{cases} \pi_k(\overline{\mathcal{A}}[\![s^{\ell_0}]\!]\,\overline{a}) & k \vDash \varphi \\ \pi_k(\overline{a}) & k \nvDash \varphi \end{cases})([\![\texttt{\#if}^{\ell}\ \varphi\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) \\
& && \text{(by def. of } \overline{\mathcal{A}}) \\
&= \prod_{k\in\mathbb{K}} \begin{cases} \pi_k(\overline{\mathcal{A}}[\![s^{\ell_0}]\!]\,[\![\texttt{\#if}^{\ell}\ \varphi\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) & k \vDash \varphi \\ \pi_k([\![\texttt{\#if}^{\ell}\ \varphi\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) & k \nvDash \varphi \end{cases} \\
& && (\beta\text{-reduction}) \\
&= \prod_{k\in\mathbb{K}} \begin{cases} \pi_k(\overline{\mathcal{A}}[\![s^{\ell_0}]\!]\,[\![s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) & k \vDash \varphi \\ \pi_k([\![\texttt{\#if}^{\ell}\ \varphi\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) & k \nvDash \varphi \end{cases} \\
& && \text{(by guarded def. of } [\![s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) \\
&\ddot{\sqsubseteq} \prod_{k\in\mathbb{K}} \begin{cases} \pi_k([\![s^{\ell_0}]\!]_{\overline{\mathsf{out}}}) & k \vDash \varphi \\ \pi_k([\![\texttt{\#if}^{\ell}\ \varphi\ s^{\ell_0}]\!]_{\overline{\mathsf{in}}}) & k \nvDash \varphi \end{cases} && \text{(by IH)} \\
&= \prod_{k\in\mathbb{K}} \begin{cases} \pi_k([\![\texttt{\#if}^{\ell}\ \varphi\ s^{\ell_0}]\!]_{\overline{\mathsf{out}}}) & k \vDash \varphi \\ \pi_k([\![\texttt{\#if}^{\ell}\ \varphi\ s^{\ell_0}]\!]_{\overline{\mathsf{out}}}) & k \nvDash \varphi \end{cases} \\
& && \text{(by def. of } [\![\texttt{\#if}^{\ell}\ \varphi\ s^{\ell_0}]\!]_{\overline{\mathsf{out}}}) \\
&= \prod_{k\in\mathbb{K}} \pi_k([\![\texttt{\#if}^{\ell}\ \varphi\ s^{\ell_0}]\!]_{\overline{\mathsf{out}}}) && \text{(simplify)} \\
&= [\![\texttt{\#if}^{\ell}\ \varphi\ s^{\ell_0}]\!]_{\overline{\mathsf{out}}} && \text{(simplify)}
\end{aligned}
$$

□

O. A Generic Soundness Proof

Proof of Thm. 8. Assume for all $s \in \mathit{Stm}$: $\alpha \circ \mathcal{Y}[\![s]\!] \circ \gamma \mathrel{\dot{\sqsubseteq}} \mathcal{X}[\![s]\!]$. Let $\overline{s}$ be given.

$$
\begin{aligned}
&\mathit{lift}(\alpha) \circ \mathit{lift}(\mathcal{Y})[\![\overline{s}]\!] \circ \mathit{lift}(\gamma) \\
&= \lambda \overline{y}.\, (\mathit{lift}(\alpha) \circ \mathit{lift}(\mathcal{Y})[\![\overline{s}]\!] \circ \mathit{lift}(\gamma))(\overline{y}) && (\eta\text{-expansion}) \\
&= \lambda \overline{y}.\, \mathit{lift}(\alpha)(\mathit{lift}(\mathcal{Y})[\![\overline{s}]\!](\mathit{lift}(\gamma)(\overline{y}))) && (\text{by def. of } \circ) \\
&= \lambda \overline{y}.\, \mathit{lift}(\alpha) \prod_{k \in \mathbb{K}} \mathcal{Y}[\![P[\![\overline{s}]\!]_k]\!](\pi_k(\mathit{lift}(\gamma)(\overline{y}))) \\
& && (\text{by def. of } \mathit{lift}(\mathcal{Y})) \\
&= \lambda \overline{y}.\, \mathit{lift}(\alpha) \prod_{k \in \mathbb{K}} \mathcal{Y}[\![P[\![\overline{s}]\!]_k]\!](\pi_k(\prod_{k' \in \mathbb{K}} \gamma(\pi_{k'}(\overline{y})))) \\
& && (\text{by def. of } \mathit{lift}(\gamma)) \\
&= \lambda \overline{y}.\, \mathit{lift}(\alpha) \prod_{k \in \mathbb{K}} \mathcal{Y}[\![P[\![\overline{s}]\!]_k]\!](\gamma(\pi_k(\overline{y}))) && (\text{by def. of } \pi_k) \\
&= \lambda \overline{y}.\, \prod_{k \in \mathbb{K}} \alpha(\mathcal{Y}[\![P[\![\overline{s}]\!]_k]\!](\gamma(\pi_k(\overline{y})))) && (\text{by def. of } \mathit{lift}(\alpha)) \\
&\mathrel{\ddot{\sqsubseteq}} \lambda \overline{y}.\, \prod_{k \in \mathbb{K}} \mathcal{X}[\![P[\![\overline{s}]\!]_k]\!](\pi_k(\overline{y})) && (\text{by assumption}) \\
&= \mathit{lift}(\mathcal{X})[\![\overline{s}]\!] && (\text{by def. of } \mathit{lift})
\end{aligned}
$$

□

P. Abstracting Variability: The Galois Connection Proof

Theorem 38. $\langle \overline{\mathbb{X}}, \dot{\sqsubseteq} \rangle \underset{\alpha_F}{\overset{\gamma_F}{\leftrightarrows}} \langle \overline{\mathbb{X}}_F, \dot{\sqsubseteq} \rangle$

Proof. α_F **is monotone:** assume $\overline{x} \mathrel{\dot{\sqsubseteq}} \overline{x}'$

$$
\begin{aligned}
&\alpha_F(\overline{x}) \\
&= \prod_{k_F \in \mathbb{K}_F} \bigsqcup_{\{k \in \mathbb{K} \mid k_F = k \cap F\}} \pi_k(\overline{x}) && \text{(by def. of } \alpha_F\text{)} \\
&\mathrel{\dot{\sqsubseteq}} \prod_{k_F \in \mathbb{K}_F} \bigsqcup_{\{k \in \mathbb{K} \mid k_F = k \cap F\}} \pi_k(\overline{x}') && \text{(by assumption)} \\
&= \alpha_F(\overline{x}') && \text{(by def. of } \alpha_F\text{)}
\end{aligned}
$$

γ_F **is monotone:** assume $\overline{x}_F \mathrel{\dot{\sqsubseteq}} \overline{x}'_F$

$$
\begin{aligned}
&\gamma_F(\overline{x}_F) \\
&= \prod_{k \in \mathbb{K}} \pi_{(k \cap F)}(\overline{x}_F) && \text{(by def. of } \gamma_F\text{)} \\
&\mathrel{\dot{\sqsubseteq}} \prod_{k \in \mathbb{K}} \pi_{(k \cap F)}(\overline{x}'_F) && \text{(by assumption)} \\
&= \gamma_F(\overline{x}'_F) && \text{(by def. of } \gamma_F\text{)}
\end{aligned}
$$

$\gamma_F \circ \alpha_F$ **extensive:**

$$
\begin{aligned}
&\gamma_F(\alpha_F(\overline{x})) \\
&= \prod_{k \in \mathbb{K}} \pi_{(k \cap F)}(\alpha_F(\overline{x})) && \text{(by def. of } \gamma_F\text{)} \\
&= \prod_{k \in \mathbb{K}} \pi_{(k \cap F)}\Big(\prod_{k_F \in \mathbb{K}_F} \bigsqcup_{\{k' \in \mathbb{K} \mid k_F = k' \cap F\}} \pi_{k'}(\overline{x})\Big) \\
& && \text{(by def. of } \alpha_F\text{)} \\
&= \prod_{k \in \mathbb{K}} \bigsqcup_{\{k' \in \mathbb{K} \mid k \cap F = k' \cap F\}} \pi_{k'}(\overline{x}) && \text{(by def. of } \pi_{(k \cap F)}\text{)} \\
&\mathrel{\dot{\sqsupseteq}} \prod_{k \in \mathbb{K}} \pi_k(\overline{x}) && \text{(since } k \cap F = k \cap F\text{)} \\
&= \overline{x} && \text{(simplify)}
\end{aligned}
$$

$\alpha_F \circ \gamma_F$ **reductive:**

$$
\begin{aligned}
&\alpha_F(\gamma_F(\overline{x}_F)) \\
&= \prod_{k_F \in \mathbb{K}_F} \bigsqcup_{\{k \in \mathbb{K} \mid k_F = k \cap F\}} \pi_k(\gamma_F(\overline{x}_F)) && \text{(by def. of } \alpha_F\text{)} \\
&= \prod_{k_F \in \mathbb{K}_F} \bigsqcup_{\{k \in \mathbb{K} \mid k_F = k \cap F\}} \pi_k(\textstyle\prod_{k' \in \mathbb{K}} \pi_{(k' \cap F)}(\overline{x}_F)) \\
& && \text{(by def. of } \gamma_F\text{)} \\
&= \prod_{k_F \in \mathbb{K}_F} \bigsqcup_{\{k \in \mathbb{K} \mid k_F = k \cap F\}} \pi_{(k \cap F)}(\overline{x}_F) && \text{(by def. of } \pi_k\text{)} \\
&= \prod_{k_F \in \mathbb{K}_F} \bigsqcup_{\{k \in \mathbb{K} \mid k_F = k \cap F\}} \pi_{k_F}(\overline{x}_F) && \text{(since } k_F = k \cap F\text{)} \\
&= \prod_{k_F \in \mathbb{K}_F} \pi_{k_F}(\overline{x}_F) && \text{(simplify)} \\
&= \overline{x}_F && \text{(simplify)}
\end{aligned}
$$

□