MiniSail: A core calculus for Sail

Contents

1 Introduction

Giving Sail a formal specification and proving safety and decidability will provide us with a contract between users of the language and builders of an implementation of the language. It will ensure that changes to the language are carefully controlled and that we do not rely on a blessed implementation for a complete reference of the language and clever engineering for a type checker that works 'most of the time'. With this in mind, we have developed MiniSail a core calculus that captures key features of Sail and is a first step toward formalising the complete language. In this work we present the syntax of MiniSail, a type system and operational semantics. We then prove a number of lemmas leading to proofs of preservation, progress and safety. MiniSail, particularly the use of bidirectional typing [2], the use of constraints and the interaction with the SMT solver, has already informed the design of the Sail type system

We choose the features of Sail to include in MiniSail to be those that interact highly with other features, those that are involved in the generation of SMT constraints and those that pose interesting challenges for features satisfying the first two criteria. The features included are:

- Lightweight dependent types. This cuts across many of the other language features. It permits immutable program variables to appear in types and means that reduction steps making use of substitution, and hence the substitution lemmas, need to take into account substitution into types and the type context.
- Refinement types with constraints that are limited to expressions within the decidable logic of the SMT solver Z3 [1].
- Simple arithmetic expressions. These give us points where constraints are synthesised and gives rise to arithmetic expressions in our constraint logic.
- Mutable variables. These require us to have an operational semantics that includes a mutable store, a guarantee that values in the store preserve their types and a substitution function substituting into the declared type for a mutable variable.
- While loops.
- Pairs and their associated typing syntax.
- Generalised sum types (corresponding to union types in Sail).
- Function definition and function application. Both of these require a subtyping check. For function definition, we need a subtype check that the value returned from the function body is a subtype of the return type given in the function signature. For function application we need a subtype check that the provided function argument is a subtype of the argument's type as given in the function's type signature.

As MiniSail is intended to be a calculus rather than a programming language, it has been designed for mathematical rather than programming convenience. To achieve mathematical convenience, the syntax of MiniSail is based on let-normal form and a stratification of what are expressions in the Sail language into values, expressions and statements in MiniSail. This leads to a corresponding stratification of the typing judgements and makes reasoning about the type system clearer. Expressions are function applications, binary operations $+$ and $\leq$ and projection from a pair but the arguments to these can only be values. Hence complex expressions, such as function application to a sum of two variables, need to be broken into a nesting of let-statements where the parts of the complex expression are bound to variables. This ensures that the types of the components of complex expressions are made explicit as the typing of the nested let-statements, possibly nested, will give us the type of each sub-expression.

We make use of bidirectional typing for defining the type system with the main typing rules for values and expressions being type synthesis and the typing rules for statements being type checking. This provides clarity on where any type annotations are necesary in the language and also aids implementation of a type checker.

Some of the features in Sail that are missing from MiniSail are:

- Vectors, including bit vectors.
- Complex pattern matching including bit vector pattern matching
- Records and mutable records.
- Overloaded functions.

The complete grammar, type system and specification of the operational semantics can be found in the Appendix.

2 Decidability of Type Checking

Both Sail and MiniSail draw inspriration for their type systems from Liquid Types [3]. Syntactically, types in MiniSail are composed of a program variable, a base type and a refinement predicate. Unlike other refinement type languages, Sail and MiniSail do not include higher order functions and so we are able to provide a strong argument that the constraints arising during MiniSail type checking, and that are passed to the SMT solver, form a logic that is decidable within the solver. We can go further and claim that the MiniSail type system is decidable - there is an algorithm for type checking a program that will terminate with the correct answer. We sketch an algorithm and indicate which features of the language are important for decidability:

- To each type checking judgement corresponds a function taking contexts, a term and a type. It returns a boolean indicating that the term has the type with respect to the supplied contexts.
- To each type synthesis judgement corresponds a function taking contexts and a term. It returns a type.
- In each of the checking and synthesis functions, we have a match statement that matches on the syntactic form of the term. As our typing rules are syntax directed we can put into each branch the logic for a single inductive rule and all of the rules for a judgement will appear in exactly one branch.
- In each branch we check the premises for the rule the branch corresponds to. The checks are as follows:
  - Lookup in a context. These are decidable since the contexts are finite.
  - Invoking a check or synthesis typing function. As these are invoked with a simpler term, the recursion will bottom out.
  - Subtype check.
- The premises of some rules will reference 'existential variables' by this we mean that these are variables appearing in the premises but not in the conclusion of a rule. All of the existential variables in the MiniSail rules can be calculated deterministically. For example, in the rule for INFER_V_DATA_CONS we have a τ, C_i, τ_i that appear just in the premises however the choice of values for these is determined by the C in the rule conclusion and that C can only appear in one union definition by our well-scoping conditions.

- The subtyping check can be implemented as a function to construct an SMT problem derived from the constraints for each of the types being checked. The details of this are covered in Section 7.3. We have ensured that the constraints generated are from a logic fragment that is decidable within the SMT solver. We make use of the follow logics that are decidable in Z3:
  - Presburger arithmetic with inequality
  - Algebraic data types

3 Syntax

In the syntax description below, we use the following metavariables:

- n - Numeric literals. We use $\bar{n}$ to refer to the actual natural number corresponding to the literal n.
- x - Program variables and variables bound in monotypes. Also used are z and w.
- u - Mutable variables.
- f - Function names.
- tid - Type identifiers.
- C - Datatype constructors.

We stratify the terms of the language into values, expressions and statements: Values are the things that we want our program to produce when provided with an empty context, expressions are how we combine values and statements give the principal structure to our programs. Values, expressions and statements are defined by the following grammar:

$$\begin{array}{lll}
\text{Value} & v & ::= \; x \mid n \mid \mathbf{T} \mid \mathbf{F} \mid (v,v) \mid \mathrm{C}\ v \mid () \\
\text{Expression} & e & ::= \; v \mid v+v \mid v \leq v \mid f\ v \mid u \mid \mathrm{fst}\ v \mid \mathrm{snd}\ v \\
\text{Statement} & s & ::= \; v \mid \mathbf{let}\ x = e\ \mathbf{in}\ s \mid \mathbf{let}\ x : \tau = s\ \mathbf{in}\ s \\
 & & \quad \mid \mathbf{if}\ v\ \mathbf{then}\ s\ \mathbf{else}\ s \mid \\
 & & \quad \mid \mathbf{match}\ v\ \{\ \mathrm{C}_1\ x_1 \Rightarrow s_1\ ,\ ..\ ,\ \mathrm{C}_n\ x_n \Rightarrow s_n \} \\
 & & \quad \mid \mathbf{var}\ u\ : \tau = v\ \mathbf{in}\ s \\
 & & \quad \mid u := v \\
 & & \quad \mid \mathbf{while}\ (e)\ \ \mathbf{do}\ \{s\} \\
 & & \quad \mid s_1 ; s_2
\end{array}$$

- Expressions only occur in let and while statements. During reduction, let is where the reduction of expressions occur and reduction of while involves unfolding it to a let-if-while construct with the let containing the expression. Using expressions in the guard of the while loop allows us to reference mutable variables in the guard and for these variables to change value as the while body is processed. Any other occurrence of an expression, for example function application to an expression, can be handled by writing as let.
- Mutable variables are expressions and the scope of a mutable variable is the statement in the var statement.
- The grammar for the calculus omits bracketing with round brackets or curly brackets; these would be included in a grammer for a MiniSail programming language.

Typing constructs are as follows:

Base type	b	::=	**int** | **bool** | tid | b × b
Constraint	ϕ	::=	**T**| **F**| $e^- = e^-$ | $e^- \leq e^-$ | $\phi \wedge \phi$ | $\phi \vee \phi$ | $\neg \phi$
Monotype	τ	::=	$\{x : b \mid \phi\}$

The nonterminal e^- is identical to the expression nonterminal e with the omission of mutable variables and function application. e^- defines what is permitted in the logic of SMT solver such as Z3. Note that we are including pairing, pair projection and data-constructors in the logic we pass to Z3. These are supported in Z3.

Constraints can include program variables and so well-scoping conditions are needed to ensure that free variables in constraints are either bound in the monotype form $\{z : b|\phi\}$ (as z is here) or are in the domain of the typing context. This is detailed below.

The top level structure of a program is composed of a sequence of type and function definitions and a single statement:

Function Definition	fd	::=	**val** $f : (x : b[\phi]) \to \tau$
			function $f(x) = s$
Union Definition	td	::=	**union** tid = $\{\mathrm{C}_1 : \tau_1, ..., \mathrm{C}_n : \tau_n\}$
Definition	def	::=	td | fd
Program	p	::=	def_1 ; .. ; def_n ; s

Note that function definition is a combination of a type signature for the function and a function body definition. In Sail these are two separate terms in the language.

We use the word ‘term’ to refer loosely to any syntactic construct.

4 Relationship to Sail

Apart from types, MiniSail is a sub-language of Sail. Types in Sail are constructed a little differently from MiniSail types and in this section we define a mechanism for converting Sail types to MiniSail types. Types in Sail are polymorphic in number, order and other types and constraints making use of type level number expressions can be included in a type. In this respect, Sail types resemble refinement types as described in [3, 4]. In refinement types, there is a value variable, a base type and a constraint making use of the value variable and program variables that are in scope. In Sail, constraints can be a separate component of the type but can also be implicitly combined with the base in a type constructor such as range. Here the base type is **int** and the constraint is that the value variable is between two type level number expressions. Furthermore, types in Sail can be nested - for example a pair type is a nesting of two types. In MiniSail, this doesn’t occur - a pair type of two types is indicated by a single top level mono-type with a paired base type and suitable constraint. Finally, a type bind contain zero or more type level variables; in MiniSail they bind exactly on value variable. We thus need to mechanism that extracts the base type and constraint components from a Sail type and tells us how to map type level variables to value variable terms.

Types in Sail can be specified in a number of different ways and one of these is as an existential type. Prior to converting to a MiniSail type, we require that the Sail type is rewritten as a Sail existential type without any nesting of existentials. More formally, we define the following subgrammar of the Sail grammar and we require that the Sail type match this grammar. In the grammar we have that n is a numeric literal and k is a Sail type level integer variable.

Value	v	$::=$	$k \mid n \mid$
Expression	e	$::=$	$v \mid v + v \mid v \leq v$
Constraint	ϕ	$::=$	$\mathbf{T} \mid \mathbf{F} \mid e = e \mid e \leq e \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi$
Simple Type	σ	$::=$	$\text{atom}(k) \mid \text{range}(n,n) \mid (\sigma,\sigma) \mid \text{tid}$
Type	τ	$::=$	$\{k_1..k_n, \phi.\sigma\} \mid \sigma$
Function Type	ψ	$::=$	$\text{forall } k_1..k_n, \phi.\sigma \rightarrow \tau$

As part of the conversion we will construct a mapping, ρ, that is a partial map from Sail type level variables to MiniSail values. We define a series of partial functions that carry out the conversion and Sail types not in the domain are not convertible.

We define a partial function that takes a τ and returns a MiniSail base type.

$$\begin{aligned}
B(\text{atom}(k)) &= \mathbf{int} \\
B(\text{range}(n_1, n_2)) &= \mathbf{int} \\
B((\tau_1, \tau_2)) &= B(\tau_1) \times B(\tau_2) \\
B(tid) &= tid \\
B(\{k_1, ...k_n, \phi.\sigma\}) &= B(\sigma)
\end{aligned}$$

Next we define a function taking a MiniSail value, a mapping ρ and a τ and returns a constraint and a mapping.

$$\begin{aligned}
N(v, \rho, \text{atom}(k)) &= \begin{cases} (v = \rho(k), \rho), & \text{if } k \in dom(\rho) \\ (\top, \rho[k \mapsto v]), & \text{otherwise} \end{cases} \\
N(v, \rho, \text{range}(c_1, c_2)) &= (n_1 \leq v \wedge v \leq n_2, \rho) \\
N(v, \rho, (\tau_1, \tau_2)) &= \text{let } (\phi_1, \rho_1) = N(\mathbf{fst}\ v, \rho, \tau_1) \text{ in} \\
&\quad\ \text{let } (\phi_1, \rho_1) = N(\mathbf{snd}\ v, \rho_1, \tau_2) \text{ in} \\
&\quad\ (\phi_1 \wedge \phi_2, \rho_2) \\
N(v, \rho, \text{ tid}) &= (\top, \rho) \\
N(v, \rho, \{k_1, ...k_n, \phi.\sigma\}) &= \{z : B(\sigma) \mid \text{let } (\phi', \rho') = N(z, \rho, \sigma) \text{ in } \phi' \wedge \rho'(\phi)\}
\end{aligned}$$

We write $\rho[k \mapsto t]$ to indicate ρ extended with a mapping of k to v. The function that performs the top level conversion is:

$$\begin{aligned}
M(\sigma) &= \text{let } (\phi, \rho) = N(z, \cdot, \sigma) \text{ in } \{z : B(\sigma) | \phi\} \\
M(\{k_1, ...k_n, \phi.\sigma\}) &= N(z, \cdot, \{k_1, ...k_n, \phi.\sigma\}) \\
M(\text{forall } k_1..k_n, \phi.\sigma \rightarrow \tau) &= \text{let } (\phi_1, \rho_1) = N(x, \cdot, \sigma) \text{ in} \\
&\quad\ \text{let } (\phi_2, \rho_2) = N(z, \rho_1, \tau) \text{ in} \\
&\quad\ x : B(\sigma)[\phi_1 \wedge \rho_1(\phi)] \rightarrow \{z : B(\tau) | \phi_2\}
\end{aligned}$$

We restrict the mapping to range types that only have numeric literals as their end points and atom types that only include a type level variable. Sail permits a wide range of arithmetic expressions in these positions however most of these can be rewritten by putting the expression into the constraint.

Examples of translation are:

$$\begin{aligned}
M(\{n, 10 \leq n.\ \text{int } (n)\}) &= \{z : \mathbf{int} | 10 \leq z\} \\
M(\{n, 42 \leq n.(\text{ atom } (n), \text{atom}(n))\}) &= \{z : \mathbf{int} \times \mathbf{int} | 42 \leq\ \mathbf{fst}\ z \wedge \mathbf{fst}\ z = \mathbf{snd}\ z\} \\
M(range(0, 31)) &= \{z : \mathbf{int} | 0 \leq z \wedge z \leq 31\} \\
M(\{n, 4 \leq n.\ (\text{range}(0, 10), \text{atom } (n))\})) &= \{z : \mathbf{int} * \mathbf{int} | 0 \leq \mathbf{fst}\ z \wedge \mathbf{fst}\ z \leq 10 \wedge 4 \leq \mathbf{snd}\ z\}
\end{aligned}$$

An example of a function type translation is:

$$\begin{aligned}&M(\text{forall } n\ m.\{p, \top.(\text{atom}(n), \text{atom}(m))\} \rightarrow \{k,\ k = m + n.\ atom(k)\})\\ &\qquad = x : \textbf{int}\ \times\ \textbf{int}\ [\top] \rightarrow \{z : \textbf{int} | z =\ \textbf{fst}\ x +\ \textbf{snd}\ x)\}\end{aligned}$$

The following type will not be translated:

$$M(\{n, 10 \leq n.\ \text{int}\ (n + 1)\})$$

but this following equivalent type will be:

$$M(\{n, 11 \leq n.\ \text{int}\ (n)\}) \quad = \quad \{z : \textbf{int}\ | 11 \leq z\}$$

The following type will not be translated:

$$\text{forall } n\ m\ k,\ k = m + n.\ \{(\text{atom}(n), \text{atom}(m))\} \rightarrow \{atom(k)\}$$

5 Example Programs

We present three short programs that illustrate a number of language features rather than being a realistic example of how Sail is used to model ISAs. The first program is written in 'idiomatic' Sail making use of a number of short-hands, the second is an equivalent Sail program, de-sugared somewhat to match the MiniSail grammer and the third is the corresponding MiniSail program that has had the type translation described above applied to it.

```
default Order dec

$include <prelude.sail>
$include <flow.sail>

union foo = {
  Bar : int(42),
  Baz : int
}

val f : foo -> int(42)
function f ( x ) = {
  y : int(42) = 42;
  foreach(i from 1 to 10) {
    if i==9 then
      match x {
        Bar(j) => y = j,
        Baz(j) => y = 42
      }
  };
  y
}
```

The following is the de-sugared version and is still valid Sail. The typing constructs that we need to convert are indicated with a grey background.

```
  default Order dec

  $include <prelude.sail>
```

```
$include <flow.sail>

union foo = {
  Bar : { 'n , 'n = 42 . int('n) },
  Baz : { 'n . int('n) }
}

val h : foo -> { 'm , 'm = 42 . int('m) }
function h ( y ) = {
  var w : foo  = Baz (1) in
  var u : { 'm , 'm = 42 . int('m) }  = 42 in
  var i : { 'm . int('m) } = 1 in
  while ( i <= 10 ) do {
    let x1 = i + 1 in i = x1;
    if i == 9 then
      w = Bar(42)
    else
      w = Baz(i);

    match w {
      Bar(j) => u = j,
      Baz(j) => u = 42
    }
  };
  u
}
```

The third is the MiniSail version that has undergone translation of the types:

```
union foo = {
  Bar : { z : int | z = 42 },
  Baz : { z : int | T }
 }

val h : x : foo [T] -> { z : int | z = 42 }
function h ( y ) = {
  var w : foo = Baz (1) in
  var u : { 'm , 'm = 42 . int('m) }  = 42 in
  var i : { 'm . int('m) } = 1 in
  while ( i <= 10 ) do {
    let x1 = i + 1 in i = x1;
    if i == 9 then
      w = Bar(42)
    else
      w = Baz(i);

    match w {
      Bar(j) => u = j,
      Baz(j) => u = 42
   }
  };
  u
}
```

6 Substitution

Substitution is the key mechanism used to execute MiniSail program and is used extensively in the typing rules. We are careful to only allow substitution of immutable variables by values. We use $\_[v/x]$ to denote the substitution of variables appearing in terms, constraints and types with values. Substitution associates to the left and distributes over connectives. So the following equivalence holds:

$$(\phi \Longrightarrow \phi')[v'/z][v/x] \equiv ((\phi[v'/z])[v/x]) \Longrightarrow ((\phi'[v'/z])[v/x])$$

We now define substitution for each sort of term in the language. In the following, the notation $x \# v$ means that x does not occur in v this is to ensure capture avoiding substitution and renaming of variables is permitted to ensure this occurs.
Substitution in values:

$$\begin{aligned}
x[v/z] &= \begin{cases} v, \text{if } x = z \\ x, \text{if } x \neq z \end{cases}\\
n[v/z] &= n\\
\mathbf{T}[v/z] &= \mathbf{T}\\
\mathbf{F}[v/z] &= \mathbf{F}\\
()[v/z] &= ()\\
(v_1, v_2)[v/z] &= (v_1[v/z], v_2[v/z])\\
(C\ v')[v/z] &= C\ (v'[v/z])
\end{aligned}$$

Substitution in expressions:

$$\begin{aligned}
(v_1 + v_2)[v/z] &= v_1[v/z] + v_2[v/z]\\
(v_1 \leq v_2)[v/z] &= v_1[v/z] \leq v_2[v/z]\\
(f\ v')[v/z] &= f(v'[v/z])\\
(\text{fst}\ v')[v/z] &= \text{fst}\ (v'[v/z])\\
(\text{snd}\ v')[v/z] &= \text{snd}\ (v'[v/z])\\
u[v/z] &= u
\end{aligned}$$

Substitution in statements. Where a statement includes a type annotation, substitution needs to occur in the type as well.

$$\begin{aligned}
(\mathbf{let}\ x = e\ \mathbf{in}\ s)[v/z] &= \mathbf{let}\ x = e[v/z]\ \mathbf{in}\ s[v/z]\\
&\ (\text{provided } x \neq z \text{ and } x\#v)\\
(\mathbf{if}\ v_1\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2)[v/z] &= \mathbf{if}\ v_1[v/z]\ \mathbf{then}\ s_1[v/z]\ \mathbf{else}\ s_2[v/z]\\
(\mathbf{let}\ x : \tau = s_1\ \mathbf{in}\ s_2)[v/z] &= \mathbf{let}\ x : \tau[v/z] = s_1[v/z]\ \mathbf{in}\ s_2[v/z]\\
&\ (\text{provided } x \neq z \text{ and } x\#v)\\
(u := v')[v/z] &= u := v'[v/z]\\
(\mathbf{var}\ u\ : \tau = v'\ \mathbf{in}\ s)[v/z] &= \mathbf{var}\ u\ : \tau[v/z] = v'[v/z]\ \mathbf{in}\ s[v/z]\\
(\mathbf{while}\ (e)\ \ \mathbf{do}\ \{s\})[v/z] &= \mathbf{while}\ (e[v/z])\ \ \mathbf{do}\ \{s[v/z]\}
\end{aligned}$$

$$\begin{aligned}
(\mathbf{match}\ v'\ \{\ \mathrm{C}_1\ x_1 \Rightarrow s_1\ ,\ ..\ ,\ \mathrm{C}_n\ x_n \Rightarrow s_n\})[v/z] =\\
\mathbf{match}\ (v'[v/z])\ \{\ \mathrm{C}_1\ x_1 \Rightarrow (s_1[v/z])\ ,\ ..\ ,\ \mathrm{C}_n\ x_n \Rightarrow (s_n[v/z])\}\\
(\text{provided } z \neq x_i \text{ and } x_i\#v,\ \text{ for } i = 1..n)
\end{aligned}$$

Substitution in constraints:

$$\begin{array}{rcl}
\mathbf{T}[v/z] & = & \mathbf{T} \\
\mathbf{F}[v/z] & = & \mathbf{F} \\
(e_1 = e_2)[v/x] & = & e_1[v/x] = e_2[v/x] \\
(e_1 \leq e_2)[v/x] & = & e_1[v/x] \leq e_2[v/x] \\
(\phi_1 \wedge \phi_2)[v/x] & = & \phi_1[v/x] \wedge \phi_2[v/x] \\
(\phi_1 \vee \phi_2)[v/x] & = & \phi_1[v/x] \vee \phi_2[v/x] \\
(\phi_1 \implies \phi_2)[v/x] & = & \phi_1[v/x] \implies \phi_2[v/x] \\
(\neg\phi)[v/x] & = & \neg(\phi[v/x])
\end{array}$$

Substitution in monotypes:

$$(\{z : b|\phi\})[v/x] \quad = \quad \{z : b|\phi[v/x]\} \text{ provided } x \neq z \text{ and } z\#v$$

7 Type System

7.1 Contexts

We have a type context, Γ, for immutable variables defined as follows:

$$\Gamma \quad = \quad x_1 : b_1[\phi_1], .., x_n : b_n[\phi_n]$$

New variables are added to the right. We also have a definition context Π that contains the list of functions and type definitions. As in Sail, we do not allow the same string to be used as a function identifier, type identifier, constructor or variable name, hence Π can contain both function and type definitions without conflict. We have a context for mutable variables, Δ. These three contexts can be viewed as functions with the domain of Γ being immutable program variables, domain of Π being function names and type identifiers and the domain of Δ being mutable variable names.

$$\begin{array}{rcl}
\Gamma(x_i) & = & b_i[\phi_i] \\
\Pi(f_i) & = & \textbf{val } f_i : (x_i : b_i[\phi_i]) \to \tau_i;\ \textbf{function } f_i(x_i) = s_i \\
\Pi(\mathrm{tid}_i) & = & \mathrm{tid}_i : \{\mathrm{C}_{i1} \ : \ \tau_{i1} \ .. \ \mathrm{C}_{in} \ : \ \tau_{in}\} \\
\Delta(u_i) & = & \tau_i
\end{array}$$

It is important to note that the context Γ has an ordering since the constraint in the type of a variable might make use variables introduced earlier in the context and so we need these latter variables to appear in the list prior to the variable being defined. We use well-scoped rules to capture this formally and these can be found in the appendix.

We do not include the side condition $x \notin vars(\Gamma)$ in the G_WS_CONS rule. Instead, globally, we adopt the Barendregt Convention where we apply alpha-conversion as needed to ensure that capture avoiding substitution is possible and a variable doesn't appear twice in the context. With the latter, the renaming will occur when a new variable and type is added to the context as in the CHECK_ANF_LET and CHECK_ANF_LET2 rules below: we rename the variable x bound in the *let* statemewnt if it already appears in the context.

$\boxed{\Pi \vdash \Gamma\, \mathbf{ws}}$ Context Γ is wellformed in context Π

$$\frac{\Pi\, \mathbf{ws}}{\Pi \vdash \cdot\, \mathbf{ws}} \quad \text{G\_WS\_EMPTY}$$

$$\frac{\begin{array}{l}\Pi \vdash \Gamma\,\mathbf{ws}\\ \Pi \vdash b\,\mathbf{ws}\\ \Pi;\Gamma, x : b[\top] \vdash \phi\,\mathbf{ws}\\ x \notin \mathrm{dom}(\Gamma)\end{array}}{\Pi \vdash \Gamma, x : b[\phi]\,\mathbf{ws}}\ \textsc{g\_ws\_cons}$$

We also define a substitution operation on the context Γ. If Γ is as above then:

$$\Gamma[v/x] = x_1 : b_1[\phi_1[v/x]], ..., x_n : b_n[\phi_n[v/x]]$$

also in the case where x appears in the context we have:

$$(\Gamma, x : b[\phi], \Gamma')[v/x] = \Gamma, \Gamma'[v/x]$$

We do need define substitution on Δ:

$$\{(u_1, \tau_1), .., (u_n, \tau_n)\}[v/x] = \{(u_1, \tau_1[v/x]), .., (u_n, \tau_n[v/x])\}$$

We do not need to apply substitution to the contexts Π.

7.2 Bidirectional Typing Rules

In this section we introduce and describe the key typing judgements and their inductive definitions. The first of these is the subtyping judgement. This provides the link between the typing rules and the SMT solver through the validity judgement $\Pi; \Gamma \models \phi$ which we discuss below.

$\boxed{\Pi; \Gamma \vdash \tau_1 \lesssim \tau_2}$

$$\frac{\begin{array}{l}\Pi; \Gamma \vdash \{z_1 : b|\phi_1\}\,\mathbf{ws}\\ \Pi; \Gamma \vdash \{z_2 : b|\phi_2\}\,\mathbf{ws}\\ \Pi; \Gamma, z_1 : b[\phi_1] \models \phi_2[z_1/z_1]\end{array}}{\Pi; \Gamma \vdash \{z_1 : b|\phi_1\} \lesssim \{z_2 : b|\phi_2\}}\ \textsc{subtype\_subtype}$$

For the typing judgements for values, expressions and statements we make use of bidirectional judgements. We have the type synthesis judgements $\Pi; \Gamma \vdash v \Rightarrow \tau$ and $\Pi; \Gamma; \Delta \vdash e \Rightarrow \tau$ for values and expressions respectively. It is important that the rules for these judgements synthesise types that have constraints that are within the decidable logic of the SMT solver. With some of the rules, for example INFER_E_PLUS, we do not need to include the constraints ϕ_1 or ϕ_2 explicitly in the conclusion of the rule: If v_1 is a variable then it and ϕ_1 will appear in Γ; if v_1 is a number or a boolean then ϕ_1 is a trivial equality constraint and will be reflected in the type synthesised by INFER_E_PLUS as v_1 appears in the constraint of the type.

$\boxed{\Pi; \Gamma \vdash v \Rightarrow \tau}$ Infer that type of v is τ

$$\frac{\begin{array}{c}\Pi \vdash \Gamma\,\mathbf{ws}\\ x : b[\phi] \in \Gamma\end{array}}{\Pi; \Gamma \vdash x \Rightarrow \{z : b|z = x\}}\ \textsc{infer\_v\_var}$$

$$\frac{\Pi \vdash \Gamma\,\mathbf{ws}}{\Pi; \Gamma \vdash () \Rightarrow \{z : \mathbf{unit}|z = ()\}}\ \textsc{infer\_v\_unit}$$

$$\frac{\Pi \vdash \Gamma\,\mathbf{ws}}{\Pi; \Gamma \vdash \mathbf{T} \Rightarrow \{z : \mathbf{bool}|z = \mathbf{T}\}}\ \textsc{infer\_v\_true}$$

$$\frac{\Pi \vdash \Gamma\,\mathbf{ws}}{\Pi; \Gamma \vdash \mathbf{F} \Rightarrow \{z : \mathbf{bool}|z = \mathbf{F}\}}\ \textsc{infer\_v\_false}$$

$$\frac{\Pi \vdash \Gamma\ \mathbf{ws}}{\Pi; \Gamma \vdash n \Rightarrow \{z : \mathbf{int} | z = n\}} \quad \text{INFER\_V\_NUM}$$

$$\frac{\begin{array}{c}\Pi; \Gamma \vdash v_1 \Rightarrow \{z_1 : b_1 | \phi_1\} \\ \Pi; \Gamma \vdash v_2 \Rightarrow \{z_2 : b_2 | \phi_2\}\end{array}}{\Pi; \Gamma \vdash (v_1, v_2) \Rightarrow \{z : b_1 * b_2 | z = (v_1, v_2)\}} \quad \text{INFER\_V\_PAIR}$$

$$\frac{\begin{array}{l}\mathbf{union}\, tid = \{\ \overline{C_i : \tau_i}^{\,i}\ \} \in \Pi \\ \Pi; \Gamma \vdash v \Leftarrow \tau\end{array}}{\Pi; \Gamma \vdash C_j\ v \Rightarrow \{z : tid | z = C_j\ v\}} \quad \text{INFER\_V\_DATA\_CONS}$$

$\boxed{\Pi; \Gamma; \Delta \vdash e \Rightarrow \tau}$ Infer that type of e is τ

$$\frac{\begin{array}{l}\Pi; \Gamma \vdash \Delta\ \mathbf{ws} \\ \Pi; \Gamma \vdash v_1 \Rightarrow \{z_1 : \mathbf{int} | \phi_1\} \\ \Pi; \Gamma \vdash v_2 \Rightarrow \{z_2 : \mathbf{int} | \phi_2\}\end{array}}{\Pi; \Gamma; \Delta \vdash v_1 + v_2 \Rightarrow \{z_3 : \mathbf{int} | z_3 = v_1 + v_2\}} \quad \text{INFER\_E\_PLUS}$$

$$\frac{\begin{array}{l}\Pi; \Gamma \vdash \Delta\ \mathbf{ws} \\ \Pi; \Gamma \vdash v_1 \Rightarrow \{z_1 : \mathbf{int} | \phi_1\} \\ \Pi; \Gamma \vdash v_2 \Rightarrow \{z_2 : \mathbf{int} | \phi_2\}\end{array}}{\Pi; \Gamma; \Delta \vdash v_1 \le v_2 \Rightarrow \{z_3 : \mathbf{bool} | z_3 = v_1 \le v_2\}} \quad \text{INFER\_E\_LEQ}$$

$$\frac{\begin{array}{l}\Pi; \Gamma \vdash \Delta\ \mathbf{ws} \\ \mathbf{val}\, f : (x : b[\phi]) \to \tau \in \Pi \\ \Pi; \Gamma \vdash v \Leftarrow \{z : b | \phi\}\end{array}}{\Pi; \Gamma; \Delta \vdash f\ v \Rightarrow \tau[v/x]} \quad \text{INFER\_E\_APP}$$

$$\frac{\begin{array}{l}\Pi; \Gamma \vdash \Delta\ \mathbf{ws} \\ \Pi; \Gamma \vdash v \Rightarrow \{z : b_1 * b_2 | \phi\}\end{array}}{\Pi; \Gamma; \Delta \vdash \mathbf{fst}\ v \Rightarrow \{z : b_1 | z = \mathbf{fst}\ v\}} \quad \text{INFER\_E\_FST}$$

$$\frac{\begin{array}{l}\Pi; \Gamma \vdash \Delta\ \mathbf{ws} \\ \Pi; \Gamma \vdash v \Rightarrow \{z : b_1 * b_2 | \phi\}\end{array}}{\Pi; \Gamma; \Delta \vdash \mathbf{snd}\ v \Rightarrow \{z : b_2 | z = \mathbf{snd}\ v\}} \quad \text{INFER\_E\_SND}$$

$$\frac{\begin{array}{l}\Pi; \Gamma \vdash \Delta\ \mathbf{ws} \\ u : \tau \in \Delta\end{array}}{\Pi; \Gamma; \Delta \vdash u \Rightarrow \tau} \quad \text{INFER\_E\_MVAR}$$

We also require for values and expressions type checking judgements. The rules for these synthesis a type and then check that it is a subtyping of the type that we are checking against.

$\boxed{\Pi; \Gamma \vdash v \Leftarrow \tau}$ Check that type of v is τ

$$\frac{\begin{array}{l}\Pi; \Gamma \vdash v \Rightarrow \{z_2 : b | \phi_2\} \\ \Pi; \Gamma \vdash \{z_2 : b | \phi_2\} \lesssim \{z_1 : b | \phi_1\}\end{array}}{\Pi; \Gamma \vdash v \Leftarrow \{z_1 : b | \phi_1\}} \quad \text{CHECK\_V\_VAL}$$

$\boxed{\Pi; \Gamma; \Delta \vdash e \Leftarrow \tau}$ Check that type of e is τ

$$\frac{\begin{array}{l}\Pi; \Gamma; \Delta \vdash e \Rightarrow \{z_2 : b | \phi_2\} \\ \Pi; \Gamma \vdash \{z_2 : b | \phi_2\} \lesssim \{z_1 : b | \phi_1\}\end{array}}{\Pi; \Gamma; \Delta \vdash e \Leftarrow \{z_1 : b | \phi_1\}} \quad \text{CHECK\_E\_EXPR}$$

For statements, we have just a checking judgement. For the typing of the **if** statement we make use of knowledge about the value of the condition in each of the two branches. We do similar in the **match** statement. Note also the two different forms for **let** statements. The first is the one without the type annotation and here we need to synthesis the type to give to the variable x. The second is the one without the type and here we merely need to check the type of the statement s_1 against the type in the annotation. The second of these is used when reducing function application where the s_1 is the function body.

$\boxed{\Pi; \Gamma; \Delta \vdash s \Leftarrow \tau}$ Check that type of s is τ

$$\frac{\begin{array}{l}\Pi; \Gamma \vdash \Delta\ \mathbf{ws} \\ \Pi; \Gamma \vdash v \Leftarrow \tau\end{array}}{\Pi; \Gamma; \Delta \vdash v \Leftarrow \tau} \quad \text{CHECK\_S\_VAL}$$

$$\frac{\begin{array}{l}u \notin \mathrm{dom}(\delta) \\ \Pi; \Gamma \vdash v \Leftarrow \tau \\ \Pi; \Gamma; \Delta, u : \tau \vdash s \Leftarrow \tau_2\end{array}}{\Pi; \Gamma; \Delta \vdash \mathbf{var}\, u : \tau := v\, \mathbf{in}\, s \Leftarrow \tau_2} \quad \text{CHECK\_S\_VAR}$$

$$\frac{\begin{array}{l}\Pi; \Gamma \vdash \Delta\ \mathbf{ws} \\ u : \tau \in \Delta \\ \Pi; \Gamma \vdash v \Leftarrow \tau\end{array}}{\Pi; \Gamma; \Delta \vdash u := v \Leftarrow \{z : \mathbf{unit} | \top\}} \quad \text{CHECK\_S\_ASSIGN}$$

$$\frac{\begin{array}{l}\Pi; \Gamma \vdash v \Rightarrow \{x : \mathbf{bool} | \phi_1\} \\ \Pi; \Gamma; \Delta \vdash s_1 \Leftarrow \{z_1 : b | (v = \mathbf{T} \wedge (\phi_1[v/x])) \implies (\phi[z_1/z])\} \\ \Pi; \Gamma; \Delta \vdash s_2 \Leftarrow \{z_2 : b | (v = \mathbf{F} \wedge (\phi_1[v/x])) \implies (\phi[z_2/z])\}\end{array}}{\Pi; \Gamma; \Delta \vdash \mathbf{if}\, v\, \mathbf{then}\, s_1\, \mathbf{else}\, s_2 \Leftarrow \{z : b | \phi\}} \quad \text{CHECK\_S\_IF}$$

$$\frac{\begin{array}{l}\Pi; \Gamma; \Delta \vdash e \Rightarrow \{z : b | \phi\} \\ \Pi; \Gamma, x : b[\phi[x/z]]; \Delta \vdash s \Leftarrow \tau\end{array}}{\Pi; \Gamma; \Delta \vdash \mathbf{let}\, x = e\, \mathbf{in}\, s \Leftarrow \tau} \quad \text{CHECK\_S\_LET}$$

$$\frac{\begin{array}{l}\Pi; \Gamma; \Delta \vdash s_1 \Leftarrow \{z : b | \phi\} \\ \Pi; \Gamma, x : b[\phi[x/z]]; \Delta \vdash s_2 \Leftarrow \tau\end{array}}{\Pi; \Gamma; \Delta \vdash \mathbf{let}\ x : \{z : b | \phi\} = s_1\ \mathbf{in}\ s_2 \Leftarrow \tau} \quad \text{CHECK\_S\_LET2}$$

$$\frac{\begin{array}{l}\mathbf{union}\, \mathit{tid} = \{\ \overline{C_i : \{z_i : b_i | \phi_i\}}^{\,i}\ \} \in \Pi \\ \Pi; \Gamma \vdash v \Rightarrow \{z : \mathit{tid} | \phi\} \\ \overline{\Pi; \Gamma, x_i : b_i[\phi_i[x_i/z_i] \wedge v = C_i\, x_i \wedge (\phi[v/z])]; \Delta \vdash s_i \Leftarrow \tau}^{\,i}\end{array}}{\Pi; \Gamma; \Delta \vdash \mathbf{match}\, v\, \mathbf{of}\, \overline{C_i\, x_i \Rightarrow s_i}^{\,i} \Leftarrow \tau} \quad \text{CHECK\_S\_MATCH}$$

$$\frac{\begin{array}{l}\Pi; \Gamma; \Delta \vdash e \Leftarrow \{z : \mathbf{bool} | \top\} \\ \Pi; \Gamma; \Delta \vdash s \Leftarrow \{z : \mathbf{unit} | \top\}\end{array}}{\Pi; \Gamma; \Delta \vdash \mathbf{while}\, (e)\, \mathbf{do}\, \{s\} \Leftarrow \{z : \mathbf{unit} | \top\}} \quad \text{CHECK\_S\_WHILE}$$

$$\frac{\begin{array}{l}\Pi; \Gamma; \Delta \vdash s_1 \Leftarrow \{z : \mathbf{unit} | \top\} \\ \Pi; \Gamma; \Delta \vdash s_2 \Leftarrow \tau\end{array}}{\Pi; \Gamma; \Delta \vdash s_1; s_2 \Leftarrow \tau} \quad \text{CHECK\_S\_SEQ}$$

7.3 SMT Validity

We have a judgement $\Gamma \models \phi$ that represents the property that a constraint ϕ when considered as a logical predicate is valid in the context Γ. In an implementation judgements of this form will be checked using an SMT solver such as Z3.

We introduce a function $|\_|$ that translates judgements of the form $\Gamma \models \phi$ into SMT logic fragment. If $\Gamma = x_1 : b_1[\phi_1], ..., x_n : b_n[\phi_n]$ then:

$$\begin{aligned} |\Gamma \models \phi| &= |\Gamma| \Longrightarrow |\phi| \\ |\Gamma| &= |\phi_1| \wedge .. \wedge |\phi_n| \end{aligned}$$

The conversion of constraints, expressions and values are done in the obvious way. The exceptions to note are that expressions that are mutable variables or function application do not have a conversion, however, as we will argue below, these will never appear in refinement constraints. We take advantage of the support for algebraic data types in SMT solvers such as Z3 so that we permit forming of pairs, projection out of pairs and use of data constructors in the expressions that can appear in constraints.

The use of the $\Gamma \models \phi$ appears in the subtyping judgement and an important use of the subtyping check is during function application. By means of an example, we show how the SMT solver is used. Suppose we have a function with the signature:

$$\textbf{val } f : (x : \textbf{int } \times \textbf{int } [0 \leq \textbf{fst } x \wedge 0 \leq \textbf{ snd } x]) \rightarrow \{z : \textbf{ int } \mid \textbf{fst } x \leq z \wedge \textbf{ snd } x \leq z\}$$

and that we need to prove the following function application judgement:

$$\Pi; a : \textbf{int } [a = 9], b : \textbf{int } [b = 10]; \Delta \vdash f\ (a, b) \Rightarrow \tau$$

The derivation of the judgement will include this subtyping check:

$$\Pi; a : \textbf{int } [a = 9], b : \textbf{int } [b = 10] \vdash \{z : \textbf{int } \times \textbf{int } | z = (a, b)\ \} \lesssim \{x : \textbf{int } \times \textbf{int } | 0 \leq \textbf{fst } x \wedge 0 \leq \textbf{ snd } x\}$$

and this validity check judgement:

$$a : \textbf{int } [a = 9], b : \textbf{int } [b = 10], z : \textbf{int } \times \textbf{int } [z = (a, b)] \models 0 \leq \textbf{fst } z \wedge 0 \leq \textbf{ snd } z$$

To check the validity of this, we generate the following Z3 script where we ask Z3 to check the satisfiability of the negated goal.

```
(declare-datatypes (T1 T2) ((Pair (mk-pair (fst T1) (snd T2)))))

(declare-const a Int)
(declare-const b Int)
(declare-const z (Pair Int Int))

(define-fun constraint () Bool (and (= a 9) (= b 10)
   (= z (mk-pair a b)) (not (and ( <= 0 (fst z )) (<= 0 (snd z))))))
(assert constraint)
(check-sat)
```

The first line is a datatype declaration for a pair type. If our context included variables of union type then we would add a declaration for those types. The next three lines declare the variables used; one declaration for each variable in the context. The Z3 type is derived from the MiniSail base type for the variable as per the context. The final three lines are where we define the goal and request Z3 to check that the goal is satisfiable. If the goal is satisfiable then it means that the subtype check has failed.

8 Operational Semantics

In this section we describe the operational semantics; that is how we 'run' a MiniSail program. As indicated above, a MiniSail program is a set of definitions and a single statement. We assume at this stage that the definitions have been processed so that running a program is equivalent to reducing the single statement. Our aim is to reduce the statement to a value however as we have permitted non-terminated while loops and unbounded recursion into the language, a program might not terminate. Since we have mutable variables, we need a store for these variables and so the reduction is the reduction of a store-statement pair to a new store-statement pair. To avoid overwriting existing mutable variable values, for example during recursive function calls, we assume renaming of mutable variables in a function body to fresh variables when applying a function (the rule REDUCE_LET_APP below) The reduction relation is defined in the context of Π that contains the function definitions that we will need when reducing function application.

One step and multi-step reduction is defined as follows:

$\boxed{\Pi \vdash \langle \delta, s\rangle \rightarrow \langle \delta', s'\rangle}$ One step reduction

$$\frac{}{\Pi \vdash \langle \delta, \mathbf{if}\,\mathbf{T}\,\mathbf{then}\, s_1\,\mathbf{else}\, s_2\rangle \rightarrow \langle \delta, s_1\rangle}\ \textsc{reduce\_if\_true}$$

$$\frac{}{\Pi \vdash \langle \delta, \mathbf{if}\,\mathbf{F}\,\mathbf{then}\, s_1\,\mathbf{else}\, s_2\rangle \rightarrow \langle \delta, s_2\rangle}\ \textsc{reduce\_if\_false}$$

$$\frac{}{\Pi \vdash \langle \delta, \mathbf{let}\, x = v\,\mathbf{in}\, s\rangle \rightarrow \langle \delta, s[v/x]\rangle}\ \textsc{reduce\_let\_value}$$

$$\frac{v_1 + v_2 = v}{\Pi \vdash \langle \delta, \mathbf{let}\, x = v_1 + v_2\,\mathbf{in}\, s\rangle \rightarrow \langle \delta, \mathbf{let}\, x = v\,\mathbf{in}\, s\rangle}\ \textsc{reduce\_let\_plus}$$

$$\frac{v_1 \leq v_2 = v}{\Pi \vdash \langle \delta, \mathbf{let}\, x = v_1 \leq v_2\,\mathbf{in}\, s\rangle \rightarrow \langle \delta, \mathbf{let}\, x = v\,\mathbf{in}\, s\rangle}\ \textsc{reduce\_let\_leq}$$

$$\frac{\begin{array}{c}\mathbf{val}\, f : (x : b[\phi]) \rightarrow \tau \in \Pi \\ \mathbf{function}\, f(x) = s_1 \in \Pi\end{array}}{\Pi \vdash \langle \delta, \mathbf{let}\, y = f\ v\,\mathbf{in}\, s_2\rangle \rightarrow \langle \delta, \mathbf{let}\ y : \tau[v/x] = s_1[v/x]\ \mathbf{in}\ s_2\rangle}\ \textsc{reduce\_let\_app}$$

$$\frac{}{\Pi \vdash \langle \delta, \mathbf{let}\, x = \mathbf{fst}\,(v_1, v_2)\,\mathbf{in}\, s\rangle \rightarrow \langle \delta, \mathbf{let}\, x = v_1\,\mathbf{in}\, s\rangle}\ \textsc{reduce\_let\_fst}$$

$$\frac{}{\Pi \vdash \langle \delta, \mathbf{let}\, x = \mathbf{snd}\,(v_1, v_2)\,\mathbf{in}\, s\rangle \rightarrow \langle \delta, \mathbf{let}\, x = v_2\,\mathbf{in}\, s\rangle}\ \textsc{reduce\_let\_snd}$$

$$\frac{v = \delta(u)}{\Pi \vdash \langle \delta, \mathbf{let}\, x = u\,\mathbf{in}\, s\rangle \rightarrow \langle \delta, \mathbf{let}\, x = v\,\mathbf{in}\, s\rangle}\ \textsc{reduce\_let\_mvar}$$

$$\frac{u \notin \mathrm{dom}(\delta)}{\Pi \vdash \langle \delta, \mathbf{var}\, u : \tau := v\,\mathbf{in}\, s\rangle \rightarrow \langle \delta[u \mapsto v], s\rangle}\ \textsc{reduce\_mvar\_decl}$$

$$\frac{\delta' = \delta[u \mapsto v]}{\Pi \vdash \langle \delta, u := v\rangle \rightarrow \langle \delta', ()\rangle}\ \textsc{reduce\_mvar\_assign}$$

$$\frac{\Pi \vdash \langle \delta, s_1\rangle \rightarrow \langle \delta', s_1'\rangle}{\Pi \vdash \langle \delta, s_1; s\rangle \rightarrow \langle \delta', s_1'; s\rangle}\ \textsc{reduce\_seq}1$$

$$\frac{}{\Pi \vdash \langle \delta, (); s\rangle \rightarrow \langle \delta, s\rangle}\ \textsc{reduce\_seq}2$$

$$\frac{}{\Pi \vdash \langle \delta, \mathbf{let}\ x : \tau = v\ \mathbf{in}\ s_2\rangle \rightarrow \langle \delta, s_2[v/x]\rangle}\ \textsc{reduce\_let}2\textsc{\_val}$$

$$\frac{\Pi \vdash \langle \delta, s_1\rangle \rightarrow \langle \delta', s_1'\rangle}{\Pi \vdash \langle \delta, \mathbf{let}\ x : \tau = s_1\ \mathbf{in}\ s_2\rangle \rightarrow \langle \delta', \mathbf{let}\ x : \tau = s_1'\ \mathbf{in}\ s_2\rangle}\ \textsc{reduce\_let}2\textsc{\_stmt}$$

$$\frac{}{\Pi \vdash \langle \delta, \mathbf{match}\,(C_j\ v)\,\mathbf{of}\,\overline{C_i\ x_i \Rightarrow s_i}^{i} \rangle \to \langle \delta, s_j[v/x_j] \rangle} \quad \text{REDUCE\_MATCH}$$

$$\frac{x\ \mathbf{fresh}}{\Pi \vdash \langle \delta, \mathbf{while}\,(e)\,\mathbf{do}\,\{s\} \rangle \to \langle \delta, \mathbf{let}\ x = e\ \mathbf{in}\ \mathbf{if}\ x\ \mathbf{then}\,(s; \mathbf{while}\,(e)\,\mathbf{do}\,\{s\})\,\mathbf{else}\,() \rangle} \quad \text{REDUCE\_WHILE}$$

$\boxed{\Pi \vdash \langle \delta_1, s_1 \rangle \xrightarrow{*} \langle \delta_2, s_2 \rangle}$ Multi-step reduction

$$\frac{\Pi \vdash \langle \delta_1, s_1 \rangle \to \langle \delta_2, s_2 \rangle}{\Pi \vdash \langle \delta_1, s_1 \rangle \xrightarrow{*} \langle \delta_2, s_2 \rangle} \quad \text{REDUCE\_MANY\_SINGLE\_STEP}$$

$$\frac{\begin{array}{c}\Pi \vdash \langle \delta_1, s_1 \rangle \to \langle \delta_2, s_2 \rangle \\ \Pi \vdash \langle \delta_2, s_2 \rangle \xrightarrow{*} \langle \delta_3, s_3 \rangle\end{array}}{\Pi \vdash \langle \delta_1, s_1 \rangle \xrightarrow{*} \langle \delta_3, s_3 \rangle} \quad \text{REDUCE\_MANY\_MANY\_STEP}$$

We prove later that the rules presented above are the only rules needed to ensure progress. Although the **if** statement takes any value, we only need to include the cases for **T** and **F**. We can excluded the case where v is a variable as the statement being reduced contains no free variables (it will contain none when the reduction starts and the preservation typing lemma tells us that it will continue to have no free variables after each reduction). Similarly, from the premises of the preservation lemma, we are only considering well typed statements and so v cannot be anything other than boolean literals. A similar argument applies to the REDUCE_LET_PLUS and REDUCE_LET_LEQ rules.

The '**let** $x : \tau = s_1$ **in** s_2' construct is a form that is introduced during the function application reduction step and allows us to process the body of a function, step-wise reducing the body until we get a value and then using REDUCE_LET_APP to substitute this value into the body of the let statement. The type annotation on the let-bound variable is needed for the proofs below.

In the reduction rule for application, occurrences of x, the function argument variable, in types (particularly types for mutable variables in the statement of the function body, s_1) will have been substituted away and so these types will contain no free variables. This is important for the operational typing rule where we check that the value for each variable in the mutable variable store is a subtype of the variable's type as specified in Δ.

9 Lemmas

This section states and proves and number of lemmas that build up to a proof of the type safety property of MiniSail. We start with the inversion lemmas, then move on to the regularity lemmas that tell us that well-scoping is preserved in the judgements that we work with. Next are lemmas that link the constraints in the types to validity. Following are substitution lemmas, proofs of preservation and progress, and finally the safety lemma.

9.1 Inversion Lemmas

Each typing judgement is defined by a set of inductive rules that can include in their premises instances of the judgement being defined, instance of other judgements defined by other rules or predicates such as membership in a context. The complete set of valid judgements of any form is the set of judgements that can be generated by all of rules and no more. Each valid judgement has a derivation tree associated to it where each node represents the application of a rule and the root node represents the rule that leads directly to the judgement. It is a feature of the MiniSail type system that there is only one possible derivation tree for a valid judgement and in particular only one rule for the root node. This is a consequence of the fact that the type synthesis rules for value and expression and the type checking rules for statement have been constructed so that there is one rule per syntactic form of values, expressions and statements. To put it another way: the rules are syntax directed.

Associated with each judgement, and its rules, is an inductive proof principle. This principle allows us to prove that a property holds for every valid judgement if we can show for every rule the following holds: If the property holds for the premises of the rule then the property holds for the conclusion of the rule. A consequence of this, and the syntax directness property, is that given any valid judgement then the only way that the judgement can arise is by an application of exactly one of the rules. This is known as "inversion" and will be used extensively in the subsequent proofs. We state the inversion lemmas for each of the key typing judgements and give one of example of a proof for one of the cases. We use a more canonical form of the rules in the statement of the lemmas: the type part of the rule conclusion is in a general form and the premises include equalities relating terms in the conclusion to more specific forms.

Lemma 1 (Inversion-$\Rightarrow$ values)**.**
If $\Pi;\Gamma \vdash x \Rightarrow \{z : b|\phi\}$ *then*

$$\Pi \vdash \Gamma \ \boldsymbol{ws}$$
$$x \in dom(\Gamma) \wedge \phi = (z = x)$$

If $\Pi;\Gamma \vdash n \Rightarrow \{z : b|\phi\}$ *then*

$$\Pi \vdash \Gamma \ \boldsymbol{ws}$$
$$b = \boldsymbol{int} \wedge \phi = (z = n)$$

If $\Pi;\Gamma \vdash \boldsymbol{T} \Rightarrow \{z : b|\phi\}$ *then*

$$\Pi \vdash \Gamma \ \boldsymbol{ws}$$
$$b = \boldsymbol{bool} \wedge \phi = (z = \boldsymbol{T})$$

If $\Pi;\Gamma \vdash \boldsymbol{F} \Rightarrow \{z : b|\phi\}$ *then*

$$\Pi \vdash \Gamma \ \boldsymbol{ws}$$
$$b = \boldsymbol{bool} \wedge \phi = (z = \boldsymbol{F})$$

If $\Pi;\Gamma \vdash (v_1, v_2) \Rightarrow \{z : b|\phi\}$ *then there are* b_1*,* b_2*,* ϕ_1 *and* ϕ_2 *such that:*

$$\Pi;\Gamma \vdash v_1 \Rightarrow \{z_1 : b_1|\phi_1\}$$
$$\Pi;\Gamma \vdash v_2 \Rightarrow \{z_2 : b_2|\phi_2\}$$
$$b = b_1 * b_2 \wedge \phi = (z = (v_1, v_2))$$

If $\Pi;\Gamma \vdash C\ v \Rightarrow \{z : b|\phi\}$ *then there is a type id tid and* τ *such that:*

$$b = tid \wedge \phi = (z = \ C\ v)$$
$$\Pi;\Gamma \vdash v \Leftarrow \tau$$
$$\boldsymbol{union}\ tid\ = \{C_1 : \tau_1, .., C : \tau, ... C_n : \tau_n\} \in \Pi$$

If $\Pi;\Gamma \vdash () \Rightarrow \{z : b|\phi\}$ *then:*

$$\Pi \vdash \Gamma \ \boldsymbol{ws}$$
$$b = \boldsymbol{unit} \wedge \phi = (z = ())$$

Proof. We use proof by rule induction with the type synthesis rules for values. We need to show that for each rule the statement of the lemma holds for the conclusion of the rule if it holds for the premises and any side conditions are satisfied. We only show the proof for the pair rule, for the other rules, the proofs follow in the same manner.

Case INFER_V_PAIR. From the induction principle, we assume that the premises of the rule INFER_V_PAIR hold. Hence we have

$$\begin{aligned} \Pi; \Gamma \vdash v_1 \Rightarrow \{z_1 : b_1 | \phi_1\} \\ \Pi; \Gamma \vdash v_2 \Rightarrow \{z_2 : b_2 | \phi_2\} \\ b = b_1 * b_2 \wedge \phi = (z = (v_1, v_2)) \end{aligned}$$

The property we need to prove is a conjunction of implications and we need to prove that it holds for $\Pi; \Gamma \vdash (v_1, v_2) \Rightarrow \{z : b | \phi\}$. All of the implications, except for the pair one, hold immediately as their antecendants do not. For the pair part, the antededent holds and the consequent holds as these are precisely the assumptions we made following the induction principle. Hence the lemma conclusions holds for this rule.

The proof for the other cases follows the same pattern with a different implication taking the role of the pair implication depending on the rule being considered.

□

Lemma 2 (Inversion-$\Leftarrow$ values). *If* $\Pi; \Gamma \vdash v \Leftarrow \tau$ *there there is a* τ' *such that* $\Pi; \Gamma \vdash v \Rightarrow \tau'$ *and* $\Pi; \Gamma \vdash \tau' \lesssim \tau$

Lemma 3 (Inversion-$\Rightarrow$ expressions).
If

$$\Pi; \Gamma; \Delta \vdash v_1 + v_2 \Rightarrow \{z : b | \phi\}$$

then there are ϕ_1 *and* ϕ_2 *such that:*

$$\begin{aligned} \Pi; \Gamma \vdash \Delta \ \boldsymbol{ws} \\ b \equiv \boldsymbol{int} \\ \phi \equiv (z = v_1 + v_2) \\ \Pi; \Gamma \vdash v_1 \Rightarrow \{z_1 : int | \phi_1\} \\ \Pi; \Gamma \vdash v_2 \Rightarrow \{z_2 : int | \phi_2\} \end{aligned}$$

If

$$\Pi; \Gamma; \Delta \vdash v_1 \leq v_2 \Rightarrow \{z : b | \phi\}$$

then there are ϕ_1 *and* ϕ_2 *such that:*

$$\begin{aligned} \Pi; \Gamma \vdash \Delta \ \boldsymbol{ws} \\ b \equiv \boldsymbol{bool} \\ \phi \equiv (z = v_1 \leq v_2) \\ \Pi; \Gamma \vdash v_1 \Rightarrow \{z_1 : int | \phi_1\} \\ \Pi; \Gamma \vdash v_2 \Rightarrow \{z_1 : int | \phi_2\} \end{aligned}$$

If

$$\Pi; \Gamma; \Delta \vdash f \ v \Rightarrow \{z : b | \phi\}$$

then there are b_1*,* ϕ_1*,* τ *and* ϕ_2 *such that:*

$$\begin{aligned} \Pi; \Gamma \vdash \Delta \ \boldsymbol{ws} \\ f :: (x : b_1[\phi_1']) \to \tau \in \Pi \\ \Pi; \Gamma \vdash v \Rightarrow \{z_1 : b_1 | \phi_1\} \\ \Pi; \Gamma \vdash \{z_1 : b_1 | \phi_1\} \lesssim \{z_1 : b_1 | \phi_1'\} \\ \phi \equiv \tau[v/z] \end{aligned}$$

If

$$\Pi;\Gamma;\Delta \vdash u \Rightarrow \{z : b|\phi\}$$

then:

$$\begin{array}{c}\Pi;\Gamma \vdash \Delta \ \boldsymbol{ws} \\ \{z : b|\phi\} = \Delta(u)\end{array}$$

If

$$\Pi;\Gamma;\Delta \vdash \boldsymbol{fst}\ v \Rightarrow \{z : b|\phi\}$$

then there are b' and ϕ' such that:

$$\begin{array}{r}\Pi;\Gamma \vdash \Delta \ \boldsymbol{ws} \\ \phi \equiv (z = \ \boldsymbol{fst}\ v) \\ \Pi;\Gamma \vdash v \Rightarrow \{z : b * b'|\phi'\}\end{array}$$

If

$$\Pi;\Gamma;\Delta \vdash \boldsymbol{snd}\ v \Rightarrow \{z : b|\phi\}$$

then there are b' and ϕ' such that:

$$\begin{array}{r}\Pi;\Gamma \vdash \Delta \ \boldsymbol{ws} \\ \phi \equiv (z = \ \boldsymbol{snd}\ v) \\ \Pi;\Gamma \vdash v_1 \Rightarrow \{z : b' * b|\phi'\}\end{array}$$

Proof. Proof follows the same pattern as the one for Lemma 1. □

Lemma 4 (Inversion-$\Leftarrow$ expressions)**.**
If $\Pi;\Gamma;\Delta \vdash e \Leftarrow \tau$ *there there is a* τ' *such that* $\Pi;\Gamma;\Delta \vdash e \Rightarrow \tau'$ *and* $\Pi;\Gamma \vdash \tau' \lesssim \tau$

Lemma 5 (Inversion-$\Leftarrow$ statements)**.**
If

$$\Pi;\Gamma;\Delta \vdash v \Leftarrow \tau$$

then:

$$\begin{array}{rcl}\Pi;\Gamma & \vdash & \Delta \ \boldsymbol{ws} \\ \Pi;\Gamma & \vdash & v \Leftarrow \tau\end{array}$$

If

$$\Pi;\Gamma;\Delta \vdash \boldsymbol{if}\ v\ \boldsymbol{then}\ s_1\ \boldsymbol{else}\ s_2 \Leftarrow \tau$$

then there are b, ϕ_1 and ϕ such that:

$$\begin{array}{rcl}\Pi;\Gamma;\Delta & \vdash & v \Rightarrow \{z : \boldsymbol{bool}\ |\phi_1\} \\ \Pi;\Gamma;\Delta & \vdash & s_1 \Leftarrow \{z_1 : b|v = \boldsymbol{T} \wedge \phi_1[v/z] \Longrightarrow \phi[z_1/z]\} \\ \Pi;\Gamma;\Delta & \vdash & s_2 \Leftarrow \{z_2 : b|v = \boldsymbol{F} \wedge \phi_1[v/z] \Longrightarrow \phi[z_2/z]\} \\ \tau & \equiv & \{z : b|\phi\}\end{array}$$

If

$$\Pi; \Gamma; \Delta \vdash \textbf{\textit{let}}\ x = e\ \ \textbf{\textit{in}}\ s \Leftarrow \tau$$

then there are b and ϕ such that:

$$\begin{array}{rcl} \Pi; \Gamma; \Delta & \vdash & e \Rightarrow \{z : b|\phi\} \\ \Pi; \Gamma, x : b[\phi[x/z]]; \Delta & \vdash & s \Leftarrow \tau \end{array}$$

If

$$\Pi; \Gamma; \Delta \vdash \textbf{\textit{let}}\ x : \tau' = s_1\ \ \textbf{\textit{in}}\ s_2 \Leftarrow \tau$$

then there are b and ϕ such that:

$$\begin{array}{rcl} \tau' \equiv \{z : b|\phi\} & & \\ \Pi; \Gamma; \Delta & \vdash & s_1 \Leftarrow \{z : b|\phi\} \\ \Pi; \Gamma; \Delta, x : b[\phi[x/z]] & \vdash & s_2 \Leftarrow \tau \end{array}$$

If

$$\Pi; \Gamma; \Delta \vdash \textbf{\textit{while}}\ (s_1)\ \ \ \textbf{\textit{do}}\ \{s_2\} \Leftarrow \tau$$

then

$$\begin{array}{rcl} \tau & \equiv & \{z :\ \ \textbf{\textit{unit}}\ |\top\} \\ \Pi; \Gamma; \Delta & \vdash & s_1 \Leftarrow \{z : \textbf{\textit{bool}}|\top\} \\ \Pi; \Gamma; \Delta & \vdash & s_2 \Leftarrow \{z : \textbf{\textit{unit}}|\top\} \end{array}$$

If

$$\Pi; \Gamma; \Delta \vdash \textbf{\textit{var}}\ u\ \ : \tau = v\ \ \textbf{\textit{in}}\ s \Leftarrow \tau'$$

then

$$\begin{array}{rcl} \Pi; \Gamma; \Delta & \vdash & v \Leftarrow \tau \\ \Pi; \Gamma; \Delta, u : \tau & \vdash & s \Leftarrow \tau' \end{array}$$

If

$$\Pi; \Gamma; \Delta \vdash u := v \Leftarrow \tau$$

then there is a τ' such that:

$$\begin{array}{r} \tau = \{z : \textbf{\textit{unit}}|\top\} \\ \tau' = \Delta(u) \\ \Pi; \Gamma; \Delta \vdash v \Leftarrow \tau' \end{array}$$

If

$$\Pi; \Gamma; \Delta \vdash s_1; s_2 \Leftarrow \tau$$

then

$$\begin{aligned}\Pi;\Gamma;\Delta &\vdash s_1 &\Leftarrow \{z:\textbf{\textit{unit}}|\top\}\\ \Pi;\Gamma;\Delta &\vdash s_2 &\Leftarrow \tau\end{aligned}$$

If

$$\Pi;\Gamma;\Delta \vdash \textbf{\textit{match}}\ v\ \{\ C_1\ x_1 \Rightarrow s_1\ ,\ ..\ ,\ C_n\ x_n \Rightarrow s_n\} \Leftarrow \tau$$

then there is a tid and ϕ such that:

$$\begin{aligned}\textbf{\textit{union}}\ \ \mathit{tid}\ = \{C_1:\{z_1:b''_1|\phi''_1\}..C_n:\{z_n:b''_n|\phi''_n\}\}\in\Pi\\ \Pi;\Gamma\vdash v\Rightarrow\{z:\mathit{tid}|\phi\}\\ \Pi;\Gamma,x_i:b''_i[\phi''_i\wedge v=C_i\ x_i\wedge\phi[v/z]];\Delta\vdash s_i\Leftarrow\tau\quad i=1..n\end{aligned}$$

Proof. Proof follows the same pattern as the proof of Lemma 1. □

9.2 Regularity Lemmas

The following lemma guarantees that for valid typing judgements, the component terms are well-scoped. This means that we can limit the use of well-scoping premises in the rules and only need to use them when well-scoping for a term cannot be inferred from other premises that may include well-scoping judgements.

Lemma 6 (Regularity lemma for values with type synthesis)**.** *If* $\Pi;\Gamma\vdash v\Rightarrow\{z:b|\phi\}$ *then* $\Pi;\Gamma\vdash v$ ***ws*** *and* $\Pi;\Gamma\vdash\{z:b|\phi\}$ ***ws*** *and* $\Pi\vdash\Gamma$ ***ws***

Proof. We prove the first part of the conclusion by induction on v.

If v is a literal then by inversion of the appropriate type synthesis rule for literals we have $\Pi\vdash\Gamma$ **ws**. Using this with the appropriate well-scoping rule for literals we have $\Pi;\Gamma\vdash v$ **ws**.

If v is a variable then using inversion of INFER_V_VAR we have that $x\in\mathrm{dom}(\Gamma)$ and $\Pi\vdash\Gamma$ **ws**. Using these with the rule V_WS_VAR we get required conclusion.

If v is a pair (v_1,v_2), then using inversion and then the induction hypothesis we have that v_1, v_2 and Γ are well-scoped and hence (v_1,v_2) is well-scoped using the rule V_WS_PAIR.

If v is the application of a data constructor, i.e. $v\equiv\ \mathrm{C}\ v'$, then from inversion of INFER_V_DATA_CONS we have:

$$\begin{aligned}\Pi;\Gamma\vdash v'\Rightarrow\tau'\\ \textbf{union}\ \mathrm{tid}\ = \{\mathrm{C}_1:\tau_1,..,\mathrm{C}:\tau',...\mathrm{C}_n:\tau_n\}\in\Pi\end{aligned}$$

By the induction hypothesis, v' and Γ are well-scoped and so we have the required premises for the V_WS_CONS rules and so $\ \mathrm{C}\ v'$ is well-scoped.

We now prove the second conclusion. For values, ϕ always has the form $z=v$. To show that $\Gamma\vdash\{z:b|z=v\}$ **ws** we need to show that $\Gamma,z:b[z=v]\vdash z=v$ **ws** and to show that we need to show:

$$\begin{aligned}\Gamma,z:b[z=v]\vdash z\ \textbf{ws}\\ \Gamma,z:b[z=v]\vdash v\ \textbf{ws}\end{aligned}$$

The first follows from the rule V_WS_VAR rule as z is provided by the context $\Gamma,z:b[z=v]$ and the second from the first part of this proof making use of weakening of well-scoping: v is well-scoped under Γ and so is well-scoped under $\Gamma,z:b[z=v]\vdash z$ □

Lemma 7 (Regularity lemma for values with type checking)**.**
If $\Pi;\Gamma \vdash v \Leftarrow \{z : b|\phi\}$ *then* $\Pi;\Gamma \vdash v$ ***ws*** *and* $\Pi;\Gamma \vdash \{z : b|\phi\}$ ***ws*** *and* $\Pi \vdash \Gamma$ ***ws***

Proof. From inversion with the rule CHECK_V we have that there is a τ such that:

$$\begin{gathered}\Pi;\Gamma \vdash v \Rightarrow \tau \\ \Pi;\Gamma \vdash \tau \lesssim \{z : b|\phi\}\end{gathered}$$

From the previous lemma we have that v and Γ are well-scoped and using inversion on SUBTYPE_SUBTYPE we have that $\{z : b|\phi\}$ is well-scoped. □

Lemma 8 (Regularity lemma for expressions with type synthesis)**.** *If* $\Pi;\Gamma;\Delta \vdash e \Rightarrow \{z : b|\phi\}$ *then* $\Pi;\Gamma;\Delta \vdash e$ ***ws*** *and* $\Pi;\Gamma \vdash \{z : b|\phi\}$ ***ws*** *and* $\Pi \vdash \Gamma$ ***ws*** *and* $\Pi;\Gamma \vdash \Delta$ ***ws***

Proof. Proof is by cases on the structure of e

Case $e \equiv v_1 + v_2$
Using the inversion lemma with INFER_ANF_PLUS we have:

$$\begin{aligned}\Pi;\Gamma &\vdash \Delta\ \mathbf{ws} && (1)\\ \{z : b|\phi\} &= \{z : \mathbf{int}\ |z = v_1 + v_2\} && (2)\\ \Pi;\Gamma &\vdash v_1 \Rightarrow \{z : \mathbf{int}\ |\phi_1\} && (3)\\ \Pi;\Gamma &\vdash v_2 \Rightarrow \{z : \mathbf{int}\ |\phi_2\} && (4)\end{aligned}$$

From the last two, and the previous lemma, v_1, v_2 and Γ are well-scoped so using the rule E_WS_PLUS, $v_1 + v_2$ is well-scoped. To show

$$\Pi;\Gamma \vdash \{z : \mathbf{int}\ |z = v_1 + v_2\}\ \mathbf{ws}$$

We need to show that:

$$\begin{aligned}\Pi;\Gamma, z : \mathbf{int}\ [z = v_1 + v_2] &\vdash z\ \mathbf{ws}\\ \Pi;\Gamma, z : \mathbf{int}\ [z = v_1 + v_2] &\vdash v_1\ \mathbf{ws}\\ \Pi;\Gamma, z : \mathbf{int}\ [z = v_1 + v_2] &\vdash v_2\ \mathbf{ws}\end{aligned}$$

and then use the rules WF_CONS_PRED and E_WS_PLUS, The first holds as z is in the domain of the context; the second two hold using Lemma 6 and (2) and (3) with weakening.

Case $e \equiv v_1 \leq v_2$
Similar to above.

Case $e \equiv f v$
Using inversion with INFER_ANF_APP we have that

$$\begin{aligned}\Pi;\Gamma &\vdash \Delta\ \mathbf{ws} && (1)\\ \mathbf{val}\ f : (x : b_1[\phi_1]) &\rightarrow \tau \in \Pi && (2)\\ \Pi;\Gamma &\vdash v \Rightarrow \{z : b|\phi_2\} && (3)\\ \Pi;\Gamma &\vdash \{z : b|\phi_2\} \lesssim \{z : b|\phi_1[z/x]\} && (4)\end{aligned}$$

Using using Lemma 6 with (2), we have that v and Γ are well-scoped and this along with (1) and (2), gives us that e is well-scoped using the rule E_WS_APP.

Case $e \equiv \textbf{fst } v$
From the inversion lemma, we have that there is a b' and ϕ such that

$$\begin{aligned} \Pi;\Gamma &\vdash \Delta \textbf{ ws} && (1)\\ \Pi;\Gamma &\vdash v \Rightarrow \{z : b * b' | \phi\} && (2)\\ \tau &= \{z : b_1 | z = \textbf{fst } v\}. && (3) \end{aligned}$$

Thus $\Pi;\Gamma \vdash v$ **ws**, using regularity for values, and so $\Pi;\Gamma \vdash \textbf{fst } v \textbf{ ws}$ from E_WS_FST. With weakening we have $\Pi;\Gamma, z : b_1[z = \text{fst } v] \vdash \text{fst } v \textbf{ ws}$ and we have $\Pi;\Gamma, z : b_1[z = \text{fst } v] \vdash z \textbf{ ws}$ and so we have $\Pi;\Gamma \vdash \{z : b_1 | z = \text{fst } v\} \textbf{ ws}$ by WF_CONS_EQ and WF_TYPE_TYPE.

Case $e \equiv \text{snd } v$
As for the previous case.

Case $e \equiv u$
From inversion of INFER_E_MVAR we have that:

$$\{z : b|\phi\} = \Delta(u)$$
$$\Pi;\Gamma \vdash \Delta \text{ wf}$$

From the first and using E_WS_MVAR we have that $\Pi;\Gamma;\Delta \vdash u \textbf{ ws}$. From the second with a finite number of inversions of rule D_WS_CONS, we have that $\Pi;\Gamma \vdash \{z : b|\phi\} \textbf{ bf}$ and so $\Pi;\Gamma \vdash \tau \textbf{ ws}$

□

Lemma 9 (Regularity lemma for expressions with type checking)**.** *If* $\Pi;\Gamma;\Delta \vdash e \Leftarrow \{z : b|\phi\}$ *then* $\Pi;\Gamma;\Delta \vdash e$ ***ws*** *and* $\Pi;\Gamma \vdash \{z : b|\phi\}$ ***ws*** *and* $\Pi \vdash \Gamma$ ***ws*** *and* $\Pi;\Gamma \vdash \Delta$ ***ws***

Proof. From inversion with the rule CHECK_E_EXPR we have that there is a τ such that:

$$\Pi;\Gamma;\Delta \vdash e \Rightarrow \tau$$
$$\Pi;\Gamma \vdash \tau \lesssim \{z : b|\phi\}$$

From the previous lemma we have that e, Γ and Δ are well-scoped and using inversion on SUBTYPE_SUBTYPE we have that $\{z : b|\phi\}$ is well-scoped. □

Lemma 10 (Regularity lemma for statements)**.** *If* $\Pi;\Gamma;\Delta \vdash s \Leftarrow \{z : b|\phi\}$ *then* $\Pi;\Gamma;\Delta \vdash s$ ***ws*** *and* $\Pi;\Gamma \vdash \{z : b|\phi\}$ ***ws*** *and* $\Pi;\Gamma \vdash \Delta$ *and* $\Pi \vdash \Gamma$ ***ws***

Proof. Proof is by cases on the structure of s

Case $s \equiv v$
Using inversion with rule CHECK_S_VAL we have that $\Pi;\Gamma \vdash v \Leftarrow \{z : b|\phi\}$ and that Δ is well-scoped and so from Lemma 7 we have that v and $\{z : b|\phi\}$ are well-scoped.

Case $s \equiv \textbf{if } v \textbf{ then } s_1 \textbf{ else } s_2$
Using inversion with CHECK_S_IF, we have that there is a ϕ' such that

$$\Pi;\Gamma \vdash v \Rightarrow \{z : \mathit{bool}|\phi'\}$$
$$\Pi;\Gamma;\Delta \vdash s_1 \Leftarrow \{z_1 : b|v = \mathbf{T} \wedge \phi'[v/x] \Longrightarrow \phi[z_1/z]\}$$
$$\Pi;\Gamma;\Delta \vdash s_2 \Leftarrow \{z_2 : b|v = \mathbf{F} \wedge \phi'[v/x] \Longrightarrow \phi[z_2/z]\}$$

From Lemma 6 and the induction hypothesis, we have that Γ, Δ, v, s_1 and s_2 are all well-scoped and so s is using rule S_WS_IF.

Using the second of the above with the induction hypothesis we have:

$$\Pi;\Gamma \vdash \{z_1 : b|v = \mathbf{T} \Longrightarrow \phi[z_1/z]\} \textbf{ ws}$$

Using inversion of TYPE_WS_TYPE we have

$$\Pi;\Gamma,z_1:b[\top]\vdash(v=\mathbf{T}\Longrightarrow\phi[z_1/z])\ \mathbf{ws}$$

and with inversion of CONS_WS_IMP we have:

$$\Pi;\Gamma,z_1:b[\top]\vdash\phi[z_1/z]\ \mathbf{ws}$$

We know that z doesn't appear in Γ [1] so we have:

$$\Pi;\Gamma,z:b[\top]\vdash\phi\ \mathbf{ws}$$

and so using TYPE_WS_TYPE we have

$$\Pi;\Gamma\vdash\{z:b|\phi\}\ \mathbf{ws}$$

Case $s\equiv\mathbf{let}\ z=e\ \mathbf{in}\ s'$
From the inversion lemma, we have that there is a b' and ϕ' such that

$$\begin{array}{rcl}\Pi;\Gamma;\Delta&\vdash&e\Rightarrow\{z:b'|\phi'\}\\ \Pi;\Gamma,x:b[\phi'[x/z]];\Delta&\vdash&s'\Leftarrow\{z:b|\phi\}\end{array}$$

From the first we have that: $\Gamma\vdash e\ \mathbf{ws}$ and from the second using the induction hypothesis we have that:

$$\begin{array}{rcl}\Pi;\Gamma,x:b[\phi'[x/z]];\Delta&\vdash&s'\ \mathbf{ws}\\ \Pi;\Gamma&\vdash&\{z:b|\phi\}\ \mathbf{ws}\end{array}$$

and using the rule WF_LET we have that s is well formed in this case.
Case $s\equiv\mathbf{let}\ z:\tau=s_1\ \mathbf{in}\ s_2$
From the inversion lemma we have that there are b' and ϕ' with $\tau=\{z:b'|\phi'\}$ such that:

$$\begin{array}{rcl}\Pi;\Gamma;\Delta&\vdash&s_1\Leftarrow\{z:b'|\phi'\}\\ \Pi;\Gamma,x:b[\phi[x/z]];\Delta&\vdash&s_2\Leftarrow\{z:b|\phi\}\end{array}$$

From the induction hypothesis, we have that τ, s_1 and s_2 are well-scoped and so s is. Furthermore, the induction hypothesis tells us that $\Pi;\Gamma\vdash\{z:b|\phi\}\ \mathbf{ws}$
Case $s\equiv\mathbf{match}\ v\ \{\ \mathrm{C}_1\ x_1\Rightarrow s_1\ ,\ ..\ ,\ \mathrm{C}_n\ x_n\Rightarrow s_n\}$
From inversion with CHECK_S_CASE we have that there exists a tid and ϕ such that:

$$\begin{array}{r}\mathbf{union}\ \ \mathrm{id}\ =\{\mathrm{C}_1:\{z_1:b''_1|\phi''_1\}...\mathrm{C}_n:\{z_n:b''_n|\phi''_n\}\}\in\Pi\\ \Pi;\Gamma\vdash v\Rightarrow\{z:tid|\phi\}\\ \Pi;\Gamma,x_i:b''_i[\phi''_i[x_i/z_i]\wedge v=\mathrm{C}\ _ix_i\wedge\phi[v/z]];\Delta\vdash s_i\Leftarrow\{z:b|\phi\}\quad i=1..n\end{array}$$

Case $s\equiv\mathbf{while}\ (s_1)\ \ \mathbf{do}\ \{s_2\}$
Using inversion of CHECK_S_WHILE we have:

$$\begin{array}{rcl}\Pi;\Gamma;\Delta&\vdash&s_1\Leftarrow\{z:\mathbf{bool}|\top\}\\ \Pi;\Gamma;\Delta&\vdash&s_2\Leftarrow\{z:\mathbf{unit}|\top\}\end{array}$$

and using the induction hypothesis on both s_1 and s_2 we have:

$$\begin{array}{rcl}\Pi;\Gamma;\Delta&\vdash&s_1\Leftarrow\{z:\mathbf{bool}|\top\}\ \mathbf{ws}\\ \Pi;\Gamma;\Delta&\vdash&s_2\Leftarrow\{z:\mathbf{unit}|\top\}\ \mathbf{ws}\\ \Pi;\Gamma;\Delta&\vdash&\{z:\mathbf{unit}|\top\}\ \mathbf{ws}\end{array}$$

The third of these is the second part of the conclusion. Using the first two of these with WF_S_WHILE we have the first part of the conclusion.

[1]Following the convention, if z had appeared in Γ then we could rename z in $\{z:b\phi\}$

Case $s \equiv \textbf{var}\ u\ : \tau = v\ \textbf{in}\ s_1$
Using inversion of rule CHECK_S_VAR we have

$$\begin{array}{rcl} \Pi; \Gamma & \vdash & v \Leftarrow \tau \\ \Pi; \Gamma; \Delta, (u, \tau) & \vdash & s \Leftarrow \{z : b|\phi\} \end{array}$$

By Lemma 6 on the first of these and the induction hypothesis on the second we have:

$$\begin{array}{rcl} \Pi; \Gamma & \vdash & v\ \textbf{ws} \\ \Pi; \Gamma; \Delta, (u, \tau) & \vdash & s\ \textbf{ws} \\ \Pi; \Gamma & \vdash & \{z : b|\phi\}\ \textbf{ws} \end{array}$$

using S_WS_VAR the conclusion follows.

Case $s \equiv u := v$
Using inversion of CHECK_S_ASSIGN we have that there is a τ such that:

$$\begin{array}{r} \Pi; \Gamma x : b_1[\phi_1], \Gamma' \vdash v \Leftarrow \tau \\ (u, \tau) \in \Delta \\ \{z : b|\phi\} = \{z : \textbf{unit}\ |\top\} \end{array}$$

From the first of these we have

$$\Pi; \Gamma x : b_1[\phi_1], \Gamma' \vdash v \Leftarrow \tau$$

and so can apply WF_S_ASSIGN and observe that $\{z : \textbf{unit}\ |\top\}$ is well-scoped to obtain the required conclusion.

Case $s \equiv s_1; s_2$
By inversion of CHECK_S_SEQ, we have

$$\begin{array}{rcl} \Pi; \Gamma; \Delta & \vdash & s_1 \Leftarrow \{z : \textbf{unit}\ |\top\} \\ \Pi; \Gamma; \Delta & \vdash & s_1 \Leftarrow \{z : b|\phi\} \end{array}$$

From the induction hypothesis we have:

$$\begin{array}{rcl} \Pi; \Gamma; \Delta & \vdash & s_1\ \textbf{ws} \\ \Pi; \Gamma; \Delta & \vdash & s_1\ \textbf{ws} \\ \Pi; \Gamma; \Delta & \vdash & \{z : b|\phi\}\ \textbf{ws} \end{array}$$

From the first two using WF_S_SEQ we have the first part the conclusion and the the third is the second part of the conclusion. □

9.3 Soundness

The following lemma is a form of soundness property: it tells us that constraints in the type of a value are valid when that value is substituted into the constraint. This follows from how the inferences rule for values specify exactly the form the constraints in the type of the value can take.

Lemma 11 (Soundness for values)**.**
If $\Pi; \Gamma \vdash v \Rightarrow \{z : b|\phi\}$ *then* $\Gamma \models \phi[v/z]$

Proof. For all forms of v, ϕ is $z = v$ and substituting v for z we get a tautology. □

9.4 Statement checking subtyping

We will need the following lemma during the preservation proof.

Lemma 12 (Subtyping typing for statements)**.**
If

$$\begin{aligned} \Pi; \Gamma; \Delta &\vdash s \Leftarrow \tau_1 \\ \Pi; \Gamma &\vdash \tau_1 \lesssim \tau_2 \end{aligned}$$

then

$$\Pi; \Gamma; \Delta \vdash s \Leftarrow \tau_2$$

Proof. Proof is by induction on the structure of s. When required we use the following: $\tau_1 = \{z : b|\phi_1\}$ and $\tau_2 = \{z : b|\phi_2\}$

Case $s \equiv v$
From the first premise and inversion of CHECK_S_VAL we have,

$$\Pi; \Gamma \vdash v \Leftarrow \tau_1$$

and using inversion of CHECK_V_VAL we have that there is a τ' such that

$$\begin{aligned} \Pi; \Gamma &\vdash v \Rightarrow \tau' \\ \Pi; \Gamma &\vdash \tau' \lesssim \tau_1 \end{aligned}$$

From transitivity of subtyping we have

$$\Pi; \Gamma \vdash \tau' \lesssim \tau_2$$

and so using CHECK_V_VAL and CHECK_S_VAL we have the required conclusion.

Case $s \equiv$ **if** v **then** s_1 **else** s_2

From the first premise and inversion with rule CHECK_ANF_IF we have that there is a ϕ_3 such that:

$$\begin{aligned} \Pi; \Gamma &\vdash v \Rightarrow \{z_3 : \mathbf{bool}\ |\phi_3\} && (1) \\ \Pi; \Gamma; \Delta &\vdash s_1 \Leftarrow \{z : b|v = \mathbf{T} \wedge (\phi_3[v/z_3]) \implies \phi_1\} && (2) \\ \Pi; \Gamma; \Delta &\vdash s_2 \Leftarrow \{z : b|v = \mathbf{F} \wedge (\phi_3[v/z_3]) \implies \phi_1\} && (3) \end{aligned}$$

We have the following subtyping relations:

$$\begin{aligned} \Pi; \Gamma \vdash \{z : b|v = \mathbf{T} \wedge (\phi_3[v/z_3]) \implies \phi_1\} \lesssim \{z : b|v = \mathbf{T} \wedge (\phi_3[v/z_3]) \implies \phi_2\} \\ \Pi; \Gamma \vdash \{z : b|v = \mathbf{F} \wedge (\phi_3[v/z_3]) \implies \phi_1\} \lesssim \{z : b|v = \mathbf{F} \wedge (\phi_3[v/z_3]) \implies \phi_2\} \end{aligned}$$

and so from the induction hypothesis we have

$$\begin{aligned} \Pi; \Gamma; \Delta &\vdash s_1 \Leftarrow \{z : b|v = \mathbf{T} \wedge (\phi_3[v/z_3]) \implies \phi_2\} \\ \Pi; \Gamma; \Delta &\vdash s_2 \Leftarrow \{z : b|v = \mathbf{F} \wedge (\phi_3[v/z_3]) \implies \phi_2\} \end{aligned}$$

and using CHECK_ANF_IF with these and (1) we obtain the required conclusion.

Case $s \equiv \textbf{let } x = e \textbf{ in } s_1$
From the first premise and inversion with rule CHECK_ANF_LET we have that there is a b_3 and ϕ_3 such that:

$$\begin{aligned} \Pi;\Gamma;\Delta &\vdash e \Rightarrow \{z_3 : b_3|\phi_3\} \\ \Pi;\Gamma, x : b_3[\phi_3[x/z_3]];\Delta &\vdash s_1 \Leftarrow \{z : b|\phi_1\} \end{aligned}$$

Using the induction hypothesis on the last of these we get

$$\Pi;\Gamma, x : b_3[\phi_3[x/z_3]];\Delta \vdash s_1 \Leftarrow \{z : b|\phi_2\}$$

and so we can use CHECK_ANF_LET to obtain the required conclusion.

Case $s \equiv \textbf{let } x : \tau = s_1 \textbf{ in } s_2$
From the first premise and inversion with rule CHECK_ANF_LET2 we have that, given $\tau = \{z_3 : b_3|\phi_3\}$

$$\begin{aligned} \Pi;\Gamma;\Delta &\vdash s_1 \Leftarrow \{z_3 : b_3|\phi_3\} \\ \Pi;\Gamma, x : b_3[\phi_3[x/z_3]];\Delta &\vdash s_2 \Leftarrow \{z : b|\phi_1\} \end{aligned}$$

Applying the induction hypothesis to the second of these and then CHECK_ANF_LET2 with the first we obtain the required conclusion.

Case $s \equiv \textbf{match } v \; \{ \; \mathrm{C}_1 \; x_1 \Rightarrow s_1 \; , \; .. \; , \; \mathrm{C}_n \; x_n \Rightarrow s_n\}$
From the premises and inversion of the CHECK_S_CASE rule we have that there is a tid such that:

$$\begin{aligned} \textbf{union} \;\; \mathrm{tid} \; &= \{\mathrm{C}\;_1 : \{z_1 : b_1''|\phi_1''\} \;\; .. \;\; \mathrm{C}\;_n : \{z_n : b_n''|\phi_n''\}\} \in \Pi && (1) \\ \Pi;\Gamma &\vdash v \Rightarrow \{z : tid|\phi\} && (2) \\ \Pi;\Gamma, x_i : b_i''[\phi_i''[x_i/z_i] \wedge v = C_i \; x_i \wedge \phi[v/z]];\Delta &\vdash s_i \Leftarrow \{z : b|\phi_1\} \;\; i = 1..n && (3) \end{aligned}$$

Apply the induction hypothesis to the last of these we have:

$$\Pi;\Gamma, x_i : b_i''[\phi_i''[x_i/z_i] \wedge v = C_i \; x_i \wedge \phi[v/z]];\Delta \vdash s_i \Leftarrow \{z : b|\phi_2\} \;\; i = 1..n \qquad (4)$$

And so we can use CHECK_S_CASE to obtain the required conclusion.

Case $s \equiv \textbf{while } (s_1) \;\; \textbf{do } \{s_2\}$
From the premises and inversion of CHECK_S_WHILE we that $b = \textbf{unit}$ and $\phi_1 = \top$. From the second premise of this lemma we can only have that $\phi_2 = \top$ and so $\tau_1 = \tau_2$ and the conclusion follows.

Case $s \equiv \textbf{var } u \;\; : \tau = v \textbf{ in } s_1$
From inversion of CHECK_S_VAR , we have:

$$\begin{aligned} \Pi;\Gamma &\vdash v \Leftarrow \tau && (1) \\ \Pi;\Gamma;\Delta,(u,\tau) &\vdash s \Leftarrow \{z : b|\phi_1\} && (2) \end{aligned}$$

Applying the induction hypothesis to the second of these we get:

$$\Pi;\Gamma,;\Delta,(u,\tau) \vdash s \Leftarrow \{z : b|\phi_2\} \qquad (3)$$

and so applying CHECK_S_VAR to (1) and (3) we get the required conclusion.

Case $s \equiv u := v$
From inversion of CHECK_S_ASSIGN, we have that $b = \textbf{unit}$ and $\phi_1 = \top$. From the second premise of the lemma we can only have $\phi_2 = \top$ and so $\tau_1 = \tau_2$ and the required conclusion follows.

Case $s \equiv s_1; s_2$
By inversion of CHECK_S_SEQ, we have

$$\begin{aligned} \Pi;\Gamma;\Delta &\vdash s_1 \Leftarrow \{z : \textbf{ unit } |\top\} && (1)\\ \Pi;\Gamma;\Delta &\vdash s_2 \Leftarrow \{z : b|\phi_1\} && (2) \end{aligned}$$

Applying the induction hypothesis to the second we get:

$$\Pi;\Gamma;\Delta \vdash s_2 \Leftarrow \{z : b|\phi_2\} \qquad (3)$$

and applything CHECK_S_SEQ to this and (1) we get the required conclusion

□

9.5 Context Subtyping

The lemmas in this section give us subtyping properties for the Γ-context. What this means is that if we have a judgement making use of a Γ-context that contains a variable x of type τ then we can replace τ with a subtype of τ and the judgement remains valid. We will make use of the context subtyping lemma for statements when we prove the substitution lemma for **let** statements as it will allow us to have a subtype of the bound x of the **let** statement in the context. We split the proofs into two goups: One group for the well-scoping judgements and one for the typing judgements.

9.5.1 Context Subtyping for Well-Scoped Judgements

Lemma 13 (Context subtyping for well-scoped Γ, values, constraint expressions and constraints)**.**

If

$$\begin{aligned} \Pi &\vdash \Gamma\ \boldsymbol{ws}\\ \Pi &\vdash b\ \boldsymbol{ws}\\ \Pi &\vdash \Gamma,\Gamma'\ \boldsymbol{ws}\\ \Pi;\Gamma &\vdash \{z : b|\phi'\} \lesssim \{z : b|\phi\}\\ x &\notin dom(\Gamma,\Gamma') \end{aligned}$$

then

1. *If* $\Pi \vdash \Gamma, x : b[\phi[x/z]], \Gamma'$ ***ws*** *then* $\Pi \vdash \Gamma, x : b[\phi'[x/z]], \Gamma'$ ***ws***

2. *If* $\Pi; \Gamma, x : b[\phi[x/z]], \Gamma' \vdash v$ ***ws*** *then* $\Pi; \Gamma, x : b[\phi'[x/z]], \Gamma' \vdash v$ ***ws***

3. *If* $\Pi; \Gamma, x : b[\phi[x/z]], \Gamma' \models^{\phi} e$ ***ws*** *then* $\Pi; \Gamma, x : b[\phi'[x/z]], \Gamma' \models^{\phi} e$ ***ws***

4. *If* $\Pi; \Gamma, x : b[\phi[x/z]], \Gamma' \vdash \phi''$ ***ws*** *then* $\Pi; \Gamma, x : b[\phi[x/z]], \Gamma' \vdash \phi''$ ***ws***

Proof. We prove this by induction on Γ'. Assume that Γ' is empty.

1. Using inversion of SUBTYPE_SUBTYPE with the fourth premise we have:

$$\Pi; \Gamma \vdash \{z : b|\phi'\}\ \textbf{ws}$$

and from this using inversion of TYPE_WS we have

$$\Pi; \Gamma, z : b[\top] \vdash \phi'\ \textbf{ws}$$

Since x does not occur in Γ we can rename z to x and obtain:

$$\Pi; \Gamma, x : b[\top] \vdash \phi'[x/z]\ \textbf{ws}$$

and using G_WS_CONS with the premises we have the required conclusions.

2. We prove this by induction on v.

   If v is a variable y then from the premise of this case and using inversion of V_WS_VAR we have $\Pi \vdash \Gamma, x : b[\phi[x/z]]$ **ws** and $y \in \mathrm{dom}(\Gamma, x : b[\phi[x/z]])$. Using the previous case on the first of these we have $\Pi \vdash \Gamma, x : b[\phi'[x/z]]$ **ws** . Furthermore we have $\mathrm{dom}(\Gamma, x : b[\phi[x/z]]) = \mathrm{dom}(\Gamma, x : b[\phi'[x/z]])$. Hence the premises of V_WS_VAR hold and the conclusion follows.

   If v is a literal then using inversion on the approppriate rule we have $\Pi \vdash \Gamma, x : b[\phi[x/z]]$ **ws** and from the previous case we have $\Pi \vdash \Gamma, x : b[\phi'[x/z]]$ **ws**. Hence we can apply the appropriate rule and obtain the required conclusion.

   If v is a pair (v_1, v_2) then from inversion of V_WS_PAIR we have

   $$\Pi; \Gamma, x : b[\phi[x/z]] \vdash v_1 \ \mathbf{ws}$$
   $$\Pi; \Gamma, x : b[\phi[x/z]] \vdash v_2 \ \mathbf{ws}$$

   and by the induction hypothesis we have

   $$\Pi; \Gamma, x : b[\phi'[x/z]] \vdash v_1 \ \mathbf{ws}$$
   $$\Pi; \Gamma, x : b[\phi'[x/z]] \vdash v_2 \ \mathbf{ws}$$

   and then using V_WS_PAIR we have the required conclusion.

   If v is the application of a data construction $v = C\ v'$ then from inversion of V_WS_CONS we have

   $$\begin{aligned} \mathbf{union}\ \mathrm{tid}\ &= \{\mathrm{C}_1 : \tau_1, .., \mathrm{C} : \tau, ...\mathrm{C}_n : \tau_n\} \in \Pi \\ &\Pi; \Gamma, x : b[\phi[x/z]] \vdash v'' \ \mathbf{ws} \end{aligned}$$

   applying the induction hypothesis to the second of these and then V_WS_CONS we obtain the required conclusion.

3. We prove this by cases on e. For each rule for $\vdash^{\phi}$, we apply inversion and obtain a well-scoping judgement for one or more variables of the form $\Pi; \Gamma, x : b[\phi[x/z]] \vdash v$ **ws** . We can then use the previous case to obtain $\Pi; \Gamma, x : b[\phi'[x/z]] \vdash v$ **ws** and then apply the rule in the forward direction to obtain the required case.

4. We prove this by induction on ϕ. For each rule for well-scoping of ϕ, we apply inversion and obtain well-scoping judgements for one or more expressions or ϕ' that are simpler than ϕ. For the first of these we can appeal to the previous case and for the second make use of the induction hypothesis. We then apply the rule in th forward direction.

Assume that Γ' is $\Gamma'', x_1 : b_1[\phi_1]$.

1. From the induction premises we have $\Pi \vdash \Gamma, x : b[\phi[x/z]], \Gamma'', x_1 : b_1[\phi_1]$ **ws**. Using inversion of G_WS_CONS we have $\Pi \vdash \Gamma, x : b[\phi[x/z]], \Gamma''$ **ws** . Applying the induction hypothesis we have $\Pi \vdash \Gamma, x : b[\phi'[x/z]], \Gamma''$ **ws**

   Furthermore from the inversion G_WS_CONS we also have $\Pi; \Gamma, x : b[\phi[x/z]], \Gamma'', x_1 : b_1[\top] \vdash \phi_1 \mathbf{ws}$ and from the induction hypthesis (recalling that this is mutually induction across all 4 conclusions) we have $\Pi; \Gamma, x : b[\phi'[x/z]], \Gamma'', x_1 : b_1[\top] \vdash \phi_1 \mathbf{ws}$. We can thus apply G_WS_CONS to obtain the required conclusion.

2. We prove this by induction on v similar to the $\Gamma' = \bullet$ induction case.

   If v is a variable y then from inversion of V_WS_VAR we have $\Pi \vdash \Gamma, x : b[\phi[x/z]], \Gamma'$ and $y \in \mathit{dom}(\Gamma, x : b[\phi[x/z]], \Gamma')$. Using the previous case on the first of these we have $\Pi \vdash \Gamma, x :$

$b[\phi'[x/z]], \Gamma'$ **ws** . Furthermore we have $dom(\Gamma, x : b[\phi[x/z]], \Gamma') = dom(\Gamma, x : b[\phi'[x/z]], \Gamma')$. Hence the premises of V_WS_VAR hold and the conclusion follows.

If v is a literal then using inversion on the approppriate rule we have $\Pi \vdash \Gamma, x : b[\phi[x/z]], \Gamma'$ **ws** and from the previous case we have $\Pi \vdash \Gamma, x : b[\phi'[x/z]], \Gamma'$ **ws** . Hence we can apply the approprate rule and obtain the required conclusion.

If v is a pair or the application of a data constructor then we make use of the induction hypothesis.

3. For e the same argument as for the base case can be used.

4. For ϕ the same argument as for the base case can be used.

□

Lemma 14 (Context subtyping for well-scoped types)**.** *If*

$$\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' \vdash \tau \ \textbf{\textit{ws}}$$
$$\Pi; \Gamma \vdash \{z_1 : b_1 | \phi'_1\} \lesssim \{z_1 : b_1 | \phi_1\}$$

then

$$\Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma' \vdash \tau \ \textbf{\textit{ws}}$$

Proof. Let $\tau = \{z : b|\phi\}$ where $z \notin \mathrm{dom}(\Gamma, x : b_1[\phi_1[x/z_1]], \Gamma')$. From the first premise and inversion of TYPE_WS_TYPE we have $\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma', z : b[\top] \vdash \phi$ **ws** . Applying Lemma 13 we have $\Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma', z : b[\top] \vdash \phi$ **ws** . Finally using TYPE_WS_TYPE we have $\Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma' \vdash \{z : b|\phi\}$ **ws**

□

Lemma 15 (Context subtyping for well-scoped Δ-context)**.** *If*

$$\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' \vdash \Delta \ \textbf{\textit{ws}}$$
$$\Pi; \Gamma \vdash \{z_1 : b_1 | \phi'_1\} \lesssim \{z_1 : b_1 | \phi_1\}$$

then

$$\Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma' \vdash \Delta \ \textbf{\textit{ws}}$$

Proof. Induction on Δ. Assume that Δ is empty. The conclusion for this case follows from the rule DELTA_WS_EMPTY

Assume that $\Delta = \Delta', (u, \tau)$ From inversion of DELTA_WS_CONS we have the following:

$$\begin{array}{rcl} \Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' & \vdash & \Delta \ \textbf{ws} \\ \Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' & \vdash & \tau \ \textbf{ws} \end{array}$$

using the induction hypothesis on the first and Lemma 14 on the second we have

$$\begin{array}{rcl} \Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma' & \vdash & \Delta \ \textbf{ws} \\ \Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma' & \vdash & \tau \ \textbf{ws} \end{array}$$

and then using DELTA_WS_CONS we obtain the required conclusion.

□

9.5.2 Context Subtyping for Typing Judgements

Lemma 16 (Context subtyping for validity)**.**
If

$$\Pi;\Gamma,x:b_1[\phi_1[x/z_1]],\Gamma' \models \phi$$
$$\Pi;\Gamma\vdash\{z_1:b_1|\phi_1'\}\lesssim\{z_1:b_1|\phi_1\}$$

then

$$\Pi;\Gamma,x:b_1[\phi_1'[x/z_1]],\Gamma' \models \phi$$

Proof. Using inversion of SUBTYPE_SUBTYPE on the second premise we have:

$$\Pi;\Gamma,z_1:b_1[\phi'] \models \phi_1 \tag{1}$$

Assume that for a particular interpretation that $|\Gamma,x:b_1[\phi_1'[x/z_1]],\Gamma'|$ is true. Hence each of $|\Gamma|$, $|x:b_1[\phi_1'[x/z_1]]|$ and $|\Gamma'|$ are true. Hence $|\Gamma,x:b_1[\phi_1'[x/z_1]]|$ is true and using (1) we have that $\phi_1[x/z_1]$ is true. Recombining these parts, we then have that $|\Gamma,x:b_1[\phi_1[x/z_1]],\Gamma'|$ is true and so, using the first premise, ϕ is true. □

Lemma 17 (Context subtyping for subtyping)**.**
If

$$\begin{aligned}\Pi;\Gamma,x:b_1[\phi_1[x/z_1]],\Gamma'\vdash\{z_2:b_2|\phi_2'\}\lesssim\{z_2:b_2|\phi_2\}\\ \Pi;\Gamma\vdash\{z_1:b_1|\phi_1'\}\lesssim\{z_1:b_1|\phi_1\}\end{aligned}$$

then

$$\Pi;\Gamma,x:b_1[\phi_1'[x/z_1]],\Gamma'\vdash\{z_2:b_2|\phi_2'\}\lesssim\{z_2:b_2|\phi_2\}$$

Proof. Using inversion of SUBTYPE_SUBTYPE on the first premise we have:

$$\begin{aligned}\Pi;\Gamma,x:b_1[\phi_1[x/z_1]],\Gamma' &\vdash \{z_2:b_2|\phi_2'\}\ \mathbf{ws}\\ \Pi;\Gamma,x:b_1[\phi_1[x/z_1]],\Gamma' &\vdash \{z_2:b_2|\phi_2\}\ \mathbf{ws}\\ \Pi;\Gamma,x:b_1[\phi_1[x/z_1]],\Gamma',z_2:b_2[\phi_2'] &\models \phi_2\end{aligned}$$

Using Lemma 14 on the first two and Lemma 16 on the third we have :

$$\begin{aligned}\Pi;\Gamma,x:b_1[\phi_1'[x/z_1]],\Gamma' &\vdash \{z_2:b_2|\phi_2'\}\ \mathbf{ws}\\ \Pi;\Gamma,x:b_1[\phi_1'[x/z_1]],\Gamma' &\vdash \{z_2:b_2|\phi_2\}\ \mathbf{ws}\\ \Pi;\Gamma,x:b_1[\phi_1'[x/z_1]],\Gamma',z_2:b_2[\phi_2'] &\models \phi_2[x/z_2]\end{aligned}$$

and using SUBTYPE_SUBTYPE the conclusion follows. □

Lemma 18 (Context subtyping for ⇒-values)**.**
If

$$\begin{aligned}\Pi;\Gamma,x:b_1[\phi_1[x/z_1]],\Gamma'\vdash v\Rightarrow\{z_2:b_2|\phi_2\}\\ \Pi;\Gamma\vdash\{z_1:b_1|\phi_1'\}\lesssim\{z_1:b_1|\phi_1\}\end{aligned}$$

then there is a ϕ_2' such that:

$$\begin{aligned}\Pi;\Gamma,x:b_1[\phi_1'[x/z_1]],\Gamma'\vdash v\Rightarrow\{z_2:b_2|\phi_2'\}\\ \Pi;\Gamma,x:b_1[\phi_1'[x/z_1]],\Gamma'\vdash\{z_2:b_2|\phi_2'\}\lesssim\{z_2:b_2|\phi_2\}\end{aligned}$$

Proof. If v is a literal or a variable then from inversion of the relevant typing rule we have that $\phi_2 \equiv (z = v)$ and that $\Gamma, x : b_1[\phi_1[x/z_1]], \Gamma'$ and Π are well-scoped. From the second premise, we have that ϕ'_1 only has variables from Γ and so $\Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma'$ is well-scoped. Thus the literal or variable typing rules can be used and we have the first conclusion:

$$\Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma' \vdash v \Rightarrow \{z_2 : b_2 | z = v\}$$

and the second conclusion holds due to reflexivity of subtyping.

If v is a pair (v_1, v_2) then we have $\phi_2 = (z = (v_1, v_2))$ and

$$\begin{aligned}\Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma' &\vdash v_1 \Rightarrow \{z_{21} : b_{21} | \phi'_{21}\}\\ \Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma' &\vdash v_2 \Rightarrow \{z_{22} : b_{22} | \phi'_{22}\}\end{aligned}$$

Let $\phi'_2 = (z = (v_1, v_2)))$ and the conclusions follow.

If v is x then, from the first premise, the variable typing rule and inversion, we have that $\phi_2 = \phi_1$. We pick ϕ'_2 to be ϕ'_1. The first part of the conclusion holds, using the variable typing rule, and the second part holds from the second premise.

If v is the application of a data constructor $C\ v'$, then from inversion of INFER_V_DATA_CONS we have that there is a *tid* and k such that $C_k = C$, $\phi_2 = (z = C\ v')$ and

$$\begin{aligned}\textbf{union}\ \ \text{tid}\ = \{\mathrm{C}_1 : \{z''_1 : b''_1 | \phi''_1\}..\mathrm{C}_n : \{z''_n : b''_n | \phi''_n\}\} \in \Pi\\ \Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' \vdash C\ v' \Rightarrow \{z_2 : \text{tid} | z_2 = C\ v'\}\\ \Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' \vdash v' \Leftarrow \{z''_k : b''_k | \phi''_k\}\end{aligned}$$

From the last of these and inversion of CHECK_V_VAL we have that there is a τ' that is a subtype of $\{z''_k : b''_k | \phi''_k\}$. Applying the induction hypothesis, we have that there is a type for v' under the context $\Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma'$ that is a subtype of τ' and hence also a subtype of $\{z''_k : b''_k | \phi''_k\}$ by transitivity of subtyping. Thus the premises of INFER_V_CONS hold and we have the first part of the conclusion

$$\Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma' \ \vdash\ C\ v' \Rightarrow \{z_2 : \text{tid} | z_2 = C\ v'\}$$

The second part holds from reflexity of subtyping. □

Lemma 19 (Context subtyping for $\Leftarrow$-values)**.**
If

$$\begin{aligned}\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' \vdash v \Leftarrow \{z_2 : b_2 | \phi_2\}\\ \Pi; \Gamma \vdash \{z_1 : b_1 | \phi'_1\} \lesssim \{z_1 : b_1 | \phi_1\}\end{aligned}$$

then

$$\Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma' \vdash v \Leftarrow \{z_2 : b_2 | \phi_2\}$$

Proof. From inversion of CHECK_V_VAL, we have that there is a ϕ'_2 such that:

$$\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' \vdash v \Rightarrow \{z_2 : b_2 | \phi'_2\} \tag{1}$$
$$\Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' \vdash \{z_2 : b_2 | \phi'_2\} \lesssim \{z_2 : b_2 | \phi_2\} \tag{2}$$

and so we can apply Lemma 18 to get that there is a ϕ''_2 such that

$$\Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma' \vdash v \Rightarrow \{z : b_2 | \phi''_2\} \tag{3}$$
$$\Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma' \vdash \{z_2 : b_2 | \phi''_2\} \lesssim \{\{z_2 : b_2 | \phi'_2\}\} \tag{4}$$

From Lemma 17 with (2) we have:

$$\Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma' \vdash \{z_2 : b_2|\phi'_2\} \lesssim \{z_2 : b_2|\phi_2\} \tag{5}$$

and so from transivity of subtyping, we have

$$\Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma' \vdash \{z_2 : b_2|\phi''_2\} \lesssim \{z_2 : b_2|\phi_2\} \tag{6}$$

With this and (3) using CHECK_V_VAL we have the required conclusion. □

Lemma 20 (Context subtyping for ⇒-expressions).
If

$$\begin{aligned} \Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma'; \Delta \vdash e \Rightarrow \{z_2 : b_2|\phi_2\} \\ \Pi; \Gamma \vdash \{z_1 : b_1|\phi'_1\} \lesssim \{z_1 : b_1|\phi_1\} \end{aligned}$$

then there is a ϕ'_2 *such that:*

$$\begin{aligned} \Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma'; \Delta \vdash e \Rightarrow \{z2 : b_2|\phi'_2\} \\ \Pi; \Gamma \vdash \{z_2 : b_2|\phi'_2\} \lesssim \{z_2 : b_2|\phi_2\} \end{aligned}$$

Proof. Proof is by cases on the structure of e

Case $e \equiv v_1 + v_2$
The first premise is

$$\Pi; \Gamma, x : b_1[\phi_1[z_1/x]], \Gamma'; \Delta \vdash e \Rightarrow \{z : \mathbf{int}|z = v_1 + v_2\}$$

From the inversion of INFER_E_PLUS, we have

$$\begin{aligned} \Pi; \Gamma, x : b_1[\phi_1[z_1/x]], \Gamma' \vdash v_1 \Rightarrow \{z : \mathbf{int}|\phi_3\} \\ \Pi; \Gamma, x : b_1[\phi_1[z_1/x]], \Gamma' \vdash v_2 \Rightarrow \{z : \mathbf{int}|\phi_4\} \end{aligned}$$

Using these and the previous lemma, there are ϕ'_3 and ϕ'_4 such that:

$$\begin{aligned} \Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma' \vdash v_1 \Rightarrow \{z : \mathbf{int}|\phi'_3\} \\ \Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma' \vdash v_2 \Rightarrow \{z : \mathbf{int}|\phi'_4\} \end{aligned}$$

With these and the rule INFER_ANF_PLUS the first part of the conclusion holds:

$$\Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma'; \Delta \vdash e \Rightarrow \{z : \mathbf{int}|z = v_1 + v_2\}$$

and so does the second part as $\phi'_2 \equiv \phi_2$

Case $e \equiv v_1 \leq v_2$
Similar to the previous case.

Case $e \equiv f\ v$
Instantiating the first premise we have:

$$\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma'; \Delta \vdash fv \Rightarrow \{z_2 : b_2|\phi_2\}$$

Using the inversion of INFER_E_APP, we have that there is a b_3 and ϕ_3 such that:

$$\begin{aligned} \mathbf{val}\ f : (x : b_3[\phi'_3[x/z_3]]) \quad &\rightarrow \quad \{z_2 : b_2|\phi_2\} \in \Pi && (1) \\ \Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' \quad &\vdash \quad v \Rightarrow \{z_3 : b_3|\phi_3\} && (2) \\ \Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' \quad &\vdash \quad \{z_3 : b_3|\phi_3\} \lesssim \{z_3 : b_3|\phi'_3\} && (3) \end{aligned}$$

Applying the previous lemma with the second of these, there is ϕ_3'' such that

$$\begin{array}{rcl} \Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma' & \vdash & v \Rightarrow \{z : b_2 | \phi_3''\} \\ \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma' & \vdash & \{z_3 : b_3 | \phi_3''\} \lesssim \{z_3 : b_3 | \phi_3\} \end{array}$$

Applying Lemma 17 to (3) above we have:

$$\Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma' \vdash \{z_3 : b_3 | \phi_3\} \lesssim \{z_3 : b_3 | \phi_3'\}$$

and with subtyping transitivity we have:

$$\Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma' \vdash \{z_3 : b_3 | \phi_3''\} \lesssim \{z_3 : b_3 | \phi_3'\}$$

and so we can apply INFER_E_APP to get the first part of the conclusion:

$$\Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma'; \Delta \vdash f\ v \Rightarrow \{z_2 : b_2 | \phi_2\}$$

The second holds through reflexivity of subtyping.

Case $e \equiv \textbf{fst}\ v$
Using inversion of INFER_E_FST on the first premise we have that there is a ϕ' and b_2'

$$\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' \vdash v \Rightarrow \{z_2 : b_2 * b_2' | \phi'\}$$

and that ϕ_2 is of the form $z =$ fst v. Then applying Lemma 18, we get that there is ϕ'' such that

$$\begin{array}{rcl} \Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma' & \vdash & v \Rightarrow \{z : b_2 * b_2' | \phi''\} \\ \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma' & \vdash & \{z_3 : b_2 * b_2' | \phi''\} \lesssim \{z_3 : b_2 * b_2' | \phi'\} \end{array}$$

Applying INFER_E_FST to the first of these gives the first part of the conclusion:

$$\Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma'; \Delta \vdash \textbf{fst}\ v \Rightarrow \{z : b_2 | z = \text{fst}\ v\}$$

The second part holds through reflexivity of subtyping.

Case $e \equiv \textbf{snd}\ v$
Similar to the case for fst.

Case $e \equiv u$
From inversion on INFER_E_MVAR we have:

$$\begin{array}{r} \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma'\ \textbf{ws} \\ \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' \vdash \Delta\ \textbf{ws} \\ \Pi\ \textbf{ws} \\ (u, \{z_2 : b_2 | \phi_2\}) \in \Pi \end{array}$$

Use subtyping of context for well-scoping we have:

$$\begin{array}{r} \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma'\ \textbf{ws} \\ \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma' \vdash \Delta\ \textbf{ws} \end{array}$$

and so can use INFER_E_MVAR to get the first part of the conclusion:

$$\Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma'; \Delta \vdash u \Rightarrow \{z_2 : b_2 | \phi_2\}$$

Again, the second part holds through reflexivity of subtyping. □

Lemma 21 (Context subtyping for $\Leftarrow$-expressions).
If

$$\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma'; \Delta \vdash e \Leftarrow \{z_2 : b_2|\phi_2\}$$
$$\Pi; \Gamma \vdash \{z_1 : b_1|\phi_1'\} \lesssim \{z_1 : b_1|\phi_1\}$$

then

$$\Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma'; \Delta \vdash e \Leftarrow \{z_2 : b_2|\phi_2\}$$

Proof. From inversion of CHECK_E_E, we have that there is a ϕ_2' such that:

$$\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma'; \Delta \vdash e \Rightarrow \{z_2 : b_2|\phi_2'\} \tag{1}$$
$$\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' \vdash \{z_2 : b_2|\phi_2'\} \lesssim \{z_2 : b_2|\phi_2\} \tag{2}$$

and so we can apply Lemma 21 to get that there is a ϕ_2'' such that

$$\Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma'; \Delta \vdash e \Rightarrow \{z : b_2|\phi_2''\} \tag{3}$$
$$\Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma' \vdash \{z_2 : b_2|\phi_2''\} \lesssim \{\{z_2 : b_2|\phi_2'\}\} \tag{4}$$

From Lemma 17 with (2) we have:

$$\Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma' \vdash \{z_2 : b_2|\phi_2'\} \lesssim \{z_2 : b_2|\phi_2\} \tag{5}$$

and so from transitivity of subtyping, we have

$$\Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma' \vdash \{z_2 : b_2|\phi_2''\} \lesssim \{z_2 : b_2|\phi_2\} \tag{6}$$

With this and (3) using CHECK_E_E we have the required conclusion. □

Lemma 22 (Context subtyping for statements).
If

$$\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma'; \Delta \vdash s \Leftarrow \{z_2 : b_2|\phi_2\}$$
$$\Gamma \vdash \{z_1 : b_1|\phi_1'\} \lesssim \{z_1 : b_1|\phi_1\}$$

then

$$\Pi; \Gamma; x : b_1[\phi_1'[x/z_1]], \Gamma'; \Delta \vdash s \Leftarrow \{z_2 : b_2|\phi_2\}$$

Proof. Proceed by induction on the structure of s. Note the cases for **let** statements when we add an additional variable to the context require the use of a generalised induction hypothesis.

Case $s \equiv v$
From the first premise and inversion of CHECK_S_VAL we have,

$$\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' \vdash v \Leftarrow \{z_2 : b_2|\phi_2\}$$

Using Lemma 19 with this and the second premise, we get

$$\Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma' \vdash v \Leftarrow \{z_2 : b_2|\phi_2\}$$

and using CHECK_S_VAL we get the required conclusion.

Case $s \equiv \textbf{if } v \textbf{ then } s_1 \textbf{ else } s_2$
From the first premise and inversion with rule CHECK_ANF_IF we have that there is a ϕ_3 such that:

$$\begin{array}{rcll}
\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' & \vdash & v \Rightarrow \{z_3 : \textbf{bool } | \phi_3\} & (1)\\
\Pi; \Gamma, x : b_1[\phi_1[z/z_1]], \Gamma'; \Delta & \vdash & s_1 \Leftarrow \{z_2 : b_2 | v = \textbf{T} \wedge (\phi_3[v/z_3]) \implies \phi_2\} & (2)\\
\Gamma, x : b_1[\phi_1[x/z_1]], \Gamma'; \Delta & \vdash & s_2 \Leftarrow \{z_2 : b_2 | v = \textbf{F} \wedge (\phi_3[v/z_3]) \implies \phi_2\} & (3)
\end{array}$$

Using Lemma 18, the second premise and (1) we have that there is a ϕ_3' such that

$$\begin{array}{rcll}
\Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma' & \vdash & v \Rightarrow \{z : \textbf{bool } | \phi_3'\} & (4)\\
\Pi; \Gamma & \vdash & \{z : \textbf{bool } | \phi_3'\} \lesssim \{z : \textbf{bool } | \phi_3\} & (5)
\end{array}$$

With (2) and (3) and the induction hypothesis we have:

$$\Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma'; \Delta \vdash s_1 \Leftarrow \{z_2 : b_2 | v = \textbf{T} \wedge (\phi_3[v/z_3]) \implies \phi_2\}$$
$$\Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma'; \Delta \vdash s_2 \Leftarrow \{z_2 : b_2 | v = \textbf{F} \wedge (\phi_3[v/z_3]) \implies \phi_2\}$$

From (5) we have since implication is contravariant on subtyping that:

$$\Pi; \Gamma \vdash \{z_2 : b_2 | v = \textbf{T} \wedge (\phi_3[v/z_3]) \implies \phi_2\} \lesssim \{z_2 : b_2 | v = \textbf{T} \wedge (\phi_3'[v/z_3]) \implies \phi_2\} \quad (6)$$
$$\Pi; \Gamma \vdash \{z_2 : b_2 | v = \textbf{T} \wedge (\phi_3[v/z_3]) \implies \phi_2\} \lesssim \{z_2 : b_2 | v = \textbf{F} \wedge (\phi_3'[v/z_3]) \implies \phi_2\} \quad (7)$$

And so using Lemma 12 we have

$$\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma'; \Delta \vdash s_1 \Leftarrow \{z_2 : b_2 | v = \textbf{T} \wedge (\phi_3'[v/z_3]) \implies \phi_2\}$$
$$\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma'; \Delta \vdash s_2 \Leftarrow \{z_2 : b_2 | v = \textbf{F} \wedge (\phi_3'[v/z_3]) \implies \phi_2\}$$

With these and (4) using the rule CHECK_ANF_IF we have the required conclusion.

Case $s \equiv \textbf{let } w = e \textbf{ in } s_1$
From the first premise and inversion with rule CHECK_ANF_LET we have that there is a b_3 and ϕ_3 such that:

$$\begin{array}{rcll}
\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma'; \Delta & \vdash & e \Rightarrow \{z_3 : b_3 | \phi_3\} & (1)\\
\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma', w : b_3[\phi_3[w/z_3]]; \Delta & \vdash & s_1 \Leftarrow \{z_2 : b_2 | \phi_2\} & (2)
\end{array}$$

Using Lemma 21, the second premise and (1) we have that there is a ϕ_3' such that

$$\Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma'; \Delta \vdash e \Rightarrow \{z_3 : b_3 | \phi_3'\} \quad (3)$$
$$\Gamma, x : b_1[\phi_1'], \Gamma' \vdash \{z_3 : b_3 | \phi_3'\} \lesssim \{z_3 : b_3 | \phi_3\} \quad (4)$$

Using the induction hypothesis with (2) and the second premise, we have that:

$$\Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma', w : b_2[\phi_3[w/z_3]]; \Delta \vdash s_1 \Leftarrow \{z_2 : b_2 | \phi_2\}$$

Applying the induction hypothesis again, with this and (4), we have that:

$$\Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma', w : b_2[\phi_3'[w/z_3]]; \Delta \vdash s_1 \Leftarrow \{z_2 : b_2 | \phi_2\}$$

And so with this and (3) we have the conclusion using rule CHECK_ANF_LET.

Case $s \equiv \textbf{let } w : \tau = s_1 \textbf{ in } s_2$
From the first premise and inversion with rule CHECK_ANF_LET2 we have that, given $\tau = \{z_3 : b_3 | \phi_3\}$

$$\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' \vdash s_1 \Leftarrow \{z_3 : b_3 | \phi_3\}$$
$$\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma', w : b_3[\phi_3[w/z_3]] \vdash s_2 \Leftarrow \{z_2 : b_2 | \phi_2\}$$

Applying the induction hypothesis on both of these we get:

$$\Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma' \vdash s_1 \Leftarrow \{z_3 : b_3 | \phi_3\}$$
$$\Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma', w : b_3[\phi_3[w/z_3]] \vdash s_2 \Leftarrow \{z_2 : b_2 | \phi_2\}$$

And using rule CHECK_ANF_LET2 we get the conclusion for this case.

Case $s \equiv$ **match** v { $\mathrm{C}_1\ x_1 \Rightarrow s_1$, .. , $\mathrm{C}_n\ x_n \Rightarrow s_n$}
From the premises and inversion of the CHECK_S_CASE rule we have that there is a tid such that:

$$\begin{array}{rcll} \textbf{union}\ \ \mathrm{tid} & = & \{\mathrm{C}\ _1 : \{z_1 : b''_1 | \phi''_1\} \quad .. \quad \mathrm{C}\ _n : \{z_n : b''_n | \phi''_n\}\} \in \Pi & (1)\\ \Pi; \Gamma, x : b_1[\phi_1], \Gamma' & \vdash & v \Rightarrow \{z : tid | \phi\} & (2)\\ \Pi; \Gamma, x : b_1[\phi_1], \Gamma', x_i : b''_i[\phi''_i[x_i/z_i] \wedge v = C_i\ x_i \wedge \phi[v/z]]; \Delta & \vdash & s_i \Leftarrow \{z_2 : b_2 | \phi_2\} \quad i = 1..n & (3) \end{array}$$

Applying Lemma 19 to (2) and the second premise: we have that there is a ϕ' such that:

$$\begin{array}{rcll} \Pi; \Gamma, x : b_1[\phi'_1], \Gamma' & \vdash & v \Rightarrow \{z : tid | \phi'\} & (4)\\ \Pi; \Gamma & \vdash & \{z : tid | \phi'\} \lesssim \{z : tid | \phi\} & (5) \end{array}$$

We have

$$\begin{array}{rl} \Pi; \Gamma, x : b_1[\phi'_1], \Gamma' \vdash \{x_i : b''_i | \phi''_i[x_i/z_i] \wedge v = C_i\ x_i \wedge \phi'[v/z]\} \lesssim & (6)\\ \{x_i : b''_i | \phi''_i[x_i/z_i] \wedge v = C_i\ x_i \wedge \phi[v/z]\}\ i = 1..n & (7) \end{array}$$

and so applying the induction hypothesis twice to (3) with this subtyping and the second premise we have:

$$\Pi; \Gamma, x : b_1[\phi'_1], \Gamma', x_i : b''_i[\phi''_i[x_i/z_i] \wedge v = C_i\ x_i \wedge \phi'[v/z]]; \Delta \ \vdash \ s_i \Leftarrow \{z_2 : b_2 | \phi_2\} \quad i = 1..n \quad (8)$$

Finally, applying the rule CHECK_S_CASE with (1), (4) and (6) we have the required conclusion:

$$\Pi; \Gamma, x : b_1[\phi'_1], \Gamma'; \Delta \vdash \textbf{match}\ v\ \{\ \mathrm{C}_1\ x_1 \Rightarrow s_1\ ,\ ..\ ,\ \mathrm{C}_n\ x_n \Rightarrow s_n\} \Leftarrow \{z_2 : b_2 | \phi_2\} \quad i = 1..n$$

Case $s \equiv$ **while** (s_1) **do** $\{s_2\}$
From the premises and inversion of CHECK_S_WHILE we have:

$$\begin{array}{rcl} \Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma'; \Delta & \vdash & s_1 \Leftarrow \{z : \textbf{bool} | \top\}\\ \Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma'; \Delta & \vdash & s_2 \Leftarrow \{z : \textbf{unit} | \top\} \end{array}$$

Applying the induction hypothesis to both and then CHECK_S_WHILE we get the required conclusion.

Case $s \equiv$ **var** $u\ : \tau = v$ **in** s_1
From inversion of CHECK_S_VAR , we have:

$$\begin{array}{rcl} \Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' & \vdash & v \Leftarrow \tau\\ \Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma'; \Delta, (u, \tau) & \vdash & s \Leftarrow \{z_2 : b_2 | \phi_2\} \end{array}$$

Applying Lemma 19 to the first and the generalised induction hypothesis to the second we have:

$$\begin{array}{rcl} \Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma' & \vdash & v \Leftarrow \tau\\ \Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma'; \Delta, (u, \tau) & \vdash & s \Leftarrow \{z_2 : b_2 | \phi_2\} \end{array}$$

We can then use CHECK_S_VAR to obtain the required conclusion.

Case $s \equiv u := v$
From inversion of CHECK_S_ASSIGN, we have that there is a τ such that:

$$\begin{array}{r} \Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' \vdash v \Leftarrow \tau\\ (u, \tau) \in \Delta\\ \{z_2 : b_2 | \phi_2\} = \{z_2 : \textbf{unit}\ | \top\} \end{array}$$

Using Lemma 19 on the first of these we have that:

$$\begin{array}{r} \Pi; \Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma' \vdash v \Leftarrow \tau\\ (u, \tau) \in \Delta \end{array}$$

and using CHECK_S_ASSIGN we get the required conclusion.

Case $s \equiv s_1; s_2$
By inversion of CHECK_S_SEQ, we have

$$\begin{array}{rcl} \Pi;\Gamma, x : b_1[\phi_1[x/z_1]], \Gamma';\Delta & \vdash & s_1 \Leftarrow \{z : \ \textbf{unit} \ |\top\} \\ \Pi;\Gamma, x : b_1[\phi_1[x/z_1]], \Gamma';\Delta & \vdash & s_2 \Leftarrow \{z_2 : b_2|\phi_2\} \end{array}$$

Applying the induction hypothesis to both, we get:

$$\begin{array}{rcl} \Pi;\Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma';\Delta & \vdash & s_1 \Leftarrow \{z : \ \textbf{unit} \ |\top\} \\ \Pi;\Gamma, x : b_1[\phi'_1[x/z_1]], \Gamma';\Delta & \vdash & s_2 \Leftarrow \{z_2 : b_2|\phi_2\} \end{array}$$

Using CHECK_S_SEQ we get the required conclusion

□

9.6 Weakening Lemmas

In the typing rules and the operational semantics we sometimes extend a Γ-context or Δ-context with additional elements. This is termed weakening and we need to ensure that certain properties hold in these weakened contexts. We first define precisely what an extension is and prove regularity lemmas.

An extension of a Γ-context or a Δ-context are defined by the following rules:

$\boxed{\Pi \vdash \Gamma \sqsubseteq \Gamma'}$ Γ' is an extension of Γ

$$\frac{\Pi \vdash \Gamma\, \textbf{ws}}{\Pi \vdash \Gamma \sqsubseteq \Gamma} \quad \text{EXTEND\_G\_REFL}$$

$$\frac{\begin{array}{c}\Pi \vdash \Gamma'' \sqsubseteq \Gamma, \Gamma' \\ x \notin \text{dom}(\Gamma, \Gamma') \\ \Pi \vdash \Gamma, x : b[\phi]\, \textbf{ws}\end{array}}{\Pi \vdash \Gamma'' \sqsubseteq \Gamma, x : b[\phi], \Gamma'} \quad \text{EXTEND\_G\_INSERT}$$

$\boxed{\Pi;\Gamma \vdash \Delta \sqsubseteq \Delta'}$ Δ' is an extension of Δ

$$\frac{\Pi;\Gamma \vdash \Delta\, \textbf{ws}}{\Pi;\Gamma \vdash \Delta \sqsubseteq \Delta} \quad \text{EXTEND\_D\_REFL}$$

$$\frac{\begin{array}{c}\Pi;\Gamma \vdash \Delta'' \sqsubseteq \Delta, \Delta' \\ u \notin \text{dom}(\Delta, \Delta') \\ \Pi;\Gamma \vdash \tau\, \textbf{ws}\end{array}}{\Pi;\Gamma \vdash \Delta'' \sqsubseteq \Delta, u : \tau, \Delta'} \quad \text{EXTEND\_D\_INSERT}$$

Lemma 23 (Extension of Γ-context prefix)**.** *If* $\Pi \vdash \Gamma \sqsubseteq \Gamma'$ *and* $\Pi \vdash \Gamma, x : b[\phi]$ ***ws*** *and* $x \notin dom(\Gamma')$ *then* $\Pi \vdash \Gamma, x : b[\phi] \sqsubseteq \Gamma', x : b[\phi]$

Proof. We prove this by induction on the derivation of $\Pi \vdash \Gamma \sqsubseteq \Gamma'$.

Assume that the last step in the derivation is the use of EXTEND_G_REFL so from inversion we have $\Gamma' = \Gamma$. From the premises we have $\Pi \vdash \Gamma, x : b[\phi]$ **ws** . and so we can apply EXTEND_G_REFL and obtain $\Pi \vdash \Gamma, x : b[\phi] \sqsubseteq \Gamma, x : b[\phi]$

Assume that the last step in the derivation is the use of EXTEND_G_INSERT. So by inversion of this rule we have the folllowing:

$$\begin{array}{r} \Gamma' = \Gamma_1, x_1 : b_1[\phi_1], \Gamma_2 \\ \Pi \vdash \Gamma \sqsubseteq \Gamma_1, \Gamma_2 \\ x_1 \notin \text{dom}(\Gamma_1, \Gamma_2) \\ \Pi \vdash \Gamma_1, x_1 : b_1[\phi_1] \ \textbf{ws} \end{array}$$

We need to show the following

$$\begin{gathered}\Pi \vdash \Gamma, x : b[\phi] \sqsubseteq \Gamma_1, \Gamma_2, x : b[\phi] \\ x_1 \notin \mathrm{dom}(\Gamma_1, \Gamma_2, x : b[\phi]) \\ \Pi \vdash \Gamma_1, x_1 : b_1[\phi_1] \ \mathbf{ws}\end{gathered}$$

and then we can apply EXTEND_G_INSERT to obtain the required conclusion. The first we get using the induction hypothesis; An additional induction premises is that $x \notin \mathrm{dom}(\Gamma_1, x_1 : b_1[\phi_1], \Gamma_2)$ from this we have x is not x_1 and so we can conclude the second. The third follows directly. □

Lemma 24 (Extension of Δ-context prefix)**.** *If* $\Pi; \Gamma \vdash \Delta \sqsubseteq \Delta'$ *and* $\Pi; \Gamma \vdash \Delta, u : \tau$ ***ws*** *and* $u \notin$ *dom*(Δ') *then* $\Pi; \Gamma \vdash \Delta, u : \tau \sqsubseteq \Delta', u : \tau$

Proof. We prove this by induction on the derivation of $\Pi; \Gamma \vdash \Delta \sqsubseteq \Delta'$ Assume that the last step in the derivation is the use of EXTEND_D_REFL so from inversion we have $\Delta' = \Delta$. From the premises we have $\Pi; \Gamma \vdash \Delta, (u, \tau)$ **ws** . and so we can apply EXTEND_D_REFL and obtain $\Pi\Gamma \vdash \Delta, (u, \tau) \sqsubseteq \Delta, (u, \tau)$

Assume that the last step in the derivation is the use of EXTEND_D_INSERT. So by inversion of this rule we have the folllowing:

$$\begin{gathered}\Delta' = \Delta_1, (u_1, \tau_1), \Delta_2 \\ \Pi; \Gamma \vdash \Delta \sqsubseteq \Delta_1, \Delta_2 \\ u_1 \notin \mathrm{dom}(\Delta_1, \Delta_2) \\ \Pi; \Gamma; \Delta_1 \vdash \tau \ \mathbf{ws}\end{gathered}$$

We need to show the following

$$\begin{gathered}\Pi; \Gamma \vdash \Delta, (u, \tau) \sqsubseteq \Delta_1, \Delta_2, (u, \tau) \\ u_1 \notin \mathrm{dom}(\Delta_1, \Delta_2, (u, \tau)) \\ \Pi; \Gamma; \Delta_1 \vdash \tau_1 \ \mathbf{ws}\end{gathered}$$

and then we can apply EXTEND_D_INSERT to obtain the required conclusion. The first we get using the induction hypothesis; An additional induction premises is that $u \notin \mathrm{dom}(\Gamma_1, (u, \tau), \Gamma_2)$ from this we have u is not u_1 and so we can conclude the second. The third follows directly. □

9.6.1 Weakening for Well-scoping

The following shows that weakening of a Γ-context preserves the well-scoped property. We show this for a single new entry added into a context. This can then be extended to the addition of multiple entries. As can be seen from the graph in Appendix 3, the well-scoping judgements for Γ, v, ϕ and e (restricted for constraints) form a closed connected cluster. This indicates we will need to prove weakening for these using mutual recursion.

Lemma 25 (Single Weakening Γ-context for Γ, values, constraint expressions and constraints)**.** *If the following hold:*

$$\begin{array}{rcl}\Pi & \vdash & b \ \boldsymbol{ws} \\ \Pi & \vdash & \Gamma \ \boldsymbol{ws} \\ \Pi & \vdash & \Gamma, \Gamma' \ \boldsymbol{ws} \\ \Pi; \Gamma, x : b[\top] & \vdash & \phi \ \boldsymbol{ws} \\ x \notin \mathit{dom}(\Gamma, \Gamma') & & \end{array}$$

then we have the following:

1. $\Pi \vdash \Gamma, x : b[\phi], \Gamma'$ ***ws***

2. If $\Pi; \Gamma, \Gamma' \vdash v$ ***ws*** *then* $\Pi; \Gamma, x : b[\phi], \Gamma' \vdash v$ ***ws***

3. If $\Pi; \Gamma, \Gamma' \models^{\phi} e$ ***ws*** *then* $\Pi; \Gamma, x : b[\phi], \Gamma' \models^{\phi} e$ ***ws***

4. If $\Pi; \Gamma, \Gamma' \vdash \phi'$ ***ws*** *then* $\Pi; \Gamma, x : b[\phi], \Gamma' \vdash \phi'$ ***ws***

Proof. We prove this by induction on Γ' and mutually across all four parts of the conclusion.
If Γ' is empty then we prove the parts as follows:

1. The main premises of the lemma include the premises for the application of the rule G_WS_CONS and using this we get the required conclusion $\Pi \vdash \Gamma, x : b[\phi]$ **ws**
2. This part is proved by induction on v. The base case is when v is a variable or a literal. If it is a variable y, then using the premise of this part of the conclusion we have $\Pi; \Gamma \vdash y$ **ws** and from inversion of INFER_V_WF we have $y \in \text{dom}(\Gamma)$ and so $y \in \text{dom}(\Gamma, x : b[\phi]])$. From the previous conclusion part we have that $\Pi \vdash \Gamma, x : b[\phi]$ **ws**. We thus have the premises required for all of the well-scoping rules for the variable and literal case and so we have $\Pi; \Gamma, x : b[\phi] \vdash v$ **ws**. For the pair and data constructor case, we apply the induction-on-v induction hypothesis.
3. Assume $\Pi; \Gamma \models^{\phi} e$ **ws**. For each of the rules, applying inversion will give us $\Pi; \Gamma \vdash v$ **ws** for one or more v. From the previous case we have $\Pi; \Gamma, x : b[\phi] \vdash v$ **ws** for each v and so we can apply the rule in the forward direction to obtain the required conclusion for the case.
4. This part is proved by induction on ϕ'. Assume $\Pi; \Gamma \vdash \phi'$ **ws**. From inversion of the well-scoping rules for ϕ we see that we only need to show that $\Pi; \Gamma, x : b[\phi] \models^{\phi} e$ **ws** for various e and $\Pi; \Gamma, x : b[\phi] \vdash \phi''$ **ws** for various ϕ'' that are simpler than ϕ'. The first we can use the previous case and the second the induction hypothesis.

Assume that $\Gamma' = \Gamma'', x_1 : b_1[\phi_1]$. We prove the parts as follows:

1. The premises of the inductive step are:

$$\begin{aligned} x \notin \mathit{dom}(\Gamma, \Gamma'', x_1 : b_1[\phi_1]) \\ \Pi \vdash \Gamma, x : b[\phi], \Gamma'' \ \textbf{ws} \\ \Pi \vdash \Gamma, \Gamma'', x_1 : b_1[\phi_1] \ \textbf{ws} \end{aligned}$$

   Applying inversion of G_WS_CONS to the last of these we have:

$$\begin{aligned} x_1 \notin \Gamma, \Gamma'' \\ \Pi \vdash b_1 \ \textbf{ws} \\ \Pi \vdash \Gamma, \Gamma'' \ \textbf{ws} \\ \Pi; \Gamma, \Gamma'' \vdash \phi_1 \ \textbf{ws} \end{aligned}$$

   We thus have:

$$\begin{aligned} x_1 \notin \Gamma, x : b[\phi_1], \Gamma'' \\ \Pi \vdash b_1 \ \textbf{ws} \\ \Pi \vdash \Gamma, x : b[\phi], \Gamma'' \ \textbf{ws} \\ \Pi; \Gamma, x : b[\phi], \Gamma'' \vdash \phi_1 \ \textbf{ws} \end{aligned}$$

   and so can apply G_WS_CONS to get the required conclusion which is

$$\Pi \vdash \Gamma, x : b[\phi], \Gamma'', x_1 : b_1[\phi_1] \ \textbf{ws}$$

2. The premises of the inductive step are:
$$\Pi; \Gamma, \Gamma'', x_1 : b_1[\phi_1] \vdash v \ \mathbf{ws}$$
We prove this by induction on v. If v is a variable y, then from the above and inversion of INFER_V_VAR we have $y \in \mathit{dom}(\Gamma, \Gamma'', x_1 : b_1[\phi_1])$ and so $y \in \mathit{dom}(\Gamma, x : b[\phi], \Gamma'', x_1 : b_1[\phi_1])$. Furthermore we also have $\Pi; \Gamma; \Pi'', x_1 : b_1[\phi_1])$ and so from the previous case we have $\Pi; \Gamma; x : b[c], \Pi'', x_1 : b_1[\phi_1])$. The conclusion hence follows for variables and literals. For pairs and data constructors we make use of the induction hypothesis.

3. Assume $\Phi; \Gamma, \Gamma' \models^{\phi} e \ \mathbf{ws}$. For each of the rules, applying inversion will give us $\Pi; \Gamma, \Gamma' \vdash v \ \mathbf{ws}$ for one ore move v. From the previous case we will then have $\Pi; \Gamma, x : b[\phi], \Gamma' \vdash v \ \mathbf{ws}$ and so we can apply the rule in the forward direction to obtain the required conclusion for the case.

4. This part is proved by induction on ϕ'. Assume $\Pi; \Gamma, \Gamma' \vdash \phi' \ \mathbf{ws}$. From inversion of the well-scoping rules for ϕ we see that we only need to show that $\Pi; \Gamma, x : b[\phi], \Gamma' \models^{\phi} e \ \mathbf{ws}$ for various e and $\Pi; \Gamma, x : b[\phi], \Gamma' \vdash \phi'' \ \mathbf{ws}$ for various ϕ'' that are simpler than ϕ'. The first we can use the previous case and the second use the induction hypothesis.

□

Lemma 26 (Weakening Γ-context for Γ)**.** *If* $\Pi \vdash \Gamma \sqsubseteq \Gamma'$ *and* $\Pi \vdash \Gamma$ ***ws*** *then* $\Pi \vdash \Gamma'$ ***ws***

Proof. Proof is by induction on the derivation of $\Pi \vdash \Gamma \sqsubseteq \Gamma'$. If the last step in the derivation is the rule EXTEND_G_REFL then we have $\Gamma' = \Gamma$ and from inversion we have that $\Pi \vdash \Gamma \ \mathbf{ws}$ and so the required conclusion follows.

If the step in the derivation is the rule EXTEND_G_REFL then we have from inversion that
$$\Gamma' = \Gamma_1, x : b[\phi], \Gamma_2 \tag{1}$$
$$\Pi \vdash \Gamma \sqsubseteq \Gamma_1, \Gamma_2 \tag{2}$$
$$x \notin \mathrm{dom}(\Gamma_1, \Gamma_2) \tag{3}$$
$$\Pi \vdash \Gamma_1, x : b[\phi] \ \mathbf{ws} \tag{4}$$
From the last of these we have using inversion of G_WS_CONS
$$\Pi \vdash \Gamma_1 \ \mathbf{ws} \tag{5}$$
$$\Pi; \Gamma_1 \vdash b \ \mathbf{ws} \tag{6}$$
$$\Pi; \Gamma_1, x : b[\top] \vdash \phi \ \mathbf{ws} \tag{7}$$
We thus have the required premises for the first part of Lemma 25 and get:
$$\Pi \vdash \Gamma_1, x : b[\phi], \Gamma_2 \ \mathbf{ws}$$
which is the required conclusion for this case □

Lemma 27 (Weakening Γ-context for v)**.**
If $\Pi \vdash \Gamma \sqsubseteq \Gamma'$ *and* $\Pi; \Gamma \vdash v$ ***ws*** *then* $\Pi; \Gamma' \vdash v$ ***ws***

Proof. As above the proof is by induction on the derivation of $\Pi \vdash \Gamma \sqsubseteq \Gamma'$ If the last step in the derivation is the rule EXTEND_G_REFL then we have $\Gamma' = \Gamma$ and from the premise for this case the required conclusion follows.

If the step in the derivation is the rule EXTEND_G_REFL then we have from inversion that
$$\Gamma' = \Gamma_1, x : b[\phi], \Gamma_2 \tag{1}$$
$$\Pi \vdash \Gamma \sqsubseteq \Gamma_1, \Gamma_2 \tag{2}$$
$$x \notin \mathrm{dom}(\Gamma_1, \Gamma_2) \tag{3}$$
$$\Pi \vdash \Gamma_1, x : b[\phi] \ \mathbf{ws} \tag{4}$$

From the last of these we have using inversion of G_WS_CONS

$$\Pi \vdash \Gamma_1 \ \textbf{ws} \tag{5}$$
$$\Pi ; \Gamma_1 \vdash b \ \textbf{ws} \tag{6}$$
$$\Pi ; \Gamma_1 , x : b[\top] \vdash \phi \ \textbf{ws} \tag{7}$$

Furthermore, from (2) we can apply the induction hypothesis and get $\Pi ; \Gamma_1 , \Gamma_2 \vdash v$ **ws** We thus have the required premises for the second part of Lemma 25 and get:

$$\Pi \Gamma_1 , x : b[\phi], \Gamma_2 \vdash v \ \textbf{ws}$$

which is the required conclusion for this case ☐

Lemma 28 (Weakening Γ-context for ϕ)**.**
If $\Pi \vdash \Gamma \sqsubseteq \Gamma'$ *and* $\Pi ; \Gamma \vdash \phi$ ***ws*** *then* $\Pi ; \Gamma' \vdash \phi$ ***ws***

Proof. As above the proof is by induction on the derivation of $\Pi \vdash \Gamma \sqsubseteq \Gamma'$ If the last step in the derivation is the rule EXTEND_G_REFL then we have $\Gamma' = \Gamma$ and from the premise for this case the required conclusion follows.

If the step in the derivation is the rule EXTEND_G_REFL then we have from inversion that

$$\Gamma' = \Gamma_1 , x : b[\phi'], \Gamma_2 \tag{1}$$
$$\Pi \vdash \Gamma \sqsubseteq \Gamma_1 , \Gamma_2 \tag{2}$$
$$x \notin \text{dom}(\Gamma_1 , \Gamma_2) \tag{3}$$
$$\Pi \vdash \Gamma_1 , x : b[\phi'] \ \textbf{ws} \tag{4}$$

From the last of these we have using inversion of G_WS_CONS

$$\Pi \vdash \Gamma_1 \ \textbf{ws} \tag{5}$$
$$\Pi ; \Gamma_1 \vdash b \ \textbf{ws} \tag{6}$$
$$\Pi ; \Gamma_1 , x : b[\top] \vdash \phi' \ \textbf{ws} \tag{7}$$

Furthermore, from (2) we can apply the induction hypothesis and get $\Pi ; \Gamma_1 , \Gamma_2 \vdash \phi$ **ws** We thus have the required premises for the fourth part of Lemma 25 and get:

$$\Pi \Gamma_1 , x : b[\phi], \Gamma_2 \vdash \phi \ \textbf{ws}$$

which is the required conclusion for this case ☐

Lemma 29 (Weakening Γ-context for well-scoped types)**.**
If $\Pi ; \Gamma \vdash \tau$ ***ws*** *and* $\Pi \vdash \Gamma \sqsubseteq \Gamma'$ *then* $\Pi ; \Gamma' \vdash \tau$ ***ws***

Proof. Let $\tau = \{ z : b | phi \}$ From the first premise and inversion of TYPE_WF_TYPE we have $\Pi ; \Gamma , x : b[\top] vdash \phi$ **ws**. From Lemma 23 we have $\Pi \vdash \Gamma , x : b[\top] \sqsubseteq \Gamma' , x : b[\top]$ and so using Lemma 28 we have $\Pi ; \Gamma' , x : b[\top] \vdash \phi$ **ws**. Finally, using TYPE_WF_TYPE we have the required conclusion. ☐

Lemma 30 (Weakening Γ-context for well-scoped Δ-context)**.** *If* $\Pi ; \Gamma \vdash \Delta$ ***ws*** *and* $\Pi \vdash \Gamma \sqsubseteq \Gamma'$ *then* $\Pi ; \Gamma' \vdash \Delta$ ***ws***

Proof. We prove this by induction on Δ. Assume that Δ is empty. We have $\Pi \vdash \Gamma'$ **ws** from Lemma 26 and so the conclusion follows from the rule DELTA_WS_EMPTY.

Assume that $\Delta = \Delta' , (u, \tau)$. From the premise we have $\Pi ; \Gamma \vdash \Delta' , (u, \tau)$ WF and from inversion of DELTA_WS_CONS we have:

$$\Pi ; \Gamma \vdash \tau \ \textbf{ws}$$
$$\Pi ; \Gamma \vdash \Delta \ \textbf{ws}$$

Using Lemma 29 on the first of these and the induction hypothesis on the second we have:

$$\Pi; \Gamma' \vdash \tau \textbf{ ws}$$
$$\Pi; \Gamma' \vdash \Delta \textbf{ ws}$$

and so can apply DELTA_WS_CONS to obtain the required conclusion. □

9.6.2 Weakening for Typing Judgements

It is useful to an understanding of the proofs for the weakening lemmas to divide the inductive typing rules for values, expressions and statements into how they make use of or manipulate contexts: The typing rules for values use lookup for Γ and Π contexts; the typing rules for expressions use lookup for Δ and Π contexts and finally the typing rules for statements use lookup for Δ (in the rule for mutable variable assignments), manipulate the Γ context (in the rules for let and case statements) and manipulate the Δ context (in the rule for mutable variable introduction).

Lemma 31 (Weakening Γ-context for validity). *If* $\Pi; \Gamma \models \phi$ *and* $\Pi \vdash \Gamma \sqsubseteq \Gamma'$ *then* $\Pi; \Gamma' \models \phi$

Proof. An interpretation that satisfies $|\Gamma', z_1 : [\phi_1]|$ will also satisfy $|\Gamma, z_1 : [\phi_1]||$ and will satisfy $|\phi_2[z_1/z_2]|$ from the above. Hence we have

$$\Pi; \Gamma', z_1 : b[\phi_1] \models \phi_2[z_1/z_2]$$

□

Lemma 32 (Weakening Γ-context for subtyping). *If* $\Pi; \Gamma \vdash \tau_1 \lesssim \tau_2$ *and* $\Pi \vdash \Gamma \sqsubseteq \Gamma'$ *then* $\Pi; \Gamma' \vdash \tau_1 \lesssim \tau_2$

Proof. Consider $\tau_1 = \{z_1 : b|\phi_1\}$ and $\tau_2 = \{z_2 : b|\phi_2\}$. From inversion of the SUBTYPE_SUBTYPE rule we have

$$\Pi; \Gamma \vdash \{z_1 : b|\phi_1\} \textbf{ ws}$$
$$\Pi; \Gamma \vdash \{z_2 : b|\phi_2\} \textbf{ ws}$$
$$\Pi; \Gamma, z_1 : b[\phi_1] \models \phi_2[z_1/z_2]$$

Using weakening of type well-scoping on the first two and weakening of validity we have:

$$\Pi; \Gamma' \vdash \tau_1 \textbf{ ws}$$
$$\Pi; \Gamma' \vdash \tau_2 \textbf{ ws}$$
$$\Pi; \Gamma', z_1 : b[\phi_1] \models \phi_2[z_1/z_2]$$

and so we can apply SUBTYPE_SUBTYPE to obtain the required conclusion. □

Lemma 33 (Weakening Γ-context for $\Rightarrow$ values). *If* $\Pi; \Gamma \vdash v \Rightarrow \tau$ *and* $\Pi \vdash \Gamma \sqsubseteq \Gamma'$ *then* $\Pi; \Gamma' \vdash v \Rightarrow \tau$.

Proof. If v is a variable, then from inversion of INFER_V_VAR we have that there is a ϕ such that $x : b[\phi] \in \Gamma$. By definition of $\Gamma \sqsubseteq \Gamma'$ we are assured that $x : b[\phi] \in \Gamma'$ and so the conclusion follows using INFER_V_VAR.

If v is a literal, then the typing of v is independent of any particular Γ and so the conclusion holds.

If v is a pair (v_1, v_2), then from inversion we obtain typing judgements for v_1 and v_2 in terms of Γ. Applying the induction hypothesis to both of these, we get typing judgements for v_1 and v_2 in terms of Γ'. We can then apply INFER_V_PAIR to get the conclusion.

If v is the application of a data constructor i.e.$v = C\ v'$, then from inversion we obtain a typing judgement for v' in terms of Γ plus conditions on C. Applying the induction hypothesis to the judgement for v', we get a typing judgement in terms of Γ'. We can then apply INFER_V_DATA_CONS to get the conclusion. □

Lemma 34 (Weakening Γ context for $\Leftarrow$ values)**.** *If* $\Pi; \Gamma \vdash v \Leftarrow \tau$ *and* $\Gamma \sqsubseteq \Gamma'$ *then* $\Pi; \Gamma' \vdash v \Leftarrow \tau$.

Proof. Using inversion of CHECK_V_VAL followed by weakening for subtyping and weakening for value type inference and then applying CHECK_V_VAL, we obtain the required conclusion. □

Lemma 35 (Weakening Γ context and Δ-context for $\Rightarrow$ expressions)**.** *If* $\Pi; \Gamma; \Delta \vdash e \Rightarrow \tau$ *and* $\Delta \sqsubseteq \Delta'$ *and* $\Gamma \sqsubseteq \Gamma'$ *then* $\Pi; \Delta'; \Gamma' \vdash v \Rightarrow \tau$.

Proof. As noted above, the rules for type inference for expressions do not directly manipulate or use the Γ context. For the Δ context, only lookup is used, and by definition of context extension, we know that the lookup in the extension will return the same result. □

Lemma 36 (Weakening Γ context and Δ-context for $\Leftarrow$ expressions)**.** *If* $\Pi; \Gamma; \Delta \vdash e \Leftarrow \tau$ *then* $\Pi; \Delta'; \Gamma' \vdash v \Leftarrow \tau$ *for any* Γ' *and* Δ' *such that* $\Pi \vdash \Gamma \sqsubseteq \Gamma'$ *and* $\Pi; \Gamma' \vdash \Delta \sqsubseteq \Delta'$

Proof. The proof is similar to the proof for value type checking. □

Lemma 37 (Weakening Γ context and Δ-context for $\Leftarrow$ statements)**.** *If* $\Pi; \Gamma; \Delta \vdash s \Leftarrow \{z : b|\phi\}$ *and* $\Pi \vdash \Gamma \sqsubseteq \Gamma'$ *and* $\Pi; \Gamma' \vdash \Delta \sqsubseteq \Delta'$ *then* $\Pi; \Delta'; \Gamma' \vdash s \Leftarrow \{z : b|\phi\}$.

Proof. In the proof cases below we use a generalised induction hypothesis that permits us to extend any Γ or Δ and not just those mentioned in the statement of the lemma. We only give the cases where the corresponding rule uses, or manipulates, a context; other cases follow directly from induction hypothesis or making use of the weakening rules for values or expressions.

Case $s \equiv \textbf{let } w = e \textbf{ in } s$
Using inversion with CHECK_ANF_LET there is a b_3 and ϕ_3 such that:

$$\Pi; \Gamma; \Delta \vdash e \Rightarrow \{z_3 : b_3|\phi_3\}$$
$$\Pi; \Gamma, w : b_3[\phi_3[w/z_3]]; \Delta \vdash s \Leftarrow \{z : b|\phi\}$$

Given a Γ' such that $\Pi \vdash \Gamma \sqsubseteq \Gamma'$ then from Lemma 23 we have $\Pi \vdash \Gamma, w : b_3[\phi_3] \sqsubseteq \Gamma', w : b_3[\phi_3]$ and so from weakening for expressions and using the generalised induction hypothesis we have:

$$\Pi; \Gamma'; \Delta' \vdash e \Rightarrow \{z_3 : b_3|\phi_3\}$$
$$\Pi; \Gamma', w : b_3[\phi_3[w/z_3]]; \Delta' \vdash s \Leftarrow \{z : b|\phi\}$$

and the conclusion follows using CHECK_ANF_LET

Case $s \equiv \textbf{let } w : \tau = s_1 \textbf{ in } s_2$
Using inversion with CHECK_ANF_LET2 we obtain the following:

$$\Pi; \Gamma, ; \Delta \vdash s_1 \Leftarrow \{z_3 : b_3|\phi_3\}$$
$$\Pi; \Gamma, w : b_3[\phi_3[w/z_3]]; \Delta \vdash s \Leftarrow \{z : b|\phi\}$$

Given a Γ' such that $\Pi \vdash \Gamma \sqsubseteq \Gamma'$ then from Lemma 23 we have $\Gamma, w : b_3[\phi_3] \sqsubseteq \Gamma', w : b_3[\phi_3]$ and so from the generalised induction hypothesis we have:

$$\Pi; \Gamma', ; \Delta' \vdash s_1 \Leftarrow \{z_3 : b_3|\phi_3\}$$
$$\Pi; \Gamma', w : b_3[\phi_3[w/z_3]]; \Delta' \vdash s \Leftarrow \{z : b|\phi\}$$

and the conclusion follows using CHECK_ANF_LET2

Case $s \equiv \textbf{match}\ v'\ \{\ \mathrm{C}_1\ x_1 \Rightarrow s_1\ ,\ ..\ ,\ \mathrm{C}_n\ x_n \Rightarrow s_n\}$
From inversion of CHECK_S_MATCH we have that there is an *id* such that

$$\begin{array}{rcl}
\textbf{union}\ id = \{\mathrm{C}\ _1 : \{z_1 : b''_1 | \phi''_1\}\ \ ..\ \ \mathrm{C}\ _n : \{z_n : b''_n | \phi''_n\}\} \in \Pi \\
\Pi;\Gamma & \vdash & v' \Rightarrow \{z : id | \phi\} \\
\Pi;\Gamma, x_i : b''_i[\phi''_i[x_i/z_i] \wedge v' = C\ x_i \wedge \phi[v/z]];\Delta & \vdash & s_i \Leftarrow \{z_1 : b_1 | \phi_1\}\quad i = 1..n
\end{array}$$

Given a Γ' such that $\Pi \vdash \Gamma \sqsubseteq \Gamma'$ then for every $i = 1..n$ we have using Lemma 23 that

$$\Pi \vdash \Gamma, x_i : b''_i[\phi''_i \wedge v' = C\ x_i \wedge \phi[v/z]] \sqsubseteq \Gamma', x_i : b''_i[\phi''_i \wedge v' = C\ x_\wedge \phi[v/z]]$$

Hence we can apply the weakening rule for values and the induction hypothesis to get that there is an *id* such that

$$\begin{array}{rcl}
\textbf{union}\ id = \{\mathrm{cons}_1 : \{z_1 : b''_1 | \phi''_1\}\ \ ..\ \ \mathrm{cons}_n : \{z_n : b''_n | \phi''_n\}\} \in \Pi \\
\Pi;\Gamma' & \vdash & v' \Rightarrow \{z : id | \top\} \\
\Pi;\Gamma', x_i : b''_i[\phi''_i[x_i/z_i] \wedge v' = C\ x_i \wedge \phi[v/z]];\Delta' & \vdash & s_i \Leftarrow \{z_1 : b_1 | \phi_1\}\quad i = 1..n
\end{array}$$

and using CHECK_S_CASE the conclusion follows.

Case $s \equiv \textbf{var}\ u\ : \tau = v'\ \textbf{in}\ s_1$
From the premises and inversion of CHECK_S_VAR we have:

$$\begin{array}{c}
\Pi;\Gamma \vdash v' \Leftarrow \tau \\
\Pi;\Gamma;\Delta,(u,\tau) \vdash s_1 \Leftarrow \{z : b | \phi\}
\end{array}$$

Given a Δ' such that $\Pi;\Gamma' \vdash \Delta \sqsubseteq \Delta'$ from Lemma 24 we have $\Pi;\Gamma' \vdash \Delta,(u,\tau) \sqsubseteq \Delta',(u,\tau)$. Hence we can apply the weakening rule for values and the induction hypothesis to obtain:

$$\begin{array}{c}
\Pi;\Gamma' \vdash v' \Leftarrow \tau \\
\Pi;\Gamma';\Delta',(u,\tau) \vdash s_1 \Leftarrow \{z : b | \phi\}
\end{array}$$

and using CHECK_S_VAR the conclusion follows.

Case $s \equiv u := v'$
From the premises and inversion of CHECK_S_ASSIGN we have that there is a τ such that:

$$(u,\tau) \in \Delta \tag{1}$$
$$\Pi;\Gamma \vdash v' \Leftarrow \tau \tag{2}$$
$$\{z : b | \phi\} = \{z : \textbf{unit}\ | \top\} \tag{3}$$

Given a Δ' such that $\Pi;\Gamma' \vdash \Delta \sqsubseteq \Delta'$ then if $(u,\tau) \in \Delta$ we have $(u,\tau) \in \Delta'$. Hence we can apply the weakening rules for values to obtain:

$$(u,\tau) \in \Delta' \tag{4}$$
$$\Pi;\Gamma' \vdash v' \Leftarrow \tau \tag{5}$$

and use the rule CHECK_S_ASSIGN to get the required conclusion. □

9.7 Substitution Lemmas

9.7.1 Substitution for Well-scoping Judgements

The following assures us that the well-scoping property is preserved under substitution in contexts.

Lemma 38 (Substitution for well-scoped Γ, values, constraint expressions and constraints)**.**
If

$$\begin{array}{rcl}
\Pi & \vdash & \Gamma \ \boldsymbol{ws} \\
\Pi & \vdash & b \ \boldsymbol{ws} \\
\Pi & \vdash & \Gamma, \Gamma' \ \boldsymbol{ws} \\
\Pi ; \Gamma & \vdash & \{z : b | \phi'\} \lesssim \{z : b | \phi\} \\
x & \notin & dom(\Gamma, \Gamma') \\
\Pi ; \Gamma & \vdash & v \Rightarrow \{z : b | \phi\}
\end{array}$$

then

1. *If* $\Pi \vdash \Gamma, x : b[\phi[x/z]], \Gamma'$ $\boldsymbol{ws}$ *then* $\Pi \vdash \Gamma, \Gamma'[v/x]$ $\boldsymbol{ws}$
2. *If* $\Pi ; \Gamma, x : b[\phi[x/z]], \Gamma' \vdash v'$ $\boldsymbol{ws}$ *then* $\Pi ; \Gamma, \Gamma'[v/x] \vdash v'[v/x]$ $\boldsymbol{ws}$
3. *If* $\Pi ; \Gamma, x : b[\phi[x/z]], \Gamma' \vdash^{\phi} e$ $\boldsymbol{ws}$ *then* $\Pi ; \Gamma, \Gamma'[v/x] \vdash^{\phi} e[v/x]$ $\boldsymbol{ws}$
4. *If* $\Pi ; \Gamma, x : b[\phi[x/z]], \Gamma' \vdash \phi''$ $\boldsymbol{ws}$ *then* $\Pi ; \Gamma, \Gamma'[v/x] \vdash \phi''[v/x]$ $\boldsymbol{ws}$

Proof. We prove this by induction on Γ'. Assume that Γ' is empty.

1. Using inversion of G_WS_CONS with $\Pi \vdash \Gamma, x : b[\phi[x/z]]$ **ws** we get $\Pi \vdash \Gamma$ which is the required conclusion.

2. We prove this by induction on v'.

   If v' is a variable y that is not equal to x then we must have $y \in \text{dom}(\Gamma)$ also from above we have $\Pi \vdash \Gamma$ **ws** and so using V_WS_VAR we have $\Pi ; \Gamma \vdash v'$ **ws** and but since $v'[v/x] = v'$ we have the required conclusion.

   If v' is equal to x then we have $v'[v/x] = v$ and by the regularity lemma we already know $\Pi ; \Gamma \vdash v$ **ws** which is the required conclusion.

   If v' is a literal then since $\Pi \vdash \Gamma$ **ws** we can use the appropriate rule to get $\Pi \vdash \Gamma v'$ **ws** and $v'[v/x] = v'$ we the required conclusion.

   If v' is a pair (v_1, v_2) then from inversion of V_WS_PAIR we have

   $$\Pi ; \Gamma, x : b[\phi[x/z]] \vdash v_1 \ \mathbf{ws}$$
   $$\Pi ; \Gamma, x : b[\phi[x/z]] \vdash v_2 \ \mathbf{ws}$$

   and by the induction hypothesis we have

   $$\Pi ; \Gamma \vdash v_1[v/x] \ \mathbf{ws}$$
   $$\Pi ; \Gamma \vdash v_2[v/x] \ \mathbf{ws}$$

   and so using V_WS_PAIR we have the required conclusion since $(v_1, v_2)[v/x] = (v_1[v/x], v_2[v/x])$

   If v' is the application of a data construction $v' = C\ v''$ then from inversion of V_WS_CONS we have

   $$\begin{array}{r}
   \mathbf{union}\ \text{tid}\ = \{\mathrm{C}_1 : \tau_1, .., \mathrm{C} : \tau, ...\mathrm{C}_n : \tau_n\} \in \Pi \\
   \Pi ; \Gamma, x : b[\phi[x/z]] \vdash v'' \ \mathbf{ws}
   \end{array}$$

   applying the induction hypothesis to the second of these and then V_WS_CONS we obtain the required conclusion as $(C\ v'')[v/x] = C\ (v''[v/x])$.

3. We prove this by cases on e. From inspection of the rules for $\vDash^{\phi}$ we see that they all make us of well-scoping of one or more values. So for each case, we can apply inversion, apply the part of this lemma for values and then apply the rule that we inverted to obtain the required conclusion.

4. We prove this by induction on ϕ. From inspection of the rules for well-scoping of ϕ we see that make use well-scoping for smaller ϕ or well-scoping of constraint expressions. So for each form of ϕ, we can use inversion, apply the part of this lemma for constraint expressions, or the induction hypothesis, and then apply the rule that we inverted to obtain the required conclusion.

Assue that Γ' is $\Gamma'', x_1 : b_1[\phi_1]$. Note that $\Gamma, (\Gamma'', x_1 : b_1[\phi_1])[v/x] = \Gamma, \Gamma''[v/x], x_1 : b_1[\phi_1[v/x]]$ from the definition of substitution over Γ-context.

1. From the induction premises we have $\Pi \vdash \Gamma, x : b[\phi[x/z]], \Gamma'', x_1 : b_1[\phi_1]$ **ws**. Using inversion of G_WS_CONS we have $\Pi \vdash \Gamma, x : b[\phi[x/z]], \Gamma''$ **ws** . Applying the induction hypothesis we have $\Pi \vdash \Gamma, \Gamma''[x/v]$ **ws**

   Furthermore from the inversion G_WS_CONS we also have $\Pi; \Gamma, x : b[\phi[x/z]], \Gamma'', x_1 : b_1[\top] \vdash \phi_1$**ws** and from the induction hypthesis (recalling that this is mutually induction across all 4 conclusions) we have $\Pi; \Gamma, (\Gamma'', x_1 : b_1[\top])[v/x] \vdash \phi_1[v/x]$**ws**. We can thus apply G_WS_CONS to get $\Pi \vdash \Gamma, (\Gamma'', x_1 : b_1[\phi])[v/x]$

2. We prove this by induction on v similar to the $\Gamma' = \cdot$ induction case.

   If v' is a variable y that is not equal to x then we must have $y \in \text{dom}(\Gamma, \Gamma'[v/x])$ also from above we have $\Pi \vdash \Gamma, \Gamma'[v/x]$ **ws** and so using V_WS_VAR we have $\Pi; \Gamma, \Gamma'[v/x] \vdash v'$ **ws** and but since $v'[v/x] = v'$ we have the required conclusion.

   If v' is equal to x then we have $v'[v/x] = v$ and by the regularity lemma we already know $\Pi; \Gamma \vdash v$ **ws** and from weakening we have $\Pi; \Gamma, \Gamma[v/x] \vdash v$ **ws** which is the required conclusion.

   If v' is a literal then since $\Pi \vdash \Gamma, \Gamma'[v/x]$ **ws** we can use the appropriate rule to get $\Pi; \Gamma, \Gamma[v/x] \vdash v'$ **ws** and $v'[v/x] = v'$ we get the required conclusion.

   If v' is a pair (v_1, v_2) then from inversion of V_WS_PAIR we have

   $$\Pi; \Gamma, x : b[\phi[x/z]], \Gamma' \vdash v_1 \textbf{ ws}$$
   $$\Pi; \Gamma, x : b[\phi[x/z]], \Gamma' \vdash v_2 \textbf{ ws}$$

   and by the induction hypothesis we have

   $$\Pi; \Gamma, \Gamma'[v/x] \vdash v_1[v/x] \textbf{ ws}$$
   $$\Pi; \Gamma\Gamma'[v/x] \vdash v_2[v/x] \textbf{ ws}$$

   and so using V_WS_PAIR we have the required conclusion since $(v_1, v_2)[v/x] = (v_1[v/x], v_2[v/x])$

   If v' is the application of a data construction $v' = C\ v''$ then from inversion of V_WS_CONS we have

   $$\textbf{union}\ \text{tid}\ = \{\text{C}_1 : \tau_1, .., \text{C} : \tau, ...\text{C}_n : \tau_n\} \in \Pi$$
   $$\Pi; \Gamma, x : b[\phi[x/z]], \Gamma' \vdash v'' \textbf{ ws}$$

   applying the induction hypothesis to the second of these and then V_WS_CONS we obtain the required conclusion as $(C\ v'')[v/x] = C\ (v''[v/x])$.

3. For e the argument for the base case applies here as well.

4. For ϕ the argument for the base case applies here as well.

□

Lemma 39 (Substitution for well-scoped types).
If $\Pi;\Gamma \vdash v \Rightarrow \{z : b|\phi\}$ *and* $\Pi;\Gamma, x : b[\phi'], \Gamma' \vdash \tau$ ***ws*** *then* $\Pi;\Gamma, \Gamma'[v/x] \vdash \tau[v/x]$ ***ws***

Proof. Let $\tau = \{z : b|\phi\}$ where $z \notin \mathrm{dom}(\Gamma, x : b[\phi'], \Gamma')$. From the second premise and inversion of TYPE_WS_TYPE we have $\Pi;\Gamma, x : b[\phi'], \Gamma', z : b[\top] \vdash \phi$ **ws** . Applying Lemma 38 we have $\Pi;\Gamma, \Gamma'[v/x], z : b[\top] \vdash \phi[v/x]$ **ws** From TYPE_WS_TYPE we have $\Pi;\Gamma, \Gamma'[v/x] \vdash \{z : b|\phi[v/x]\}$ **ws** and the conclusion follows from the definition of substitution on types. □

Lemma 40 (Substitution for well-scoped Δ-contexts).
If

$$\begin{array}{c}\Pi;\Gamma \vdash v \Rightarrow \{z_1 : b_1|\phi_1\} \\ \Pi;\Gamma, x : b_1[\phi_1'], \Gamma' \vdash \Delta\ \textit{\textbf{ws}}\end{array}$$

then

$$\Pi;\Gamma, \Gamma'[v/x] \vdash \Delta[v/x]\ \textit{\textbf{ws}}$$

Proof. Induction on Δ. If Δ is empty, then the conclusion holds as the empty Δ-context is well-scoped under any Π-context and Γ-context by rule D_WS_EMPTY. If $\Delta = \Delta', u : \tau$, then from inversion of D_WS_CONS we have

$$\begin{array}{rcl}\Pi;\Gamma, x : b_1[\phi_1'], \Gamma' & \vdash & \Delta'\ \mathbf{ws} \\ \Pi;\Gamma, x : b_1[\phi_1'], \Gamma' & \vdash & \tau\ \mathbf{ws}\end{array}$$

From the induction hypothesis and the previous lemma we have:

$$\begin{array}{rcl}\Pi;\Gamma, \Gamma'[v/x] & \vdash & \Delta'[v/x]\ \mathbf{ws} \\ \Pi;\Gamma, \Gamma'[v/x] & \vdash & \tau[v/x]\ \mathbf{ws}\end{array}$$

Hence from rule D_WS_CONS and distributivity of substitution for Δ-context we have the required conclusion. □

9.7.2 Substitution for Typing Judgements

We prove a set of lemmas that guarantee that typing is stable under substitution. In particular, if we have a term that has a variable x and we substitute in a value that has a type that is a subtype of x's type then the type of the new term is a subtype of the original term. This is key to ensuring that type preservation occurs when reducing let expressions. These lemmas are more complex than the equivalent lemmas for simply typed languages as the substitution is also applied to types as these can include the variable we are substituting for.

Lemma 41 (Substitution for validity). *If*

$$\begin{array}{r}\Gamma \vdash v \Rightarrow \{z_1 : b_1|\phi_1\} \\ \Pi;\Gamma, x : b_1[\phi_1'], \Gamma' \models \phi \\ \Pi;\Gamma \vdash \{z_1 : b_1|\phi_1\} \lesssim \{x : b_1|\phi_1'\}\end{array}$$

then

$$\Pi;\Gamma, \Gamma'[v/x] \models \phi[v/x]$$

Proof. From Lemma 16 and the last two premises, we have

$$\Pi; \Gamma, x : b_1[\phi_1], \Gamma' \models \phi$$

Assume that there is an interpretation that makes $|\Gamma, \Gamma'[v/x]|$ true. Extend this to an interpretation that also maps the variable x to v. From Lemma 11 we have $\Gamma \models \phi_1[v/z_1]$ and so $|\Gamma, x : b_1[\phi_1]|$ is true and hence ϕ'_1 is true from the third premise. Hence ϕ is true under the extended interpretation from the second premise. Hence $\phi[v/x]$ is true under the original interpretation. □

Lemma 42 (Substitution for subtyping). *If*

$$\begin{aligned}\Gamma \vdash v \Rightarrow \{z_1 : b_1|\phi_1\}\\ \Pi; \Gamma, x : b_1[\phi'_1], \Gamma' \vdash \{z_2 : b_2|\phi_2\} \lesssim \{z_2 : b_2|\phi'_2\}\\ \Pi; \Gamma \vdash \{z_1 : b_1|\phi_1\} \lesssim \{x : b_1|\phi'_1\}\end{aligned}$$

then

$$\Pi; \Gamma, \Gamma'[v/x] \vdash \{z_2 : b_2|\phi_2\}[v/x] \lesssim \{z_2 : b_2|\phi'_2\}[v/x])$$

Proof. From the second premise using inversion of SUBTYPE_SUBTYPE we have the following

$$\begin{aligned}\Pi; \Gamma, x : b_1[\phi'_1], \Gamma' \vdash \{z_2 : b_2|\phi_2\} \ \mathbf{ws}\\ \Pi; \Gamma, x : b_1[\phi'_1], \Gamma' \vdash \{z_2 : b_2|\phi'_2\} \ \mathbf{ws}\\ \Pi; \Gamma, x : b_1[\phi'_1], \Gamma', z_2 : b_2[\phi_2] \models \phi'_2\end{aligned}$$

Applying Lemma 39 to the first two and the previous lemma to the third we get:

$$\begin{aligned}\Pi; \Gamma, \Gamma'[v/x] \vdash \{z_2 : b_2|\phi_2\}[v/x] \ \mathbf{ws}\\ \Pi; \Gamma, \Gamma'[v/x] \vdash \{z_2 : b_2|\phi'_2\}[v/x] \ \mathbf{ws}\\ \Pi; \Gamma, \Gamma'[v/x], z_2 : b_2[\phi_2[v/x]] \models \phi'_2[v/x]\end{aligned}$$

Finally using SUBTYPE_SUBTYPE we get the required conclusion making use of the definition of the substitution function for types. □

Lemma 43 (Substitution for $\Rightarrow$ values). *If*

$$\begin{aligned}\Gamma \vdash v \Rightarrow \{z_1 : b_1|\phi_1\}\\ \Pi; \Gamma, x : b_1[\phi'_1], \Gamma' \vdash v' \Rightarrow \{z_2 : b_2|\phi'_2\}\\ \Pi; \Gamma \vdash \{z_1 : b_1|\phi_1\} \lesssim \{x : b_1|\phi'_1\}\end{aligned}$$

then there is a ϕ_2 such that the following hold:

$$\begin{aligned}\Pi; \Gamma, \Gamma'[v/x] \vdash v'[v/x] \Rightarrow \{z_2 : b_2|\phi_2\}\\ \Pi; \Gamma, \Gamma'[v/x] \vdash \{z_2 : b_2|\phi_2\} \lesssim \{z_2 : b_2|\phi'_2\}[v/x])\end{aligned}$$

Proof. Proof is by cases on the structure of v'. For all cases $\phi_1 \equiv (z_1 = v')$ and $\phi_2 \equiv (z_2 = v'[v/x])$
Case $v' = ()$
Let $\phi_2 \equiv (z_2 = ())$
From inversion of INFER_V_UNIT with the second premise we have $\phi'_2 = (z_2 = ())$ and $b_2 = \mathbf{unit}$. From the substitution function we have $()[v/x] = ()$. Furthermore, for any Γ'' we have from INFER_V_UNIT that:

$$\Pi; \Gamma'' \vdash () \Rightarrow \{z : \mathbf{unit} \ |z = ()\}$$

and in particular we can pick $\Gamma'' = \Gamma, \Gamma'[v/x]$ and the first part of the conclusion holds:

$$\Pi; \Gamma, \Gamma'[v/x] \vdash ()[v/x] \Rightarrow \{z : b_2|\phi_2\}$$

The second part of the conclusion holds as for this case ϕ_2 and ϕ'_2 are identical.

Case $v' = x$
Let $\phi_2 \equiv (z_2 = v)$
From the substitution function we have that:

$$x[v/x] = v$$

and with the first premise, and using weakening to add $\Gamma'[v/x]$, we get the first part of the conclusion:

$$\Pi; \Gamma, \Gamma'[v/x] \vdash x[v/x] \Rightarrow \{z_2 : b_2 | z_2 = v\}$$

The second part of the conclusion is:

$$\Pi; \Gamma, \Gamma'[v/x] \vdash \{z_2 : b_2 | z_2 = v\} \lesssim (\{z_2 : b_2 | z_2 = x\}[v/x])$$

The constraints in both sides become $z_2 = v$ and so the conclusion follows.
Case $v' = w$ and $x \neq w$
Let $\phi_2 \equiv (z_2 = w)$
From the second premise we have that:

$$\Pi; \Gamma, x : b_1[\phi'_1], \Gamma' \vdash w \Rightarrow \{z_2 : b_2 | \phi'_2\}$$

Since v' is well-scoped, w is in the domain of Γ or Γ' and so from the rule INFER_V_VAR we have

$$\Pi; \Gamma, \Gamma'[v/x] \vdash w \Rightarrow \{z_2 : b_2 | z_2 = w\}$$

Then since $w[v/x] = w$ we have the first part of the conclusion

$$\Pi; \Gamma, \Gamma'[v/x] \vdash w[v/x] \Rightarrow \{z_2 : b_2 | z_2 = w\}$$

Furthermore, we have $\phi'_2 = (z_2 = w)$ and so the second part of the conclusion holds as well:

$$\Pi; \Gamma, \Gamma'[v/x] \vdash \{z_2 : b_2 | z_2 = w\} \lesssim (\{z_2 : b_2 | z_2 = w\}[v/x])$$

Case v' is an int or bool literal
We just give the proof for when $v' = n$; the proof for bool is similar.
Let $\phi_2 \equiv (z_2 = n)$
From the second premise and inversion of INFER_V_NUM we have:

$$\Pi; \Gamma, x : b_1[\phi'_1], \Gamma' \vdash n \Rightarrow \{z_2 : \textbf{int} \ | z_2 = n\}$$

From INFER_N_NUM we have

$$\Pi; \Gamma, \Gamma'[v/x] \vdash n \Rightarrow \{z_2 : \textbf{int} \ | z_2 = n\}$$

Since $n[v/x] = n$ we have the first part of the conclusion

$$\Pi; \Gamma, \Gamma'[v/x] \vdash n[v/x] \Rightarrow \{z_2 : \textbf{int} \ | z_2 = n\}$$

Furthermore, we have $\phi'_2[v/x] = (z_2 = n)$ and so the second part of the conclusion holds as well:

$$\Pi; \Gamma, \Gamma'[v/x] \vdash \{z_2 : \textbf{int} \ | z_2 = n\} \lesssim (\{z_2 : \textbf{int} \ | z_2 = n\}[v/x])$$

Case v' is (v_1, v_2)
Let $\phi_2 \equiv z_2 = (v_1[v/x], v_2[v/x])$
From inversion of INFER_V_PAIR on the second premise we have that there are b_{2i} and ϕ'_{2i} such that:

$$\Pi; \Gamma, x : b_1[\phi'_1], \Gamma' \vdash v_i \Rightarrow \{z_{2i} : b_{2i} | \phi'_{2i}\}, i = 1, 2$$

and that $\phi'_2 \equiv (z_2 = (v_1, v_2))$ and $b_2 = b_{21} * b_{22}$. From the induction hypothesis, there are ϕ_{21} and ϕ_{22} such that:

$$\begin{array}{rcl} \Pi; \Gamma, \Gamma'[v/x] & \vdash & v_i[v/x] \Rightarrow \{z_{2i} : b_{2i} | \phi_{2i}\} \\ \Pi; \Gamma, \Gamma'[v/x] & \vdash & \{z_{2i} : b_{2i} | \phi_{2i}\} \lesssim \{z_{2i} : b_{2i} | \phi'_{2i}\}[v/x]), i = 1, 2 \end{array}$$

And using INFER_V_PAIR the first part of the conclusion holds:

$$\Pi; \Gamma, \Gamma'[v/x] \vdash (v_1, v_2)[v/x] \Rightarrow \{z_2 : b_{21} * b_{22} \mid z_2 = (v_1[v/x], v_2[v/x])\ \}$$

as well as the second:

$$\Pi; \Gamma, \Gamma'[v/x] \vdash \{z_2 : b_2 | z_2 = (v_1[v/x], v_2[v/x])\} \lesssim (\{z_2 : b_2 | z_2 = (v_1, v_2)\}[v/x])$$

Case v' is $C\ v''$
Let $\phi_2 \equiv (z_2 = C\ (v''[v/x]))$
From the second premise and inversion of INFER_V_DATA_CONS we have that there is a *tid* and k such that $C_k = C$ and

$$\begin{array}{r} \textbf{union}\ \ \text{tid}\ = \{\mathrm{C}_1 : \{z''_1 : b''_1 | \phi''_1\} .. \mathrm{C}_n : \{z''_n : b''_n | \phi''_n\}\} \in \Pi \\ \Pi; \Gamma, x : b_1[\phi'_1], \Gamma' \vdash C\ v'' \Rightarrow \{z_2 : \text{tid} | z_2 = C\ v''\} \\ \Pi; \Gamma, x : b_1[\phi'_1], \Gamma' \vdash v'' \Leftarrow \{z''_k : b''_k | \phi''_k\} \end{array}$$

From the last of these we have that there is a ϕ'' such that:

$$\Pi; \Gamma, x : b_1[\phi'_1], \Gamma' \vdash v'' \Rightarrow \{z : b''_k | \phi''\} \quad (1)$$
$$\Pi; \Gamma, x : b_1[\phi'_1], \Gamma' \vdash \{z : b''_k | \phi''\} \lesssim \{z''_k : b''_k | \phi''_k\} \quad (2)$$

Applying the induction hypothesis to the first of these we have that there is ϕ''' such that:

$$\Pi; \Gamma, \Gamma'[v/x] \vdash v''[v/x] \Rightarrow \{z : b''_k | \phi'''\} \quad (3)$$
$$\Pi; \Gamma, \Gamma'[v/x] \vdash \{z : b''_k | \phi'''\} \lesssim \{z : b''_k | \phi''\}[v/x] \quad (4)$$

Using Lemma 17 on (2) we have:

$$\Pi; \Gamma, \Gamma'[v/x] \vdash \{z : b''_k | \phi''\}[v/x] \lesssim \{z''_k : b''_k | \phi''_k\}[v/x] \quad (5)$$

and so by transitivity of subtyping we have

$$\Pi; \Gamma, \Gamma'[v/x] \vdash \{z : b''_k | \phi'''\}[v/x] \lesssim \{z''_k : b''_k | \phi''_k\}[v/x] \quad (6)$$

and thus

$$\Pi; \Gamma, \Gamma'[v/x] \vdash v''[v/x] \Leftarrow \{z''_k : b''_k | \phi''_k\}[v/x]$$

From well-scoping of the **union** definitions, we know that x will not appear in ϕ''_k and so we have:

$$\Pi; \Gamma, \Gamma'[v/x] \vdash v''[v/x] \Leftarrow \{z : b''_k | \phi''_k\}$$

We can now apply INFER_V_DATA_CONS to get:

$$\Pi; \Gamma, \Gamma'[v/x] \vdash C\ v'' \Rightarrow \{z : \mathrm{tid} | z = C\ (v''[v/x])\}$$

which is the first part of the required conclusion. The second part has the form:

$$\Pi; \Gamma, \Gamma'[v/x] \vdash \{z : \mathrm{tid} | z = C\ (v''[v/x])\} \lesssim \{z : \mathrm{tid} | z = C\ v'')[v/x]\}$$

which holds as both types in the subtype expression are equal.

□

Lemma 44 (Substitution for $\Leftarrow$ values)**.** *If*

$$\begin{aligned} \Gamma \vdash v \Rightarrow \{z_1 : b_1 | \phi_1\} \\ \Pi; \Gamma, x : b_1[\phi_1'[x/z_2]], \Gamma' \vdash v' \Leftarrow \{z_2 : b_2 | \phi_2'\} \\ \Pi; \Gamma \vdash \{z_1 : b_1 | \phi_1\} \lesssim \{z_1 : b_1 | \phi_1'\} \end{aligned}$$

then

$$\Pi; \Gamma, \Gamma'[v/x] \vdash v'[v/x] \Leftarrow \{z_2 : b_2 | \phi_2'\}[v/x]$$

Proof. From the second premise and inversion of CHECK_V we have that there is a ϕ_2'' such that

$$\begin{aligned} \Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma' \vdash v' \Rightarrow \{z_2 : b_2 | \phi_2''\} \\ \Pi; \Gamma, x : b_1[\phi_1'[x/z_1]], \Gamma' \vdash \{z_2 : b_2 | \phi_2''\} \lesssim \{z_2 : b_2 | \phi_2'\} \end{aligned}$$

Applying the previous lemma to the first of these we have that there is a ϕ_2''' such that the following hold:

$$\begin{aligned} \Pi; \Gamma, \Gamma'[v/x] \vdash v'[v/x] \Rightarrow \{z_2 : b_2 | \phi_2'''\} \\ \Pi; \Gamma, \Gamma'[v/x] \vdash \{z_2 : b_2 | \phi_2'''\} \lesssim \{z_2 : b_2 | \phi_2''\}[v/x] \end{aligned}$$

We also have:

$$\Pi; \Gamma, \Gamma'[v/x] \vdash \{z_2 : b_2 | \phi_2''\}[v/x] \lesssim \{z_2 : b_2 | \phi_2'\}[v/x]$$

and so from transitivity we have

$$\Pi; \Gamma, \Gamma'[v/x] \vdash \{z_2 : b_2 | \phi_2'''\}[v/x] \lesssim \{z_2 : b_2 | \phi_2'\}[v/x]$$

and applying CHECK_V we have the required conclusion. □

Lemma 45 (Substitution for $\Rightarrow$ expression)**.**
If

$$\begin{aligned} \Pi; \Gamma \vdash v \Rightarrow \{z_1 : b_1 | \phi_1\} \\ \Pi; \Gamma, x : b_1[\phi_1'], \Gamma'; \Delta \vdash e \Rightarrow \{z_2 : b_2 | \phi_2'\} \\ \Pi; \Gamma \vdash \{z_1 : b_1 | \phi_1\} \lesssim \{x : b_1 | \phi_1'\} \end{aligned}$$

then there is a ϕ_2 such that the following hold:

$$\begin{aligned} \Pi; \Gamma, \Gamma'[v/x]; \Delta[v/x] \vdash e[v/x] \Rightarrow \{z_2 : b_2 | \phi_2\} \\ \Pi; \Gamma[v/x] \vdash \{z_2 : b_2 | \phi_2\} \lesssim \{z_1 : b_2 | \phi_2'\}[v/x] \end{aligned}$$

Proof. The proof is by cases on the structure of e. For all cases we have $\phi_1 \equiv z = v$.

Case $e = v_1 + v_2$
Let $\phi_2 \equiv (z = (v_1 + v_2)[v/x]) \equiv \phi_2'[v/x]$
We need to show:

$$\begin{array}{r}\Pi; \Gamma, \Gamma'[v/x]; \Delta[v/x] \vdash (v_1 + v_2)[v/x] \Rightarrow \{z : \mathbf{int} | z = (v_1 + v_2)[v/x]\} \\ \Pi; \Gamma, \Gamma'[v/x] \vdash \{z_2 : \mathbf{int} | (z = v_1 + v_2)[v/x]\} \lesssim \{z_2 | \phi_2'\}[v/x]\end{array}$$

From the second premise and inversion on INFER_E_PLUS we have that there exist ϕ_{21}' and ϕ_{22}' such that $b_2 = \mathbf{int}$ and $\phi_2' \equiv (z = v_1 + v_2)$ and the following holds:

$$\begin{array}{r}\Gamma, x : b_1[\phi_1'], \Gamma' \vdash \Delta \ \mathbf{ws} \\ \Pi; \Gamma, x : b_1[\phi_1'], \Gamma' \vdash v_1 \Rightarrow \{w_1 : \mathbf{int} | \phi_{21}'\} \\ \Pi; \Gamma, x : b_1[\phi_1'], \Gamma' \vdash v_2 \Rightarrow \{w_2 : \mathbf{int} | \phi_{22}'\}\end{array}$$

Using Lemma 43 on the last two we have there are ϕ_{21} and ϕ_{22} such that:

$$\begin{array}{c}\Pi; \Gamma, \Gamma'[v/x] \vdash v_1[v/x] \Rightarrow \{w_1 : \mathbf{int} | \phi_{21}\} \\ \Pi; \Gamma, \Gamma'[v/x] \vdash v_2[v/x] \Rightarrow \{w_2 : \mathbf{int} | \phi_{22}\}\end{array}$$

and from Lemma 40 on the first we have

$$\Gamma, \Gamma'[v/x] \vdash \Delta[v/x] \ \mathbf{ws}$$

and so using INFER_E_PLUS we have:

$$\Pi; \Gamma, \Gamma'[v/x]; \Delta[v/x] \vdash (v_1[v/x]) + (v_2[v/x]) \Rightarrow \{z : \mathbf{int} | (z = v_1[v/x] + v_2[v/x])\}$$

and pulling out the substitutions from the type and expression, we have the first part of the conclusion:

$$\Pi; \Gamma, \Gamma'[v/x]; \Delta[v/x] \vdash (v_1 + v_2)[v/x] \Rightarrow \{z : \mathbf{int} | z = (v_1 + v_2)[v/x]\}$$

The second part of the conclusion has the form:

$$\Pi; \Gamma, \Gamma'[v/x] \vdash \{z_2 : \mathbf{int} | ((z = v_1 + v_2)[v/x]\} \lesssim \{z_2 : \mathbf{int} | z = v_1 + v_2\}[v/x]$$

which holds and so the required conclusion follows.

Case $e = v_1 \leq v_2$
Similar to the previous case with $\phi_2 \equiv (z = v_1 \leq v_2)[v/x]$

Case $e = f \ v'$
We pick as the ϕ_2 the constraint in the return type of the function f, so assume that f has type:

$$y : b[\phi] \rightarrow \{z_2 : b_2 | \phi_2\}$$

Taking the second lemma premise and using rule inversion with INFER_ANF_APP, we have

$$\Pi; \Gamma, x : b_1[\phi_1'], \Gamma' \vdash v' \Leftarrow \{z : b | \phi[z/y]\} \tag{7}$$

$$\Pi; \Gamma, x : b_1[\phi_1'], \Gamma' \vdash \Delta \ \mathbf{ws} \tag{8}$$

$$\phi_2 = \phi_2' \tag{9}$$

Apply the substitution lemma for values to the first of these, we get:

$$\Pi; \Gamma, \Gamma'[v/x] \vdash v'[v/x] \Leftarrow \{z : b | \phi'[z/y]\}[v/x] \tag{10}$$

From the well-scoping of function definitions DEF_WS_VALSPEC we have that x cannot occur in ϕ' and so we have $\{z : b|\phi'[z/y]\}[v/x] = \{z : b|\phi'[z/y]\}$. Hence with Lemma 40 we can apply INFER_E_APP to get:

$$\Pi; \Gamma, \Gamma'[v/x], \Delta[v/x] \vdash f(v'[v/x]) \Rightarrow \{z_2 : b_2|\phi_2\} \tag{11}$$

Again from DEF_WS_VALSPEC, x doesnot appear in the return type of the function and so using substitution for function application, we have the first part of the conclusion

$$\Pi; \Gamma, \Gamma'[v/x], \Delta[v/x] \vdash (f v')[v/x]) \Rightarrow \{z_2 : b_2|\phi_2\}[v/x] \tag{12}$$

The second part of the conclusion also holds through reflexivity of subtyping.

Case $e = u$
Let $\phi_2 = \phi_2'[v/x]$
From the second premise using inversion of INFER_E_MVAR we have:

$$\Gamma, x : b_1[\phi_1], \Gamma' \vdash \Delta \ \mathbf{ws} \tag{1}$$
$$(u, \{z_2 : b_2|\phi_2'\}) \in \Delta \tag{2}$$

From Lemma 40 we have

$$\Gamma, \Gamma'[x/v] \vdash \Delta[v/x] \ \mathbf{ws} \tag{3}$$

Furthermore, from the definition of substitution for Δ we have

$$(u, \{z_2 : b_2|\phi_2'[v/x]\}) \in \Delta[v/x] \tag{4}$$

So using (3) and (4) with INFER_E_MVAR we have:

$$\Pi; \Gamma, \Gamma'[v/x]; \Delta[v/x] \vdash u \Rightarrow \{z_2 : b_2|\phi_2'[v/x]\}$$

From the definition of substitution for mutable variables we have that $u[v/x] = u$ and so the first part of the conclusion holds. The second part holds since $\phi_2 = \phi_2'[v/x]$

Case $e = \mathbf{fst}\ v'$
Using inversion of INFER_E_FST on the first premise we have that there is a ϕ' and b_2'

$$\Pi; \Gamma, x : b_1[\phi_1[x/z_1]], \Gamma' \vdash v' \Rightarrow \{z_2 : b_2 * b_2'|\phi'\}$$
$$\{z_2 : b_2|\phi_2'\} = \{z_2 : b_2|z = \mathbf{fst}\ v'\}$$

Applying Lemma 44 we have that there is a ϕ'' such that

$$\Pi; \Gamma, \Gamma'[v/x] \vdash v'[v/x] \Rightarrow \{z_2 : b_2 * b_2'|\phi''\}$$

and applying INFER_E_FST to this we have:

$$\Pi; \Gamma, \Gamma'[v/x] \vdash \mathbf{fst}\ v'[v/x] \Rightarrow \{z_2 : b_2|z = \mathbf{fst}\ v'[v/x]\}$$

Hence the first part of the conclusion holds after moving the substitution out of the type. The second part holds since since $\phi_2 = \phi_2'[v/x]$.

Case $e = \mathbf{snd}\ v'$
Similar to the case for **fst** above.

□

Lemma 46 (Substitution for $\Leftarrow$ expressions)**.** *If*

$$\Pi;\Gamma\vdash v \Rightarrow \{z_1 : b_1|\phi_1\}$$
$$\Pi;\Gamma, x : b_1[\phi_1'],\Gamma';\Delta\vdash e \Leftarrow \{z : b_2|\phi_2'\}$$
$$\Pi;\Gamma\vdash \{z_1 : b_1\phi_1\} \lesssim \{z_1 : b_1\phi_1'\}$$

then the following holds:

$$\Pi;\Gamma,\Gamma'[v/x];\Delta[v/x]\vdash e[v/x] \Leftarrow \{z : b_2|\phi_2'\}[v/x]$$

Proof. From the second premise and inversion of CHECK_E we have that there is a ϕ_2'' such that

$$\Pi;\Gamma, x : b_1[\phi_1'],\Gamma';\Delta\vdash e \Rightarrow \{z : b_2|\phi_2''\} \tag{1}$$
$$\Pi;\Gamma, x : b_1[\phi_1'],\Gamma'\vdash \{z : b_2|\phi_2''\} \lesssim \{z : b_2|\phi_2'\} \tag{2}$$

Applying the previous lemma to the first of these we have that there is a ϕ_2''' such that the following hold:

$$\Pi;\Gamma,\Gamma'[v/x];\Delta[v/x]\vdash e[v/x] \Rightarrow \{z : b_2|\phi_2'''\} \tag{3}$$
$$\Pi;\Gamma,\Gamma'[v/x]\vdash \{z : b_2|\phi_2'''\} \lesssim \{z : b_2|\phi_2''\}[v/x] \tag{4}$$

Applying Lemma 42 to (2) we have:

$$\Pi;\Gamma,\Gamma'[v/x]\vdash \{z : b_2|\phi_2''\}[v/x] \lesssim \{z : b_2|\phi_2'\}[v/x]$$

and so from transitivity we have

$$\Pi;\Gamma,\Gamma'[v/x]\vdash \{z : b_2|\phi_2'''\} \lesssim \{z : b_2|\phi_2'\}[v/x]$$

and applying CHECK_E we have the required conclusion.

□

Note that in the following lemma there is no implication statement in the conclusion as the type of s is already overspecified.

Lemma 47 (Substitution for $\Leftarrow$ statements)**.** *If*

$$\Pi;\Gamma\vdash v \Rightarrow \{z_1 : b_1|\phi_1\}$$
$$\Pi;\Gamma, x : b_1[\phi_1'],\Gamma';\Delta]\vdash s \Leftarrow \{z : b_2|\phi_2'\}$$
$$\Pi;\Gamma\vdash \{z_1 : b_1\phi_1\} \lesssim \{z_1 : b_1\phi_1'\}$$

then

$$\Pi;\Gamma,\Gamma'[v/x];\Delta[v/x]\vdash s[v/x] \Leftarrow \{z : b_2|\phi_2'\}[v/x]$$

Proof. Proceed by induction on the structure of s:

- *Case* $s \equiv v'$
  From the second premise, we have using inversion with CHECK_ANF_VAL that

  $$\Pi;\Gamma, x : b_1[\phi_1'],\Gamma'\vdash \Delta \ \mathbf{ws} \tag{1}$$
  $$\Pi;\Gamma, x : b_1[\phi_1'],\Gamma'\vdash v' \Leftarrow \{z : b_2|\phi_2'\} \tag{2}$$

  Applying the substitution lemma for checking the type of values to the second of these we have

  $$\Pi;\Gamma,\Gamma'[v/x]\vdash v' \Leftarrow \{z : b_2|\phi_2'\}[v/x] \tag{3}$$

  Applying Lemma **??** to (1) and the rule CHECK_ANF_VAL to (3) we get the required conclusion:

  $$\Pi;\Gamma,\Gamma'[v/x];\Delta[v/x]\vdash v' \Leftarrow \{z_2 : b_2|\phi_2'\}[v/x]$$

- *Case* $s \equiv \textbf{if } v' \textbf{ then } s_1 \textbf{ else } s_2$
  From the second premise, we have using inversion with CHECK_ANF_IF that there is a ϕ_3 such that:

$$\begin{aligned}
\Pi;\Gamma,x:b_1[\phi_1'],\Gamma' &\vdash v' \Rightarrow \{z_3:\textbf{bool}|\phi_3\}\\
\Pi;\Gamma,x:b_1[\phi_1'],\Gamma';\Delta &\vdash s_1 \Leftarrow \{z_2:b_2|v'=\textbf{T}\wedge\phi_3[v'/z_3]\implies\phi_2'\}\\
\Pi;\Gamma,x:b_1[\phi_1'],\Gamma';\Delta &\vdash s_2 \Leftarrow \{z_2:b_2|v'=\textbf{F}\wedge\phi_3[v'/z_3]\implies\phi_2'\}
\end{aligned}$$

  We apply the substitution lemma for values to the first of these and the induction hypothesis to the next two to obtain that there is ϕ_3' such that:

$$\begin{aligned}
\Pi;\Gamma,\Gamma'[v/x] &\vdash v'[v/x]\Rightarrow\{z:\textbf{bool}|\phi_3'\} && (1)\\
\Pi;\Gamma,\Gamma'[v/x];\Delta[v/x] &\vdash s_1[v/x]\Leftarrow\{z_2:b_2|v'=\textbf{T}\wedge\phi_3[v'/z_3]\implies\phi_2'\}[v/x] && (2)\\
\Pi;\Gamma,\Gamma'[v/x];\Delta[v/x] &\vdash s_2[v/x]\Leftarrow\{z_2:b_2|v'=\textbf{F}\wedge\phi_3[v'/z_3]\implies\phi_2'\}[v/x] && (3)\\
\Pi;\Gamma,\Gamma'[v/x] &\vdash \{z:\textbf{bool}|\phi_3'\}\lesssim\{z:\textbf{bool}|\phi_3\}[v/x] && (4)
\end{aligned}$$

  From (4) we have since implication is contravariant on subtyping that:

$$\Pi;\Gamma\vdash\{z_2:b_2|v'=\textbf{T}\wedge(\phi_3[v'/z_3])\implies\phi_2'\}[v/x]\lesssim\{z_2:b_2|v=\textbf{T}\wedge(\phi_3'[v'/z_3])\implies\phi_2'[v/x]\}\quad(5)$$
$$\Pi;\Gamma\vdash\{z_2:b_2|v'=\textbf{T}\wedge(\phi_3[v'/z_3])\implies\phi_2\}[v/x]\lesssim\{z_2:b_2|v=\textbf{F}\wedge(\phi_3'[v'/z_3])\implies\phi_2'[v/x]\}\quad(6)$$

  Note that we have applied some rewriting of the substitutions making use of the fact that x does not occur in ϕ_3' or v. Using (5) and (6) with Lemma 12 to (2) and (3) we get:

$$\begin{aligned}
\Pi;\Gamma,\Gamma'[v/x];\Delta[v/x] &\vdash s_1[v/x]\Leftarrow\{z_2:b_2|v'=\textbf{T}\wedge\phi_3'[v'/z_3]\implies\phi_2'[v/x]\} && (7)\\
\Pi;\Gamma,\Gamma'[v/x];\Delta[v/x] &\vdash s_2[v/x]\Leftarrow\{z_2:b_2|v'=\textbf{F}\wedge\phi_3'[v'/z_3]\implies\phi_2'[v/x]\} && (8)
\end{aligned}$$

  Using CHECK_ANF_IF with (1), (7) and (8) we get the required conclusion:

$$\Pi;\Gamma,\Gamma'[v/x];\Delta[v/x]\ \vdash\ \textbf{if } v'[v/x] \textbf{ then } s_1[v/x] \textbf{ else } s_2[v/x]\Leftarrow\{z_2:b_2|\phi_2'\}[v/x]$$

- *Case* $s \equiv \textbf{let } w = e \textbf{ in } s$
  If $w = x$ then the substitution is the identity and we are done. Assume we have $x \neq w$ and w is not free in v. Using inversion with CHECK_ANF_LET there is a b_3 and ϕ_3 such that:

$$\begin{aligned}
\Pi;\Gamma,x:b_1[\phi_1'],\Gamma';\Delta\vdash e\Rightarrow\{z:b_3|\phi_3\}\\
\Pi;\Gamma,x:b_1[\phi_1'],\Gamma',w:b_3[\phi_3[w/z]];\Delta\vdash s\Leftarrow\{z:b_2|\phi_2'\}
\end{aligned}$$

  Applying the substitution lemma for expressions to the first and the induction hypothesis to the second we obtain a ϕ_3' such that:

$$\begin{aligned}
\Pi;\Gamma,\Gamma'[v/x];\Delta[v/x] &\vdash e\Rightarrow\{z:b_3|\phi_3'\} && (1)\\
\Pi;\Gamma,\Gamma'[v/x] &\vdash \{z:b_3|\phi_3'\}\lesssim\{z:b_3|\phi_3\}[v/x] && (2)\\
\Pi;\Gamma,\Gamma'[v/x],w:b_3[\phi_3[w/z][v/x]];\Delta[v/x] &\vdash s[v/x]\Leftarrow\{z:b_2|\phi_2'\}[v/x] && (3)
\end{aligned}$$

  With Lemma 22 and (2) and (3) we have:

$$\Pi;\Gamma,\Gamma'[v/x],w:b_3[\phi_3'[w/z]];\Delta[v/x]\vdash s[v/x]\Leftarrow\{z:b_2|\phi_2'\}[v/x]$$

  and this with (1) and the rule CHECK_S_LET gives the required conclusion.

- *Case* $s \equiv \textbf{let}\ w : \tau = s_1\ \textbf{in}\ s_2$
  As above, we just need to consider the case where $x \neq w$ and w is not free in v. Let $\tau = \{z : b_3 | \phi_3\}$. Using inversion with CHECK_ANF_LET2 we obtain the following:

$$\begin{array}{rcl} \Pi; \Gamma, x : b_1[\phi_1'], \Gamma'; \Delta & \vdash & s_1 \Leftarrow \{z_3 : b_3 | \phi_3\} \\ \Pi; \Gamma, x : b_1[\phi_1'], \Gamma', w : b_3[\phi_3[w/z_3]]; \Delta & \vdash & s_2 \Leftarrow \{z_2 : b_2 | \phi_2'\} \end{array}$$

  Applying the induction hypothesis to these we get:

$$\begin{array}{r} \Pi; \Gamma, \Gamma'[v/x]; \Delta[v/x] \vdash s_1[v/x] \Leftarrow \{z_3 : b_3 | \phi_3\}[v/x] \\ \Pi; \Gamma, \Gamma'[v/x], w : b_3[\phi_3[w/z_3][v/x]]; \Delta[v/x] \vdash s_2[v/x] \Leftarrow \{z_2 : b_2 | \phi_2'\}[v/x] \end{array}$$

  We then use CHECK_ANF_LET2 to obtain the conclusion:

$$\Pi; \Gamma, \Gamma'[v/x]; \Delta[v/x] \vdash \textbf{let}\ z : \tau[v/x] = s_1[v/x]\ \textbf{in}\ s_2[v/x] \Leftarrow \{z_2 : b_2 | \phi_2'\}$$

- *Case* $s \equiv \textbf{match}\ v'\ \{\ \mathrm{C}_1\ x_1 \Rightarrow s_1\ ,\ ..\ ,\ \mathrm{C}_n\ x_n \Rightarrow s_n\}$
  From the second premise and rule inversion of check_s_match we have that there is an id such that:

$$\begin{array}{rcl} \textbf{union}\ \ \mathrm{id} & = & \{\mathrm{cons}_1 : \{z_1 : b_1'' | \phi_1''\}\ \ ..\ \ \mathrm{cons}_n : \{z_n : b_n'' | \phi_n''\}\} \in \Pi \\ \Pi; \Gamma, x : b_1[\phi_1'], \Gamma' & \vdash & v' \Leftarrow \{z : id | \top\} \\ \Pi; \Gamma, x : b_1[\phi_1'], \Gamma', x_i : b_i''[\phi_i''[x_i/z_i]]; \Delta & \vdash & s_i \Leftarrow \{z_2 : b_2 | \phi_2'\}\ \ i = 1..n \end{array}$$

  Applying the induction hypothesis to the second of these and the n instances of the third, we get:

$$\begin{array}{rcl} \Pi; \Gamma, \Gamma'[v/x] & \vdash & v'[v/x] \Leftarrow \{z : id | \top\} \\ \Pi; \Gamma, \Gamma'[v/x], x_i : b_i''[\phi_i''[x_i/z_i][v/x]]; \Delta[v/x] & \vdash & s_i[v/x] \Leftarrow \{z_2 : b_2 | \phi_2'\}[v/x]\ \ i = 1..n \end{array}$$

  The well-scoping conditions on type definitions tells us that the $\{z_i | b_i'' | \phi_i''\}$ contain no free variables, hence the last judgement of the above is equivalent to:

$$\Pi; \Gamma, \Gamma'[v/x], x_i : b_i''[\phi_i''[x_i/z_i]]; \Delta[v/x] \vdash s_i[v/x] \Leftarrow \{z_2 : b_2 | \phi_2'\}\ \ i = 1..n$$

  Hence we can use CHECK_S_MATCH to get:

$$\begin{array}{r} \Pi; \Gamma, \Gamma'[v/x], \Delta \vdash \textbf{match}\ v'[v/x]\ \{\ \mathrm{C}_1\ x_1 \Rightarrow s_1[v/x]\ ,\ ..\ ,\ \mathrm{C}_n\ x_n \Rightarrow s_n[v/x]\} \\ \Leftarrow \{z_2 : b_2 | \phi_2'\}[v/x] \end{array}$$

  and since

  $\textbf{match}\ v'[v/x]\ \{\ \mathrm{C}_1\ x_1 \Rightarrow s_1[v/x]\ ,\ ..\ ,\ \mathrm{C}_n\ x_n \Rightarrow s_n[v/x]\} = (\textbf{match}\ v'\ \{\ \mathrm{C}_1\ x_1 \Rightarrow s_1\ ,\ ..\ ,\ \mathrm{C}_n\ x_n \Rightarrow s_n\})$

  we get the required conclusion:

$$\begin{array}{r} \Pi; \Gamma, \Gamma'[v/x], \Delta \vdash \textbf{match}\ v'\ \{\ \mathrm{C}_1\ x_1 \Rightarrow s_1\ ,\ ..\ ,\ \mathrm{C}_n\ x_n \Rightarrow s_n\}[v/x] \\ \Leftarrow \{z_2 : b_2 | \phi_2'\}[v/x] \end{array}$$

- *Case* $s \equiv \textbf{while}\ (s_1)\ \ \textbf{do}\ \{s_2\}$
  From the premises and inversion of CHECK_S_WHILE we have:

$$\begin{array}{rcl} \Pi; \Gamma, x : b_1[\phi_1'], \Gamma'; \Delta & \vdash & s_1 \Leftarrow \{z : \textbf{bool} | \top\} \\ \Pi; \Gamma, x : b_1[\phi_1'], \Gamma'; \Delta & \vdash & s_2 \Leftarrow \{z : \textbf{unit} | \top\} \\ \{z : \textbf{unit} | \top\} & = & \{z_2 : b_2 | \phi_2\} \end{array}$$

Applying the induction hypothesis to both of these we have:

$$\begin{aligned}\Pi;\Gamma,\Gamma'[v/x];\Delta[v/x] &\vdash s_1[v/x] \Leftarrow \{z:\mathbf{bool}|\top\}[v/x]\\ \Pi;\Gamma,\Gamma'[v/x];\Delta[v/x] &\vdash s_2[v/x] \Leftarrow \{z:\mathbf{unit}|\top\}[v/x]\end{aligned}$$

The substitution on the types is the identity and so from CHECK_S_WHILE we have:

$$\Pi;\Gamma,\Gamma'[v/x];\Delta \vdash \mathbf{while}\ (s_1[v/x])\ \ \mathbf{do}\ \{s_2[v/x]\} \Leftarrow \{z:\mathbf{unit}|\top\}$$

and since

$$\mathbf{while}\ (s_1)\ \ \mathbf{do}\ \{s_2\}[v/x] = \mathbf{while}\ (s_1[v/x])\ \ \mathbf{do}\ \{s_2[v/x]\}$$

the required conclusion follows.

- *Case* $s \equiv \mathbf{var}\ u\ \ : \tau = v'\ \mathbf{in}\ s_1$
  From the premises and inversion of CHECK_S_VAR we have:

$$\begin{aligned}\Pi;\Gamma,x:b_1[\phi_1'],\Gamma' &\vdash v' \Leftarrow \tau\\ \Pi;\Gamma,x:b_1[\phi_1'],\Gamma';\Delta,(u,\tau) &\vdash s_1 \Leftarrow \{z_2:b_2|\phi_2'\}\end{aligned}$$

  Applying the substitution lemma for checking values to the first of these and the induction hypothesis to the second, we get:

$$\begin{aligned}\Pi;\Gamma,\Gamma'[v/x];\Delta[v/x] &\vdash v'[v/x] \Leftarrow \tau[v/x]\\ \Pi;\Gamma,\Gamma'[v/x];\Delta[v/x],(u,\tau[v/x]) &\vdash s_1[v/x] \Leftarrow \{z_2:b_2|\phi_2'\}[v/x]\end{aligned}$$

  after pushing the substitution into the type of the second of these we get when applying CHECK_S_VAR:

$$\Pi;\Gamma,\Gamma'[v/x];\Delta \vdash \mathbf{var}\ u\ \ : \tau[v/x] = v'[v/x]\ \mathbf{in}\ s_1[v/x] \Leftarrow \{z_2:b_2|\phi_2'[v/x]\}$$

  since

$$\mathbf{var}\ u\ \ : \tau[v/x] = v'[v/x]\ \mathbf{in}\ s_1[v/x] = (\mathbf{var}\ u\ \ : \tau = v'\ \mathbf{in}\ s_1)[v/x]$$

  the conclusion holds after pulling the substitution out of the type.

- *Case* $s \equiv u := v'$
  From the premises and inversion of CHECK_S_ASSIGN we have that there is a τ such that:

$$(u,\tau) \in \Delta \tag{1}$$
$$\Pi;\Gamma,x:b_1[\phi_1'],\Gamma' \vdash v' \Leftarrow \tau \tag{2}$$
$$\{z_2:b_2|\phi_2\} = \{z_2:\mathbf{unit}|z_2 = ()\} \tag{3}$$

  Applying the induction hypothesis to the second of these gives:

$$\Pi;\Gamma,\Gamma'[v/x] \vdash v'[v/x] \Leftarrow \tau[v/x] \tag{4}$$

  From the definition of substitution into Δ we have $(u,\tau[v/x]) \in \Delta[v/x]$ and so can use (4) to get:

$$\Pi;\Gamma,\Gamma'[v/x];\Delta[v/x] \vdash u := v'[v/x] \Leftarrow \{z_2:\mathbf{unit}|z = ()\}$$

  and thus the required conclusion:

$$\Pi;\Gamma,\Gamma'[v/x];\Delta[v/x] \vdash (u := v')[v/x] \Leftarrow \{z_2:\mathbf{unit}|z = ()\}[v/x]$$

  since substitution into the type is the identity.

- *Case* $s \equiv s_1; s_2$
  From the premises and inversion of CHECK_S_SEQ we have:

$$\begin{array}{rcl} \Gamma, x : b_1[\phi'_1], \Gamma'; \Delta & \vdash & s_1 \Leftarrow \{z : \mathbf{unit} | z = ()\} \\ \Gamma, x : b_1[\phi'_1], \Gamma'; \Delta & \vdash & s_2 \Leftarrow \{z : b_2 | \phi'_2\} \end{array}$$

  Applying the induction hypothesis to both of these we have:

$$\begin{array}{rcl} \Gamma, \Gamma'[v/x]; \Delta[v/x] & \vdash & s_1[v/x] \Leftarrow \{z : unit | \top\} \\ \Gamma, \Gamma'[v/x]; \Delta[v/x] & \vdash & s_2[v/x] \Leftarrow \{z : b_2 | \phi'_2[v/x]\} \end{array}$$

  From CHECK_S_SEQ we have:

$$\Gamma, \Gamma'[v/x]; \Delta \vdash s_1[v/x]; s_2[v/x] \Leftarrow \{z : b_2 | \phi'_2[v/x]\}$$

  Since

$$(s_1; s_2)[v/x] = s_1[v/x]; s_2[v/x]$$

  the required conclusion follows after pulling the substitution out of the type.

□

10 Safety

The operational semantics of MiniSail is stated in terms of the reduction of a pair $\langle \delta, s \rangle$ where δ is the mutable variable store and s is a statement. In the preservation lemma that follows not only do we need to ensure that the type of the statement is preserved under reduction but that the types of the values in the mutable variable store do not diverge from what is specified in the current Δ context. We therefore formulate a program-state typing judgement and give the progress and preservation lemmas in terms of this judgement. We also formulate a judgement to specify that the types of the values associated to variables in the mutable variable store is compatible with the types for the variables given in Δ. This judgement and the mutable variable store check are defined as follows:

$\boxed{\Pi; \Delta \vdash (\delta, s) \Leftarrow \tau}$ Program state typing judgement

$$\frac{\begin{array}{c} \Pi \vdash \Delta \sim \delta \\ \Pi; \cdot; \Delta \vdash s \Leftarrow \tau \end{array}}{\Pi; \Delta \vdash (\delta, s) \Leftarrow \tau} \quad \text{CHECK\_REDEX\_STMT}$$

$\boxed{\Pi \vdash \Delta \sim \delta}$

$$\frac{\begin{array}{l} \delta = u_1 \to v_1, \,..\,, u_n \to v_n \\ \Delta = u_1 : \tau_1, \,..\,, u_n : \tau_n \\ \Pi; \cdot \vdash v_1 \Leftarrow \tau_1 \quad .. \quad \Pi; \cdot \vdash v_n \Leftarrow \tau_n \end{array}}{\Pi \vdash \Delta \sim \delta} \quad \text{DSIM\_DSIM}$$

Lemma 48. *If* $\Pi; \Delta \vdash \langle \delta, s \rangle \Leftarrow \{z : b | \phi\}$ *then* $\Pi; \bullet \vdash \Delta \sqsubseteq \Delta$

Proof. Using G_WS_EMPTY we have that the empty Γ-context is well-scoped and using EXTEND_D_EMPTY the conclusion follows. □

Lemma 49. *If* $\Pi; \bullet \vdash \tau$ ***ws*** *and* $\Pi; \bullet \vdash \Delta$ ***ws*** *and* $u \notin dom(\Delta)$ *then* $\Pi; \bullet \vdash \Delta \subseteq \Delta, (u, \tau)$

Proof. The conclusons follows using EXTEND_D_CONS with the Δ' of the rule being empty. □

Lemma 50 is a key lemma for showing soundness of the MiniSail type system with respect to the defined operational semantics. A number of reduction steps in the semantics make use of substitution and the substitution lemmas proved above play an important role in the proof.

Lemma 50 (Single Step Preservation)**.** *If* $\Pi; \Delta \vdash \langle \delta, s \rangle \Leftarrow \{z : b|\phi\}$ *and* $\Pi \vdash \langle \delta, s \rangle \rightarrow \langle \delta', s' \rangle$ *then there is a* Δ' *such that* $\Pi; \bullet \vdash \Delta \sqsubseteq \Delta'$ *and* $\Pi; \Delta' \vdash \langle \delta', s' \rangle \Leftarrow \{z : b|\phi\}$*.*

Proof. Induction on the reduction step $\langle \delta, s \rangle \rightarrow \langle \delta', s' \rangle$. In all but one of the cases below, we will let Δ be the Δ' that we need to exhibit and make use of Lemma 48.

- *Case* '$\Pi \vdash \langle \delta, \textbf{if T then } s_1 \textbf{ else } s_2 \rangle \rightarrow \langle \delta, s_1 \rangle$'
  Using inversion of CHECK_REDEX_STMT on the lemma premises, we have:

$$\begin{aligned} \Pi &\vdash \Delta \sim \delta \\ \Pi; \bullet; \Delta &\vdash \textbf{if T then } s_1 \textbf{ else } s_2 \Leftarrow \{z : b|\phi\} \end{aligned}$$

  Using inversion of CHECK_ANF_IF on the second of these we have:

$$\begin{aligned} \Pi; \bullet; \Delta \vdash s_1 &\Leftarrow \{z_1 : b|\mathbf{T} = \mathbf{T} \wedge \mathbf{T} = \mathbf{T} \Longrightarrow \phi[z_1/z]\} \\ \Pi; \bullet; \Delta \vdash s_2 &\Leftarrow \{z_2 : b|\mathbf{T} = \mathbf{F} \wedge \mathbf{T} = \mathbf{T} \Longrightarrow \phi[z_2/z]\} \end{aligned}$$

  We have

$$\Pi; \bullet \vdash \{z_1 : b|\mathbf{T} = \mathbf{T} \wedge \mathbf{T} = \mathbf{T} \Longrightarrow \phi[z_1/z]\} \lesssim \{z : b|\phi\}$$

  and so using Lemma 17 we have

$$\Pi; \bullet; \Delta \vdash s_1 \quad \Leftarrow \quad \{z : b|\phi\}$$

  If we pick Δ' to be Δ then the premises of CHECK_REDEX_STMT hold and the conclusions follows.

- *Case* '$\Pi \vdash \langle \delta, \textbf{if F then } s_1 \textbf{ else } s_2 \rangle \rightarrow \langle \delta, s_2 \rangle$'
  Similar to the previous case.

- *Case* '$\Pi \vdash \langle \delta, \textbf{let } x = v \textbf{ in } s \rangle \rightarrow \langle \delta, s[v/x] \rangle$
  Using inversion of CHECK_REDEX_STMT on the lemma premises, we have:

$$\begin{aligned} \Pi &\vdash \Delta \sim \delta \\ \Pi; \bullet; \Delta &\vdash \textbf{let } x = v \textbf{ in } s \Leftarrow \{z : b|\phi\} \end{aligned}$$

  From inversion of the rule CHECK_ANF_LET, there is a type $\{z_1 : b_1|\phi_1\}$ such that the following hold:

$$\begin{aligned} \Pi; \bullet &\vdash v \Rightarrow \{z_1 : b_1|\phi_1\} \\ \Pi; x : b_1[\phi_1[x/z]]; \Delta &\vdash s \Leftarrow \{z : b|\phi\} \end{aligned}$$

  Since

$$\bullet \vdash \{z_1 : b_1|\phi_1\} \lesssim \{z_1 : b_1|\phi_1\}$$

  we have the premises of the substitution lemma for statements with Γ and Γ' being empty and using this lemma we get

$$\Pi; \bullet; \Delta \vdash s[v/x] \Leftarrow \{z : b|\phi\}[v/x]$$

However, x is fresh in ϕ and so the substitution is the identity and we have:

$$\Pi; \bullet; \Delta \vdash s[v/x] \Leftarrow \{z : b|\phi\}$$

If we pick Δ' to be Δ then the premises of CHECK_REDEX_STMT hold and the conclusion follows.

- *Case* '$\Pi \vdash \langle \delta, \textbf{let } x = n_1 + n_2 \textbf{ in } s\rangle \rightarrow \langle \delta, \textbf{let } x = n \textbf{ in } s\rangle$' where $\overline{n} = \overline{n}_1 + \overline{n}_2$
Using inversion of CHECK_REDEX_STMT on the lemma premises, we have:

$$\begin{array}{rcl} \Pi & \vdash & \Delta \sim \delta \\ \Pi; \bullet; \Delta & \vdash & \textbf{let } x = n_1 + n_2 \textbf{ in } s \Leftarrow \{z : b|\phi\} \end{array}$$

Using inversion with the rule CHECK_ANF_LET, we have that there are b' and ϕ' such that:

$$\begin{array}{rcl} \Pi; \bullet; \Delta & \vdash & n_1 + n_2 \Rightarrow \{z' : b'|\phi'\} \\ \Pi; \Gamma, x : b[\phi'[x/z']]; \Delta & \vdash & s \Leftarrow \{z : b|\phi\} \end{array}$$

Using inversion with the rule INFER_ANF_PLUS on the first of these, we can infer $b' \equiv \textbf{int}$ and $\phi' \equiv (z' = n_1 + n_2)$. Since

$$\Pi; \bullet \vdash \{z' : \textbf{ int } |z' = n\} \lesssim \{z' : \textbf{ int } |z' = n_1 + n_2\}$$

we can use Lemma 22 to get:

$$\Pi; x : b[x = n]; \Delta \vdash s \Leftarrow \{z : b|\phi\}$$

and since

$$\Pi; \bullet \vdash n \Rightarrow \{z' : \textbf{int } |z' = n\}$$

using the rule INFER_V_NUM we can use CHECK_ANF_LET we get the first part of the conclusion

$$\Pi; \bullet; \Delta \vdash \textbf{let } x = n \textbf{ in } s \Leftarrow \{z : b|\phi\}$$

If we pick Δ' to be Δ then the premises of CHECK_REDEX_STMT hold and the conclusion follows.

- *Case* '$\Pi \vdash \langle \delta, \textbf{let } x = n_1 \leq n_2 \textbf{ in } s\rangle \rightarrow \langle \delta, \textbf{let } x = b \textbf{ in } s\rangle$' where $\overline{b} = \overline{n}_1 \leq \overline{n}_2$
Similar to previous case.

- *Case* '$\Pi \vdash \langle \delta, \textbf{let } x = f\ v \textbf{ in } s\rangle \rightarrow \langle \delta, \textbf{let } x : \tau = s'[v/z] \textbf{ in } s\rangle$
Using inversion of CHECK_REDEX_STMT on the premises for this case, we have:

$$\begin{array}{rcl} \Pi & \vdash & \Delta \sim \delta \\ \Pi; \bullet; \Delta & \vdash & \textbf{let } x = fv \textbf{ in } s \Leftarrow \{z : b|\phi\} \end{array}$$

From this and inversion of CHECK_ANF_LET we have:

$$\begin{array}{rcll} \Pi; \bullet; \Delta & \vdash & f\ v \Rightarrow \{z_2 : b_2|\phi_2\} & (1) \\ \Pi; x : b_2[\phi_2[x/z_2]]; \Delta & \vdash & s \Leftarrow \{z : b|\phi\} & (2) \end{array}$$

From (1) and inversion of INFER_ANF_APP we have that:

$$\begin{array}{rcll} \textbf{val } f : (w : b_1[\phi_1]) & \rightarrow & \tau \in \Pi & (3) \\ \textbf{function } f(w) & = & s' \in \Pi & (4) \\ \Pi; \bullet; \Delta & \vdash & v \Leftarrow \{z_1 : b_1|\phi_1[z_1/w]\} & (5) \\ \{z_2 : b_2|\phi_2\} & = & \tau[v/w] & (6) \end{array}$$

From (5) and inversion of CHECK_V_VAL we have that there is a ϕ_3 such that

$$\begin{aligned} \Pi; \bullet; \Delta \ &\vdash \ v \Rightarrow \{z_3 : b_1 | \phi_3\} && (7) \\ \Pi; \bullet \ &\vdash \ \{z_3 : b_1 | \phi_3\} \lesssim \{z_1 : b_1 | \phi_1[z_1/x]\} && (8) \end{aligned}$$

From (4) and that we know that the function f is correctly typed and using inversion of CHECK_DEFS_FUNDEF we have that:

$$\Pi; w : b_1[\phi_1]; \Delta \ \vdash s' \ \Leftarrow \tau \qquad (9)$$

Using (6), (7), (8), (9) and the substitution lemma for statements we have:

$$\Pi; \bullet; \Delta \vdash s'[v/w] \Leftarrow \{z_2 : b_2 | \phi_2\} \qquad (10)$$

Using (2), (10) and the rule CHECK_ANF_LET2 we have:

$$\Pi; \bullet; \Delta \vdash \mathbf{let}\ x : \{z_2 : b_2 | \phi_2\} = s'[v/w]\ \mathbf{in}\ s \Leftarrow \{z : b | \phi\}$$

If we pick Δ' to be Δ then the premises of CHECK_REDEX_STMT hold and the conclusion follows.

- *Case* '$\Pi \vdash \langle \delta, \mathbf{let}\ x : \tau = v\ \mathbf{in}\ s_2 \rangle \rightarrow \langle \delta, s_2[v/x] \rangle$'
This is the similar to the proof for the case $\Pi \vdash \langle \delta, \mathbf{let}\ x = v\ \mathbf{in}\ s \rangle \rightarrow \langle \delta, s[v/x] \rangle$

- *Case* '$p \vdash \langle \delta, \mathbf{let}\ x : \tau = s_1\ \mathbf{in}\ s_2 \rangle \rightarrow \langle \delta', \mathbf{let}\ x : \tau = s_1'\ \mathbf{in}\ s_2 \rangle$' where $\langle \delta, s_1 \rangle \rightarrow \langle \delta', s_1' \rangle$
Using inversion of CHECK_REDEX_STMT on the premises for this case, we have:

$$\begin{aligned} \Pi \ &\vdash \ \Delta \sim \delta \\ \Pi; \bullet; \Delta \ &\vdash \ \mathbf{let}\ x : \tau = s_1\ \mathbf{in}\ s_2 \Leftarrow \{z : b | \phi\} \end{aligned}$$

Using inversion of CHECK_ANF_LET2, we have that there is a b' and ϕ' such that $\tau = \{z' : b' | \phi'\}$ and

$$\begin{aligned} \Pi; \bullet; \Delta \ &\vdash \ s_1 \Leftarrow \{z' : b' | \phi'\} && (1) \\ \Pi; x : b'[\phi'[x/z]]; \Delta \ &\vdash \ s_2 \Leftarrow \{z : b | \phi\} && (2) \end{aligned}$$

Since s_1 reduces to s_1', we have by the induction hypothesis that there is a Δ' such that $\Delta \sqsubseteq \Delta'$ and

$$\Pi; \bullet; \Delta' \vdash s_1' \Leftarrow \{z' : b' | \phi'\}$$

and from (2) by weakening, we have:

$$\Pi; x : b'[\phi'[x/z]]; \Delta' \vdash s_2 \Leftarrow \{z : b | \phi\}$$

Applying CHECK_ANF_LET2 in the forward direction we have:

$$\Pi; \bullet; \Delta' \vdash \mathbf{let}\ x : \tau = s_1'\ \mathbf{in}\ s_2 \Leftarrow \{z : b | \phi\}$$

If we pick Δ' to be the Δ' we got from the reduction of s_1 then the premises of CHECK_REDEX_STMT hold and the conclusion follows.

- *Case* $\Pi \vdash \langle \delta, \mathbf{match}\ C\ v\ \{\ \mathrm{C}_1\ x_1 \Rightarrow s_1\ ,\ .., \mathrm{C}\ x \Rightarrow s\ ,\ \mathrm{C}_n\ x_n \Rightarrow s_n\ \} \rangle \rightarrow \langle \delta, s[v/x] \rangle$
Using inversion of CHECK_REDEX_STMT on premises for this case, we have:

$$\begin{aligned} \Pi \ &\vdash \ \Delta \sim \delta \\ \Pi; \bullet; \Delta \ &\vdash \ \mathbf{match}\ C\ v\ \{\ \mathrm{C}_1\ x_1 \Rightarrow s_1\ ,\ .., \mathrm{C}\ x \Rightarrow s\ ,\ \mathrm{C}_n\ x_n \Rightarrow s_n\ \} \Leftarrow \{z : b | \phi\} \end{aligned}$$

From inversion of CHECK_S_MATCH, we have:

$$\begin{aligned}
\Pi; \bullet \quad &\vdash \quad C\ v \Rightarrow \{z' : \text{tid} | \phi'\} && (1)\\
\textbf{union}\ \text{tid}\ &= \{\text{C}_1 : \{z_1 : b''_1 | \phi''_1\}, \quad .. \quad , \text{C}_n : \{z_n : b''_n | \phi''_n\}\} \in \Pi && (2)\\
\Pi; x_i : b''_i[\phi''_i[x_i/z_i]] \wedge C\ v = C_i\ x_i \wedge \phi'[C\ v/z']; \Delta \quad &\vdash \quad s_i[v/x] \Leftarrow \{z : b | \phi\} \quad i = 1..n && (3)
\end{aligned}$$

From (1) and inversion of INFER_V_DATA_CONS we have that there is a k such that $C\ v = C_k\ v$ and $s_k = s$ and:

$$\Pi; \bullet \vdash v \Leftarrow \{z''_k | b''_k | \phi''_k]\} \tag{4}$$

Using inversion of CHECK_V_VAL we have that there is a ϕ'' such that:

$$\begin{aligned}
\Pi; \bullet \vdash v \Rightarrow \{z'' | b''_k | \phi'']\} && (5)\\
\Pi; \bullet \vdash \{z'' : b''_k | \phi''\} \lesssim \{z'' : b''_k | \phi''_k\} && (6)
\end{aligned}$$

From the kth judgement in (3) we have:

$$\Pi; x_k : b''_k[\phi''_k[v/x]]; \Delta \vdash s_k[v/x_k] \Leftarrow \{z : b | \phi\} \tag{7}$$

Using (7), (5) and (6) with the substitution lemma for statements we have:

$$\Pi; \bullet \vdash s_k[v/x_k] \Leftarrow \{z : b | \phi\}. \tag{8}$$

If we pick Δ' to be Δ then the premises of CHECK_REDEX_STMT hold and the conclusion follows.

- *Case* $\Pi \vdash \langle \delta, \textbf{while}\ (e)\ \ \textbf{do}\ \{s\} \rangle \to \langle \delta, \textbf{let}\ x = e\ \textbf{in}\ (\textbf{if}\ x\ \textbf{then}\ (s; \textbf{while}\ (e)\ \ \textbf{do}\ \{s\})\ \textbf{else}\ ()) \rangle$
Using inversion of CHECK_REDEX_STMT on premises for this case, we have:

$$\begin{aligned}
\Pi \quad &\vdash \quad \Delta \sim \delta && (1)\\
\Pi; \bullet; \Delta \quad &\vdash \quad \textbf{while}\ (e)\ \ \textbf{do}\ \{s\} \Leftarrow \{z : \textbf{unit}\ | z = ()\} && (2)
\end{aligned}$$

Using inversion of CHECK_S_WHILE we have:

$$\begin{aligned}
\Pi; \bullet; \Delta \quad &\vdash \quad e \Leftarrow \{z_1 : \textbf{bool}\ | \top\} && (3)\\
\Pi; \bullet; \Delta \quad &\vdash \quad s \Leftarrow \{z : \textbf{unit}\ | z = ()\} && (4)
\end{aligned}$$

We now show a type derivation for the unrolled while statement.

Using inversion of CHECK_E with (3) we have that there is a ϕ_2 such that:

$$\begin{aligned}
\Pi; \bullet; \Delta \quad &\vdash \quad e \Rightarrow \{z_2 : \textbf{bool}\ | \phi_2\} && (5)\\
\Pi; \bullet; \Delta \quad &\vdash \quad \{z_2 : \textbf{bool}\ | \phi_2\} \lesssim \{z_1 : \textbf{bool}\ | \top\} && (6)
\end{aligned}$$

From the rule INFER_V_VAR we have:

$$\Pi; x : \textbf{bool}\ [\phi_2] \vdash x \Rightarrow \{z_2 : \textbf{bool}\ | \phi_2\} \tag{7}$$

Using the rule CHECK_S_SEQ with (2) and (4) we have:

$$\Pi; \bullet; \Delta \vdash s; \textbf{while}\ (e)\ \ \textbf{do}\ \{s\} \Leftarrow \{z : \textbf{unit}\ | z = ()\} \tag{8}$$

Using weaking of Γ on this, (4) and (5) we have:

$$\begin{aligned}
\Pi; x : \textbf{bool}\ [\phi_2]; \Delta \quad &\vdash \quad e \Rightarrow \{z_2 : \textbf{bool}\ | \phi_2\} && (9)\\
\Pi; x : \textbf{bool}\ [\phi_2]; \Delta \quad &\vdash \quad s;\ \textbf{while}\ (e)\ \ \textbf{do}\ \{s\} \Leftarrow \{z : \textbf{unit}\ | z = ()\} && (10)\\
\Pi; x : \textbf{bool}\ [\phi_2]; \Delta \quad &\vdash \quad s \Leftarrow \{z : \textbf{unit}\ | z = ()\} && (11)
\end{aligned}$$

and using CHECK_S_IF on (7), (10) and (11) we have:

$$\Pi; x : \textbf{bool}\ [\phi_2]; \Delta \vdash \textbf{if}\ x\ \textbf{then}\ s; \textbf{while}\ (e)\ \ \textbf{do}\ \{s\}\ \textbf{else}\ () \Leftarrow \{z : \textbf{unit}\ |z = ()\} \tag{12}$$

Using rule CHECK_S_LET with (9) and (12) we have:

$$\Pi; \bullet; \Delta \vdash \textbf{let}\ x = e\ \textbf{in}\ \textbf{if}\ x\ \textbf{then}\ s; \textbf{while}\ (e)\ \ \textbf{do}\ \{s\}\ \textbf{else}\ () \Leftarrow \{z : \textbf{unit}\ |z = ()\} \tag{13}$$

If we pick Δ' to be Δ then from (1) and (13) the premises of CHECK_REDEX_STMT hold and the conclusion follows.

- *Case* $\Pi \vdash \langle \delta, \textbf{var}\ u\ \ : \tau = v\ \textbf{in}\ s \rangle \to \langle \delta[u \mapsto v], s \rangle$
From the premises and inversion for the program state typing judgement, we have

$$\begin{aligned} \Pi &\ \vdash\ \Delta \sim \delta \\ \Pi; \bullet; \Delta &\ \vdash\ \textbf{var}\ u\ \ : \tau = v\ \textbf{in}\ s \Leftarrow \{z : b|\phi\} \end{aligned}$$

From inversion of CHECK_S_VAR we have that:

$$u \notin \text{dom}(\Delta) \tag{1}$$
$$\Pi; \bullet \vdash v \Leftarrow \tau \tag{2}$$
$$\Pi; \bullet : \Delta, (u, \tau) \vdash s \Leftarrow \{z : b|\phi\} \tag{3}$$

Let $\Delta' = \Delta, (u, \tau)$. From the inversion of the reduction rule, we have that u is an addition to δ and from above it is also an addition to Δ and so

$$\Pi \ \vdash\ \Delta, (u, \tau) \sim \delta[u \mapsto v]$$

with this and (3) the premises of CHECK_REDEX_STMT hold and so we have $\Pi \vdash \langle \delta[u \mapsto v], s \rangle \Leftarrow \tau$. Furthermore, we have $\Pi; \bullet \vdash \Delta \sqsubseteq \Delta, (u, \tau)$ from Lemma 49 and so we have the required conclusion.

- *Case* $\Pi \vdash \langle \delta, u := v \rangle \to \langle \delta[u \mapsto v], () \rangle$
Using inversion of CHECK_REDEX_STMT on premises for this case, we have:

$$\begin{aligned} \Pi &\ \vdash\ \Delta \sim \delta \\ \Pi; \bullet; \Delta &\ \vdash\ u := v \Leftarrow \{z : b|\phi\} \end{aligned}$$

From inversion of CHECK_S_ASSIGN we have that:

$$\begin{aligned} \Pi; \bullet \vdash \Delta\ \textbf{ws} \\ \{z : b|\phi\} = \{z : \textbf{unit}|\top\} \\ (u, \tau) \in \Delta \\ \Pi; \bullet \vdash v \Leftarrow \tau \end{aligned}$$

We have $\Pi; \bullet; \Delta \vdash () \Leftarrow \{z : \textbf{unit}|\top\}$ and so if we pick Δ' to be Δ then all of the premises of CHECK_REDEX_STMT hold and the conclusion follows.

- *Case* $\Pi \vdash \langle \delta, (); s_2 \rangle \to \langle \delta, s_2 \rangle$
Using inversion of CHECK_REDEX_STMT on premises for this case, we have:

$$\begin{aligned} \Pi &\ \vdash\ \Delta \sim \delta \\ \Pi; \bullet; \Delta &\ \vdash\ (); s_2 \Leftarrow \{z : b|\phi\} \end{aligned}$$

With this and inversion of CHECK_S_SEQ we have:

$$\begin{array}{rcl}\Pi; \bullet; \Delta & \vdash & () \Leftarrow \{z : \mathbf{unit} | \top\} \\ \Pi; \bullet; \Delta & \vdash & s_2 \Leftarrow \{z : b | \phi\}\end{array}$$

If we pick Δ' to be Δ then the premises of CHECK_REDEX_STMT hold and and the conclusion follows.

- *Case* $\Pi \vdash \langle \delta, s_1; s_2 \rangle \rightarrow \langle \delta', s_1'; s_2 \rangle$
  Using inversion of CHECK_REDEX_STMT on premises for this case, we have:

$$\begin{array}{rcl}\Pi & \vdash & \Delta \sim \delta \\ \Pi; \bullet; \Delta & \vdash & s_1; s_2 \Leftarrow \{z : b | \phi\}\end{array}$$

  With the second of these and inversion of CHECK_S_SEQ we have:

$$\begin{array}{rcl}\Pi; \bullet; \Delta & \vdash & s_1 \Leftarrow \{z : \mathbf{unit} | \top\} \\ \Pi; \bullet; \Delta & \vdash & s_2 \Leftarrow \{z : b | \phi\}\end{array}$$

  s_1' is the result of a one step reduction of s_1 and so by the induction hypothesis we have that there is a Δ'' such that

$$\Pi; \Delta' \vdash \langle \delta', s_1' \rangle \Leftarrow \{z : \mathbf{unit} | \top\}$$

  and $\Delta \sqsubseteq \Delta'$. Using inversion of CHECK_REDEX_STMT, we then have:

$$\begin{array}{rcl}\Pi & \vdash & \Delta' \sim \delta' \\ \Pi; \bullet; \Delta' & \vdash & s_2' \Leftarrow \{z : \mathbf{unit} | \top\}\end{array}$$

  Using CHECK_S_SEQ the type of $s_1'; s_2$ is $\{z : b | \phi\}$. Hence we have the required premises for CHECK_REDEX_STMT and the conclusion follows.

□

Lemma 51 (Multistep Preservation). *If* $\Pi; \Delta \vdash \langle \delta, s \rangle \Leftarrow \tau$ *and* $\langle \delta, s \rangle \xrightarrow{*} \{\delta', s'\}$ *then there is a* Δ' *such* $\Pi; \Delta' \vdash \langle \delta', s' \rangle \Leftarrow \tau$ *and* $\Delta \sqsubseteq \Delta'$

Proof. We prove this by induction on the length, l, of the reduction $\langle \delta, s \rangle \xrightarrow{*} \langle \delta', s' \rangle$.
If $l = 0$ then $s' = s$ and $\delta' = \delta$ and the conclusion holds.
Assume that the lemma holds for $l = n$ and that we have an n step reduction $\langle \delta, s \rangle \xrightarrow{*} \langle \delta'', s'' \rangle$ and a single step reduction $\langle \delta'', s'' \rangle \xrightarrow{*} \{\delta', s'\}$. By the induction hypothesis, we have that there is a Δ'' such that $\Pi; \Delta'' \vdash \langle \delta'', s'' \rangle \Leftarrow \tau$ and $\Delta \sqsubseteq \Delta''$ With the single step reduction we apply the preservation lemma and so there is a Δ' such that $\Pi; \Delta' \vdash \langle \delta', s' \rangle \Leftarrow \tau$ and $\Delta'' \sqsubseteq \Delta'$. From the transitivity of $\sqsubseteq$ we have that $\Delta \sqsubseteq \Delta'$ and the required conclusion follows. □

The following lemma assures us that we have set up the reduction rules correctly and that any well typed statement that isn't a value will have a reduction step.

Lemma 52 (Progress). *If* $\Pi; \Delta \vdash \langle \delta, s \rangle \Leftarrow \tau$ *then either* s *is a value or there is an* s' *and* δ' *such that* $\langle \delta, s \rangle \rightarrow \langle \delta', s' \rangle$.

Proof. Proof is by induction on s.

- *Case* s is a value.
  Lemma conclusion holds.

- *Case* '**if** v **then** s_1 **else** s_2'
  Instantiating the premise and using inversion of CHECK_REDEX_STMT, we have:

  $$\Pi; \bullet; \Delta \vdash \mathbf{if}\ v\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2 \Leftarrow \tau$$

  Using inversion with the rule CHECK_S_IF we have that there is a ϕ' such that:

  $$\Pi; \bullet \vdash v \Rightarrow \{z' : \mathbf{bool} | \phi'\}$$

  Since the Γ-context is empty and the base type is **bool**, v can only be a boolean literal - **T** or **F**. If $v \equiv \mathbf{T}$ then the reduction rule REDUCE_TRUE is applicable and we have a reduction to $\langle \delta, s_1 \rangle$. Otherwise we have $v \equiv \mathbf{F}$ and the reduction rule REDUCE_FALSE is applicable and we have a reduction to $\langle \delta, s_2 \rangle$.

- *Case* '**let** $x = e$ **in** s'
  Using inversion of CHECK_REDEX_STMT we have

  $$\begin{aligned} \Pi &\vdash \Delta \sim \delta && (1) \\ \Pi; \bullet; \Delta &\vdash s \Leftarrow \tau && (2) \end{aligned}$$

  and inversion of CHECK_S_LET on the second of these, we get a judgement:

  $$\Pi; \bullet; \Delta \vdash e \Rightarrow \{z' : b' | \phi'\} \qquad (3)$$

  As the typing context is empty e will not contain immutable variables. We consider each possible form for e.

  - *Case* $e \equiv v$
    Apply the reduction rule REDUCE_LET_VALUE to obtain a reduction to $\langle \delta, s[v/x] \rangle$
  - *Case* $e \equiv v_1 + v_2$
    Using inversion of INFER_E_PLUS we have that v_1 and v_2 have base type **int**. Since the context is empty, these have to be integer literals and so there is a n such that $\overline{n} = \overline{v_1} + \overline{v_2}$ and we can apply the reduction rule REDUCE_LET_PLUS to obtain a reduction to $\langle \delta, \mathbf{let}\ x = n\ \mathbf{in}\ s \rangle$.
  - *Case* $e \equiv v_1 \leq v_2$
    Similar to the previous case.
  - *Case* $e \equiv f\ v$
    Using inversion with the rule INFER_E_APP we have

    $$\begin{aligned} \mathbf{val}\ f : (x : b_1[\phi_1]) \to \tau \in \Pi && (4) \\ \mathbf{function}\ f(x) = s' \in \Pi && (5) \end{aligned}$$

    $$\begin{aligned} \Pi; \bullet &\vdash v \Rightarrow \{z : b | \phi_2\} && (6) \\ \Pi; \bullet &\vdash \{z : b | \phi_2\}[v/z] \lesssim \{z : b | \phi_1\}[v/x] && (7) \end{aligned}$$

    (4) and (5) are the preconditions for the reduction rule REDUCE_LET_APP and so we can apply this to obtain a reduction to $\langle \delta, \mathbf{let}\ x : \tau[v/x] = s'[v/x]\ \mathbf{in}\ s \rangle$
  - *Case* $e \equiv u$
    From (3) above and inversion of INFER_E_MVAR we have that $(u, \tau) \in \Delta$ and from this and (1) above we have that u is in the domain of δ. Hence we can apply REDUCE_LET_MVAR to obtain a reduction to $\langle \delta, s[v/x] \rangle$ where $v = \delta(u)$.

  - *Case* $e \equiv$ **fst** v
    Using inversion of INFER_E_FST we have that v is a pair of values (v_1, v_2). We can apply the reduction rule REDUCE_LET_FST to obtain a reduction to $\langle \delta, s[v_1/x] \rangle$
  - *Case* $e \equiv$ **snd** v
    Using inversion of INFER_E_FST we have that v is a pair of values (v_1, v_2). Apply the reduction rule REDUCE_LET_SND to obtain a reduction to $\langle \delta, s[v_2/x] \rangle$

- *Case* '**let** $x : \tau' = s_1$ **in** s_2'
  If s_1 is a value, then the we can apply REDUCE_LET2_VAL . If it isn't a value then using inversion of CHECK_REDEX_STMT we have

$$\begin{array}{rcl} \Pi & \vdash & \Delta \sim \delta \qquad (1) \\ \Pi; \cdot; \Delta & \vdash & \textbf{let } x : \tau' = s_1 \textbf{ in } s_2 \Leftarrow \tau \qquad (2) \end{array}$$

  and from the second using inversion of CHECK_S_LET2 we have

$$\Pi; \bullet; \Delta \vdash s_1 \Leftarrow \tau' \qquad (3)$$

  and so using (1) and (3) we can use CHECK_REDEX_STMT to get:

$$\Pi; \Delta \vdash \langle \delta, s_1 \rangle \Leftarrow \tau' \qquad (4)$$

  and so can apply the induction hypothesis to obtain δ' and s_1' such that $\langle \delta, s_1 \rangle \rightarrow \langle \delta', s_1' \rangle$ and we can then apply REDUCE_LET2_STMT to obtain a reduction to $\langle \delta', \textbf{let } x : \tau' = s_1' \textbf{ in } s_2$

- *Case* '**match** v' { $\mathrm{C}_1\ x_1 \Rightarrow s_1$, .. , $\mathrm{C}_n\ x_n \Rightarrow s_n$}'
  Using inversion of CHECK_REDEX_STMT we have that

$$\Pi; \bullet; \Delta \vdash \textbf{match } v' \ \{\ \mathrm{C}_1\ x_1 \Rightarrow s_1\ ,\ ..\ ,\ \mathrm{C}_n\ x_n \Rightarrow s_n\} \Leftarrow \tau$$

  and with this and inversion of CHECK_S_MATCH we have

$$\begin{array}{rcl} & & \textbf{union } tid = \{\mathrm{C}_1 : \{z_1 : b_1''|\phi_1''\} \ \ ..\ \ \mathrm{C}_n : \{z_n : b_n''|\phi_n''\}\} \in \Pi \\ \Pi; \Gamma & \vdash & v' \Rightarrow \{z : tid|\phi\} \\ \Pi; \Gamma, x_i : b_i''[\phi_i''[x_i/z_i] \wedge v' = C\ x_i \wedge \phi[v/z]]; \Delta & \vdash & s_i \Leftarrow \{z_1 : b_1|\phi_1\} \quad i = 1..n \end{array}$$

  With the second of these and inversion of INFER_V_CONS we have that there is a j and v'' such that $v' = C_j\ v''$ and further that C_j matches one of the branches of the case statement. Hence we can apply the reduction rule REDUCE_MATCH and obtain a reduction to $\langle \delta, s_j[v''/x_j] \rangle$.

- *Case* '**while** (e) **do** $\{s\}$'
  The rule REDUCE_WHILE only has a freshness condition and after picking a fresh variable x, we can reduce to $\langle \delta, \textbf{let } x = e \textbf{ in if } x \textbf{ then } s; \textbf{while } (e) \textbf{ do } \{x\} \textbf{ else } () \rangle$.

- *Case* '$u := v$'
  Using inversion of CHECK_REDEX_STMT we have that

$$\begin{array}{rcl} \Pi & \vdash & \Delta \sim \delta \qquad (1) \\ \Pi; \cdot; \Delta & \vdash & u := v \Leftarrow \tau \qquad (2) \end{array}$$

  and from inversion of CHECK_S_ASSIGN on the second of these we have that that there is a τ' such that $(u, \tau') \in \Delta$. From (1) and inversion of CHECK_DELTA we have that u is in the domain of δ and using rule REDUCE_MVAR_ASSIGN we can obtain a reduction to $(\delta[u \mapsto v], ())$.

- *Case* '**var** $u\ : \tau = v$ **in** s'
  Renaming u to a new variable name if u is in the domain of δ, we reduce to $(\delta[u \mapsto v], s)$.

- *Case* '$s_1; s_2$'
  Using inversion of CHECK_REDEX_STMT we have that

$$\Pi \;\vdash\; \Delta \sim \delta \tag{1}$$
$$\Pi; \cdot; \Delta \;\vdash\; s_1; s_2 \Leftarrow \tau \tag{2}$$

  and using inversion of CHECK_S_SEQ on the second of these we get:

$$\Pi; \cdot; \Delta \;\vdash\; s_1 \Leftarrow \{z : \mathbf{unit}\ |\top\} \tag{3}$$

  If s_1 is a value then it must be () and so we can apply REDUCE_SEQ2 to reduce to $\langle\delta, s_2\rangle$. If s_1 is not a value then we can use (1) and (3) with the rule CHECK_REDEX_STMT to obtain:

$$\Pi; \Delta \vdash \langle\delta, s_1\rangle \Leftarrow \{z : \mathbf{unit}\ |\top\}$$

  and then apply the induction hypothesis to obtain a reduction to $\langle\delta', s_1'\rangle$ and finally use REDUCE_SEQ1 to obtain a reduction of the original statement to $\langle\delta', s_1'; s_2\rangle$.

□

Using the above two lemmas we can prove:

Lemma 53 (Safety)**.** *If* $\Pi; \Delta \vdash \langle\delta, s\rangle \Leftarrow \tau$ *and* $\langle\delta, s\rangle \xrightarrow{*} \langle\delta', s'\rangle$ *then either* s' *is a value or there is an* s'' *and* δ'' *such that* $\langle\delta', s'\rangle \rightarrow \langle\delta'', s''\rangle$.

Proof. From the multi-step Preservation lemma, we have that there is a Δ' such that $\Pi; \Delta' \vdash \langle\delta', s'\rangle \Leftarrow \tau$ and from the Progress lemma we have that s' is a value or that we can make a reduction step from s'. □

11 Conclusion

The final lemma, the safety property for MiniSail, tells us that a well typed statement that has a reduction has reduced to a value or can make further reduction steps. If we consider a MiniSail program to be a type checked context, Π, and a final statement, s, then we can claim that a well-typed MiniSail program does not get stuck. We can not say that a well-typed program will terminate; proving termination is not possible due to the inclusion of while loops and recursive functions in the language. Non-termination is true of Sail as well.

12 Appendix 1 - Complete Syntax, Type System and Operational Semantics

n, m, i, j	Index variables for meta-lists
num	Numeric literals
x, y, z, w, f, $\hat{a}$	Identifier
u	Mutable Variables
tid	Type ID
C	Data Constructor

b	::=		Base Type
	|	**int**	
	|	**bool**	
	|	tid	Type ID
	|	**unit**	
	|	$b_1 * b_2$	

τ	::=		Refined Type
	|	$\{x : b \| \phi\}$	bind x in ϕ
	|	$x : b[\phi]$	bind x in ϕ

	$\vert$	$\tau[v/x]$	M	
	$\vert$	(τ)	S	
A	$::=$			Dependent Function Type
	$\vert$	τ		
	$\vert$	$x : b[\phi] \to \tau_2$		
ϕ	$::=$			Refinement Constraints - Quantifier free lo
	$\vert$	$\top$		
	$\vert$	$\phi_1 \wedge \phi_2$		
	$\vert$	$\neg\phi$		
	$\vert$	$e_1 = e_2$		
	$\vert$	$e_1 \Leftarrow e_2$		
	$\vert$	(ϕ)	S	
	$\vert$	$\phi[e/x]$	M	
	$\vert$	$\phi_1 \implies \phi_2$		
v	$::=$			Values
	$\vert$	x		Immutable variable
	$\vert$	n		Numeric literal
	$\vert$	$\mathbf{T}$		True boolean literal
	$\vert$	$\mathbf{F}$		False boolean literal
	$\vert$	$v_1[v_2/x]$	M	Substitution
	$\vert$	(v)	S	
	$\vert$	(v_1, v_2)		Value pair
	$\vert$	$C\ v$		Data constructor
	$\vert$	$()$		Unit value
e	$::=$			Expressions
	$\vert$	v		Value
	$\vert$	u		Mutable Variable
	$\vert$	$f\ v$		Function Application
	$\vert$	$v_1 + v_2$		Addition
	$\vert$	$v_1 \leq v_2$		Less than or equal
	$\vert$	$\mathbf{fst}\ v$		Project first part of pair
	$\vert$	$\mathbf{snd}\ v$		Project second part of pair
	$\vert$	$e[v/x]$	M	Substitution
s	$::=$			Statement
	$\vert$	v		
	$\vert$	$\mathbf{let}\ x = e\ \mathbf{in}\ s$	bind x in s	Let binding
	$\vert$	$\mathbf{let}\ x : \tau = s_1\ \mathbf{in}\ s_2$	bind x in s_2	Let binding with type annotation
	$\vert$	$\mathbf{if}\ v\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2$		If-then-else
	$\vert$	$s[v/x]$	M	Substitution
	$\vert$	$\mathbf{match}\ v\ \mathbf{of}\ C_1\ x_1 \Rightarrow s_1, \ldots, C_n\ x_n \Rightarrow s_n$		Match statement
	$\vert$	$\mathbf{var}\ u : \tau := v\ \mathbf{in}\ s$	bind u in s	Declaration and scoping of mutable varia
	$\vert$	$u := v$		Assignment to mutable variable

	|	$\mathbf{while}\,(e)\,\mathbf{do}\,\{s\}$		While loop
	|	$s_1 ; s_2$		Statement sequence
	|	(s)	S	
def	::=			Definitions
	|	$\mathbf{val}\, f : (x : b[\phi]) \to \tau$	bind x in τ	
			bind x in ϕ	
	|	$\mathbf{function}\, f(x) = s$	bind x in s	
	|	$\mathbf{union}\, tid = \{ C_1 : \tau_1, \ldots, C_n : \tau_n \}$		
p	::=			Program
	|	$def_1 ; .. ; def_n ; s$		
Γ	::=			Variable type context
	|	$\cdot$		Empty context
	|	$\Gamma, x : b[\phi]$		
	|	(Γ)	S	
	|	Γ_1, Γ_2		
	|	$\Gamma[v/x]$	M	
Π	::=			Function and definitions context
	|	ϵ		
	|	Π, def		
Δ	::=			Mutable variables context
	|	ϵ		
	|	Δ_1, Δ_2		
	|	(Δ)	S	
	|	$\Delta, u : \tau$		
π	::=			Reduction Function Body Context
	|	ϵ		
	|	$\pi, f : s$		
δ	::=			Reduction Local Store
	|	$\delta[u \mapsto v]$		
terminals	::=			
	|	$\sqsubseteq$		
	|	$\to$		
	|	$\vdash$		
	|	$\vdash^{\phi}$		
	|	$\vdash_a$		
	|	$\vdash$		
	|	$\dashv$		
	|	$\models$		
	|	$\Rightarrow$		

	$\|$	$\Leftarrow$
	$\|$	$\vee$
	$\|$	$\wedge$
	$\|$	$\lesssim$
	$\|$	$\uplus$
	$\|$	$\forall$
	$\|$	$\exists$
	$\|$	$\Longrightarrow$
	$\|$	$\longrightarrow$
	$\|$	$\leadsto$
	$\|$	$\in$
	$\|$	$\notin$
	$\|$	$\mapsto$
formula	$::=$	
	$\|$	*judgement*
	$\|$	$formula_1 \quad .. \quad formula_n$
	$\|$	$\Pi; \Gamma \models \phi$
	$\|$	$x : b[\phi] \in \Gamma$
	$\|$	$u : \tau \in \Delta$
	$\|$	$\textbf{union}\, tid = \{ C_1 : \tau_1, .., C_n : \tau_n \} \ \in\ \Pi$
	$\|$	$x \in \mathrm{dom}(\Gamma)$
	$\|$	$\textbf{val}\, f : (x : b[\phi]) \to \tau \ \notin\ \Pi$
	$\|$	$\textbf{val}\, f : (x : b[\phi]) \to \tau \ \in\ \Pi$
	$\|$	$\textbf{function}\, f(x) = s \ \notin\ \Pi$
	$\|$	$\textbf{function}\, f(x) = s \ \in\ \Pi$
	$\|$	$f \in \mathrm{dom}(\Pi)$
	$\|$	$u \in \mathrm{dom}(\Delta)$
	$\|$	$tid \notin \Pi$
	$\|$	$C \notin \Pi$
	$\|$	$f \notin \Pi$
	$\|$	$u \notin \mathrm{dom}(\delta)$
	$\|$	$u \notin \mathrm{dom}(\Delta)$
	$\|$	$x \notin \mathrm{dom}(\Gamma)$
	$\|$	$v_1 + v_2 = v$
	$\|$	$v_1 \leq v_2 = v$
	$\|$	$f\, x = e$
	$\|$	$x_1 = x_2$
	$\|$	$x_1 \neq x_2$
	$\|$	$x \# e$
	$\|$	$x\, \textbf{fresh}$
	$\|$	$v = \delta(u)$
	$\|$	$\delta' = \delta[u \mapsto v]$
	$\|$	$\delta = u_1 \to v_1, .., u_n \to v_n$
	$\|$	$\Delta = u_1 : \tau_1, .., u_n : \tau_n$

wellformedness	::=			
		|	$\Pi \vdash b\,\mathbf{ws}$	Base b is wellformed in context Π
		|	$\Pi \vdash \Gamma\,\mathbf{ws}$	Context Γ is wellformed in context Π
		|	$\Pi \vdash \mathit{def}\,\mathbf{ws}$	Definition def is wellformed in context Π
		|	$\Pi\,\mathbf{ws}$	Context Π is wellformed
		|	$\Pi ; \Gamma \vdash \Delta\,\mathbf{ws}$	Context Δ is wellformed in contexts Π and Γ
		|	$\Pi ; \Gamma \vdash v\ \mathrm{ws}$	Value v is wellformed in contexts Π and Γ
		|	$\Pi ; \Gamma \vDash^{\phi} e\,\mathbf{ws}$	Constraint expression e is wellformed in context Π an
		|	$\Pi ; \Gamma ; \Delta \vdash e\,\mathbf{ws}$	Expression e is wellformed in context Π and Γ
		|	$\Pi ; \Gamma ; \Delta \vdash s\,\mathbf{ws}$	Expression s is wellformed in contexts Π, Δ and Γ
		|	$\Pi ; \Gamma \vdash \phi\,\mathbf{ws}$	Constraint ϕ is wellformed in contexts Π and Γ
		|	$\Pi ; \Gamma \vdash \tau\,\mathbf{ws}$	Type τ is wellformed in context Π, Γ and $\mathbf{D}$
extension	::=			
		|	$\Pi \vdash \Gamma \sqsubseteq \Gamma'$	Γ' is an extension of Γ
		|	$\Pi ; \Gamma \vdash \Delta \sqsubseteq \Delta'$	Δ' is an extension of Δ
subtype	::=			
		|	$\Pi ; \Gamma \vdash \tau_1 \lesssim \tau_2$	
typing	::=			
		|	$\Pi ; \Gamma \vdash v \Rightarrow \tau$	Infer that type of v is τ
		|	$\Pi ; \Gamma \vdash v \Leftarrow \tau$	Check that type of v is τ
		|	$\Pi ; \Gamma ; \Delta \vdash e \Rightarrow \tau$	Infer that type of e is τ
		|	$\Pi ; \Gamma ; \Delta \vdash e \Leftarrow \tau$	Check that type of e is τ
		|	$\Pi ; \Gamma ; \Delta \vdash s \Leftarrow \tau$	Check that type of s is τ
		|	$\Pi ; \Gamma \vdash \mathit{def}_1\ ..\ \mathit{def}_n \rightsquigarrow \Pi'$	
		|	$\Pi ; \Gamma \vdash p$	
		|	$\Pi \vdash \Delta \sim \delta$	
		|	$\Pi ; \Delta \vdash (\delta, s) \Leftarrow \tau$	Program state typing judgement
reduction	::=			
		|	$\Pi \vdash \langle \delta, s \rangle \rightarrow \langle \delta', s' \rangle$	One step reduction
		|	$\Pi \vdash \langle \delta_1, s_1 \rangle \xrightarrow{*} \langle \delta_2, s_2 \rangle$	Multi-step reduction
judgement	::=			
		|	*wellformedness*	
		|	*extension*	
		|	*subtype*	
		|	*typing*	
		|	*reduction*	
user_syntax	::=			
		|	n	
		|	num	
		|	x	
		|	u	

| tid
| C
| b
| τ
| A
| ϕ
| v
| e
| s
| def
| p
| Γ
| Π
| Δ
| π
| δ
| $\mathit{terminals}$
| $\mathit{formula}$

$\boxed{\Pi \vdash b\,\mathbf{ws}}$ Base b is wellformed in context Π

$$\frac{\Pi\,\mathbf{ws}}{\Pi \vdash \mathbf{bool}\,\mathbf{ws}} \quad \text{B\_WS\_BOOL}$$

$$\frac{\Pi\,\mathbf{ws}}{\Pi \vdash \mathbf{int}\,\mathbf{ws}} \quad \text{B\_WS\_INT}$$

$$\frac{\Pi\,\mathbf{ws}}{\Pi \vdash \mathbf{unit}\,\mathbf{ws}} \quad \text{B\_WS\_UNIT}$$

$$\frac{\begin{array}{l}\Pi \vdash b_1\,\mathbf{ws}\\ \Pi \vdash b_2\,\mathbf{ws}\end{array}}{\Pi \vdash b_1 * b_2\,\mathbf{ws}} \quad \text{B\_WS\_PAIR}$$

$$\frac{\begin{array}{l}\Pi\,\mathbf{ws}\\ \mathbf{union}\,\mathit{tid} = \{C_1 : \tau_1, .., C_n : \tau_n\} \in \Pi\end{array}}{\Pi \vdash \mathit{tid}\,\mathbf{ws}} \quad \text{B\_WS\_TID}$$

$\boxed{\Pi \vdash \Gamma\,\mathbf{ws}}$ Context Γ is wellformed in context Π

$$\frac{\Pi\,\mathbf{ws}}{\Pi \vdash \cdot\,\mathbf{ws}} \quad \text{G\_WS\_EMPTY}$$

$$\frac{\begin{array}{l}\Pi \vdash \Gamma\,\mathbf{ws}\\ \Pi \vdash b\,\mathbf{ws}\\ \Pi; \Gamma, x : b[\top] \vdash \phi\,\mathbf{ws}\\ x \notin \mathrm{dom}(\Gamma)\end{array}}{\Pi \vdash \Gamma, x : b[\phi]\,\mathbf{ws}} \quad \text{G\_WS\_CONS}$$

$\boxed{\Pi \vdash \mathit{def}\,\mathbf{ws}}$ Definition def is wellformed in context Π

$$\frac{\begin{array}{l}\overline{\Pi; \cdot \vdash \tau_i\,\mathbf{ws}}^{\,i}\\ \mathit{tid} \notin \Pi\\ \overline{C_i \notin \Pi}^{\,i}\end{array}}{\Pi \vdash \mathbf{union}\,\mathit{tid} = \{\,\overline{C_i : \tau_i}^{\,i}\,\}\,\mathbf{ws}} \quad \text{DEF\_WS\_UNIONDEF}$$

$$\frac{\begin{array}{l}\Pi; \cdot, x : b[\top] \vdash \phi\,\mathbf{ws}\\ \Pi; \cdot, x : b[\phi] \vdash \tau\,\mathbf{ws}\\ \mathbf{val}\, f : (x : b[\phi]) \to \tau \;\notin\; \Pi\end{array}}{\Pi \vdash \mathbf{val}\, f : (x : b[\phi]) \to \tau\,\mathbf{ws}} \quad \text{DEF\_WS\_VALSPEC}$$

$$\frac{\begin{array}{l}\mathbf{val}\, f : (x : b[\phi]) \to \tau \;\in\; \Pi\\ \Pi; \cdot, x : b[\phi]; \Delta \vdash s\,\mathbf{ws}\\ \mathbf{function}\, f(x) = s \;\notin\; \Pi\end{array}}{\Pi \vdash \mathbf{function}\, f(x) = s\,\mathbf{ws}} \quad \text{DEF\_WS\_FUNDEF}$$

$\boxed{\Pi\,\mathbf{ws}}$ Context Π is wellformed

$$\frac{}{\epsilon\,\mathbf{ws}} \quad \text{P\_WS\_EMPTY}$$

$$\frac{\Pi \vdash \mathit{def}\,\mathbf{ws}}{\Pi, \mathit{def}\,\mathbf{ws}} \quad \text{P\_WS\_CONS}$$

$\boxed{\Pi; \Gamma \vdash \Delta\,\mathbf{ws}}$ Context Δ is wellformed in contexts Π and Γ

$$\frac{\Pi \vdash \Gamma\,\mathbf{ws}}{\Pi; \Gamma \vdash \epsilon\,\mathbf{ws}} \quad \text{D\_WS\_EMPTY}$$

$$\frac{\begin{array}{c}\Pi; \Gamma \vdash \Delta\,\mathbf{ws}\\ \Pi; \Gamma \vdash \tau\,\mathbf{ws}\end{array}}{\Pi; \Gamma \vdash \Delta, u : \tau\,\mathbf{ws}} \quad \text{D\_WS\_CONS}$$

$\boxed{\Pi; \Gamma \vdash v \;\text{ws}}$ Value v is wellformed in contexts Π and Γ

$$\frac{\begin{array}{c}\Pi \vdash \Gamma\,\mathbf{ws}\\ x \in \mathrm{dom}(\Gamma)\end{array}}{\Pi; \Gamma \vdash x \;\text{ws}} \quad \text{V\_WS\_VAR}$$

$$\frac{\Pi \vdash \Gamma\,\mathbf{ws}}{\Pi; \Gamma \vdash n \;\text{ws}} \quad \text{V\_WS\_NUM}$$

$$\frac{\Pi \vdash \Gamma\,\mathbf{ws}}{\Pi; \Gamma \vdash \mathbf{T} \;\text{ws}} \quad \text{V\_WS\_TRUE}$$

$$\frac{\Pi \vdash \Gamma\,\mathbf{ws}}{\Pi; \Gamma \vdash \mathbf{F} \;\text{ws}} \quad \text{V\_WS\_FALSE}$$

$$\frac{\Pi \vdash \Gamma\,\mathbf{ws}}{\Pi; \Gamma \vdash () \;\text{ws}} \quad \text{V\_WS\_UNIT}$$

$$\frac{\begin{array}{l}\Pi; \Gamma \vdash v \;\text{ws}\\ \mathbf{union}\, \mathit{tid} = \{\, \overline{C_i : \tau_i}^{\,i} \,\} \;\in\; \Pi\end{array}}{\Pi; \Gamma \vdash C_j\; v \;\text{ws}} \quad \text{V\_WS\_CONS}$$

$$\frac{\begin{array}{c}\Pi; \Gamma \vdash v_1 \;\text{ws}\\ \Pi; \Gamma \vdash v_2 \;\text{ws}\end{array}}{\Pi; \Gamma \vdash (v_1, v_2) \;\text{ws}} \quad \text{V\_WS\_PAIR}$$

$\boxed{\Pi; \Gamma \models^{\phi} e\,\mathbf{ws}}$ Constraint expression e is wellformed in context Π and Γ

$$\frac{\Pi; \Gamma \vdash v \;\text{ws}}{\Pi; \Gamma \models^{\phi} v\,\mathbf{ws}} \quad \text{EC\_WS\_VAR}$$

$$\frac{\begin{array}{c}\Pi;\Gamma \vdash v_1 \ \text{ws} \\ \Pi;\Gamma \vdash v_2 \ \text{ws}\end{array}}{\Pi;\Gamma \vdash^{\phi} v_1 + v_2 \,\mathbf{ws}} \quad \text{EC\_WS\_PLUS}$$

$$\frac{\begin{array}{c}\Pi;\Gamma \vdash v_1 \ \text{ws} \\ \Pi;\Gamma \vdash v_2 \ \text{ws}\end{array}}{\Pi;\Gamma \vdash^{\phi} v_1 \leq v_2 \,\mathbf{ws}} \quad \text{EC\_WS\_LEQ}$$

$$\frac{\Pi;\Gamma \vdash v \ \text{ws}}{\Pi;\Gamma \vdash^{\phi} \mathbf{fst}\, v \,\mathbf{ws}} \quad \text{EC\_WS\_FST}$$

$$\frac{\Pi;\Gamma \vdash v \ \text{ws}}{\Pi;\Gamma \vdash^{\phi} \mathbf{snd}\, v \,\mathbf{ws}} \quad \text{EC\_WS\_SND}$$

$\boxed{\Pi;\Gamma;\Delta \vdash e \,\mathbf{ws}}$ Expression e is wellformed in context Π and Γ

$$\frac{\begin{array}{l}\Pi;\Gamma \vdash \Delta \,\mathbf{ws} \\ \Pi;\Gamma \vdash v \ \text{ws} \\ \mathbf{val}\, f : (x : b[\phi]) \to \tau \ \in \ \Pi\end{array}}{\Pi;\Gamma;\Delta \vdash f\ v \,\mathbf{ws}} \quad \text{E\_WS\_APP}$$

$$\frac{\begin{array}{c}\Pi;\Gamma \vdash \Delta \,\mathbf{ws} \\ \Pi;\Gamma \vdash v_1 \ \text{ws} \\ \Pi;\Gamma \vdash v_2 \ \text{ws}\end{array}}{\Pi;\Gamma;\Delta \vdash v_1 + v_2 \,\mathbf{ws}} \quad \text{E\_WS\_PLUS}$$

$$\frac{\begin{array}{c}\Pi;\Gamma \vdash \Delta \,\mathbf{ws} \\ \Pi;\Gamma \vdash v_1 \ \text{ws} \\ \Pi;\Gamma \vdash v_2 \ \text{ws}\end{array}}{\Pi;\Gamma;\Delta \vdash v_1 \leq v_2 \,\mathbf{ws}} \quad \text{E\_WS\_LEQ}$$

$$\frac{\begin{array}{c}\Pi;\Gamma \vdash \Delta \,\mathbf{ws} \\ \Pi;\Gamma \vdash v \ \text{ws}\end{array}}{\Pi;\Gamma;\Delta \vdash \mathbf{fst}\, v \,\mathbf{ws}} \quad \text{E\_WS\_FST}$$

$$\frac{\begin{array}{c}\Pi;\Gamma \vdash \Delta \,\mathbf{ws} \\ \Pi;\Gamma \vdash v \ \text{ws}\end{array}}{\Pi;\Gamma;\Delta \vdash \mathbf{snd}\, v \,\mathbf{ws}} \quad \text{E\_WS\_SND}$$

$$\frac{\begin{array}{c}\Pi;\Gamma \vdash \Delta \,\mathbf{ws} \\ u \in \mathrm{dom}(\Delta)\end{array}}{\Pi;\Gamma;\Delta \vdash u \,\mathbf{ws}} \quad \text{E\_WS\_MVAR}$$

$\boxed{\Pi;\Gamma;\Delta \vdash s \,\mathbf{ws}}$ Expression s is wellformed in contexts Π, Δ and Γ

$$\frac{\begin{array}{c}\Pi;\Gamma \vdash \Delta \,\mathbf{ws} \\ \Pi;\Gamma \vdash v \ \text{ws}\end{array}}{\Pi;\Gamma;\Delta \vdash v \,\mathbf{ws}} \quad \text{S\_WS\_VAL}$$

$$\frac{\begin{array}{l}\Pi;\Gamma \vdash v \ \text{ws} \\ \Pi;\Gamma;\Delta, u : \tau \vdash s \,\mathbf{ws} \\ \Pi;\Gamma \vdash \tau \,\mathbf{ws}\end{array}}{\Pi;\Gamma;\Delta \vdash \mathbf{var}\, u : \tau := v \,\mathbf{in}\, s \,\mathbf{ws}} \quad \text{S\_WS\_VAR}$$

$$\frac{\begin{array}{l}\Pi;\Gamma \vdash v \ \text{ws} \\ \Pi;\Gamma \vdash \tau \,\mathbf{ws} \\ \Pi;\Gamma;\Delta \vdash u \,\mathbf{ws}\end{array}}{\Pi;\Gamma;\Delta \vdash u := v \,\mathbf{ws}} \quad \text{S\_WS\_ASSIGN}$$

$$\frac{\begin{array}{c}\Pi;\Gamma\vdash v\ \mathrm{ws}\\ \Pi;\Gamma;\Delta\vdash s_1\,\mathbf{ws}\\ \Pi;\Gamma;\Delta\vdash s_2\,\mathbf{ws}\end{array}}{\Pi;\Gamma;\Delta\vdash \mathbf{if}\,v\,\mathbf{then}\,s_1\,\mathbf{else}\,s_2\,\mathbf{ws}}\ \text{S\_WS\_IF}$$

$$\frac{\begin{array}{c}\Pi;\Gamma;\Delta\vdash e\,\mathbf{ws}\\ \Pi;\Gamma,x:b[\phi];\Delta\vdash s\,\mathbf{ws}\end{array}}{\Pi;\Gamma;\Delta\vdash \mathbf{let}\,x=e\,\mathbf{in}\,s\,\mathbf{ws}}\ \text{S\_WS\_LET}$$

$$\frac{\begin{array}{l}\Pi;\Gamma;\Delta\vdash s_1\,\mathbf{ws}\\ \Pi;\Gamma,x:b[\phi];\Delta\vdash s_2\,\mathbf{ws}\\ \Pi;\Gamma\vdash\tau\,\mathbf{ws}\end{array}}{\Pi;\Gamma;\Delta\vdash \mathbf{let}\ x:\{x:b|\phi\}=s_1\ \mathbf{in}\ s_2\,\mathbf{ws}}\ \text{S\_WS\_LET2}$$

$$\frac{\begin{array}{l}\mathbf{union}\,tid=\{\ \overline{C_i:\{z_i:b_i|\phi_i\}}^{\,i}\ \}\ \in\ \Pi\\ \Pi;\Gamma\vdash v\ \mathrm{ws}\\ \overline{\Pi;\Gamma,x_i:b_i[\phi_i];\Delta\vdash s_i\,\mathbf{ws}}^{\,i}\end{array}}{\Pi;\Gamma;\Delta\vdash \mathbf{match}\,v\,\mathbf{of}\,\overline{C_i\,x_i\Rightarrow s_i}^{\,i}\ \mathbf{ws}}\ \text{S\_WS\_MATCH}$$

$$\frac{\begin{array}{c}\Pi;\Gamma;\Delta\vdash e\,\mathbf{ws}\\ \Pi;\Gamma;\Delta\vdash s\,\mathbf{ws}\end{array}}{\Pi;\Gamma;\Delta\vdash \mathbf{while}\,(e)\,\mathbf{do}\,\{s\}\,\mathbf{ws}}\ \text{S\_WS\_WHILE}$$

$$\frac{\begin{array}{c}\Pi;\Gamma;\Delta\vdash s_1\,\mathbf{ws}\\ \Pi;\Gamma;\Delta\vdash s_2\,\mathbf{ws}\end{array}}{\Pi;\Gamma;\Delta\vdash s_1;s_2\,\mathbf{ws}}\ \text{S\_WS\_SEQ}$$

$\boxed{\Pi;\Gamma\vdash\phi\,\mathbf{ws}}$ Constraint ϕ is wellformed in contexts Π and Γ

$$\frac{}{\Pi;\Gamma\vdash\top\,\mathbf{ws}}\ \text{CONS\_WS\_TOP}$$

$$\frac{\begin{array}{c}\Pi;\Gamma\vdash\phi_1\,\mathbf{ws}\\ \Pi;\Gamma\vdash\phi_2\,\mathbf{ws}\end{array}}{\Pi;\Gamma\vdash(\phi_1\wedge\phi_2)\,\mathbf{ws}}\ \text{CONS\_WS\_CONJ}$$

$$\frac{\Pi;\Gamma\vdash\phi\,\mathbf{ws}}{\Pi;\Gamma\vdash\neg\phi\,\mathbf{ws}}\ \text{CONS\_WS\_NOT}$$

$$\frac{\begin{array}{c}\Pi;\Gamma\overset{\phi}{\vdash}e_1\,\mathbf{ws}\\ \Pi;\Gamma\overset{\phi}{\vdash}e_2\,\mathbf{ws}\end{array}}{\Pi;\Gamma\vdash e_1=e_2\,\mathbf{ws}}\ \text{CONS\_WS\_EQ}$$

$$\frac{\begin{array}{c}\Pi;\Gamma\vdash\phi_1\,\mathbf{ws}\\ \Pi;\Gamma\vdash\phi_2\,\mathbf{ws}\end{array}}{\Pi;\Gamma\vdash(\phi_1\Longrightarrow\phi_2)\,\mathbf{ws}}\ \text{CONS\_WS\_IMP}$$

$\boxed{\Pi;\Gamma\vdash\tau\,\mathbf{ws}}$ Type τ is wellformed in context Π, Γ and $\mathbf{D}$

$$\frac{\Pi;\Gamma,z:b[\top]\vdash\phi\,\mathbf{ws}}{\Pi;\Gamma\vdash\{z:b|\phi\}\,\mathbf{ws}}\ \text{TYPE\_WS\_TYPE}$$

$\boxed{\Pi\vdash\Gamma\sqsubseteq\Gamma'}$ Γ' is an extension of Γ

$$\frac{\Pi\vdash\Gamma\,\mathbf{ws}}{\Pi\vdash\Gamma\sqsubseteq\Gamma}\ \text{EXTEND\_G\_REFL}$$

$$\frac{\begin{array}{l}\Pi \vdash \Gamma'' \sqsubseteq \Gamma, \Gamma' \\ x \notin \mathrm{dom}(\Gamma, \Gamma') \\ \Pi \vdash \Gamma, x : b[\phi]\ \mathbf{ws}\end{array}}{\Pi \vdash \Gamma'' \sqsubseteq \Gamma, x : b[\phi], \Gamma'} \quad \text{EXTEND\_G\_INSERT}$$

$\boxed{\Pi; \Gamma \vdash \Delta \sqsubseteq \Delta'}$ Δ' is an extension of Δ

$$\frac{\Pi; \Gamma \vdash \Delta\ \mathbf{ws}}{\Pi; \Gamma \vdash \Delta \sqsubseteq \Delta} \quad \text{EXTEND\_D\_REFL}$$

$$\frac{\begin{array}{l}\Pi; \Gamma \vdash \Delta'' \sqsubseteq \Delta, \Delta' \\ u \notin \mathrm{dom}(\Delta, \Delta') \\ \Pi; \Gamma \vdash \tau\ \mathbf{ws}\end{array}}{\Pi; \Gamma \vdash \Delta'' \sqsubseteq \Delta, u : \tau, \Delta'} \quad \text{EXTEND\_D\_INSERT}$$

$\boxed{\Pi; \Gamma \vdash \tau_1 \lesssim \tau_2}$

$$\frac{\begin{array}{l}\Pi; \Gamma \vdash \{z_1 : b | \phi_1\}\ \mathbf{ws} \\ \Pi; \Gamma \vdash \{z_2 : b | \phi_2\}\ \mathbf{ws} \\ \Pi; \Gamma, z_1 : b[\phi_1] \models \phi_2[z_1/z_1]\end{array}}{\Pi; \Gamma \vdash \{z_1 : b | \phi_1\} \lesssim \{z_2 : b | \phi_2\}} \quad \text{SUBTYPE\_SUBTYPE}$$

$\boxed{\Pi; \Gamma \vdash v \Rightarrow \tau}$ Infer that type of v is τ

$$\frac{\begin{array}{c}\Pi \vdash \Gamma\ \mathbf{ws} \\ x : b[\phi] \in \Gamma\end{array}}{\Pi; \Gamma \vdash x \Rightarrow \{z : b | z = x\}} \quad \text{INFER\_V\_VAR}$$

$$\frac{\Pi \vdash \Gamma\ \mathbf{ws}}{\Pi; \Gamma \vdash () \Rightarrow \{z : \mathbf{unit} | z = ()\}} \quad \text{INFER\_V\_UNIT}$$

$$\frac{\Pi \vdash \Gamma\ \mathbf{ws}}{\Pi; \Gamma \vdash \mathbf{T} \Rightarrow \{z : \mathbf{bool} | z = \mathbf{T}\}} \quad \text{INFER\_V\_TRUE}$$

$$\frac{\Pi \vdash \Gamma\ \mathbf{ws}}{\Pi; \Gamma \vdash \mathbf{F} \Rightarrow \{z : \mathbf{bool} | z = \mathbf{F}\}} \quad \text{INFER\_V\_FALSE}$$

$$\frac{\Pi \vdash \Gamma\ \mathbf{ws}}{\Pi; \Gamma \vdash n \Rightarrow \{z : \mathbf{int} | z = n\}} \quad \text{INFER\_V\_NUM}$$

$$\frac{\begin{array}{c}\Pi; \Gamma \vdash v_1 \Rightarrow \{z_1 : b_1 | \phi_1\} \\ \Pi; \Gamma \vdash v_2 \Rightarrow \{z_2 : b_2 | \phi_2\}\end{array}}{\Pi; \Gamma \vdash (v_1, v_2) \Rightarrow \{z : b_1 * b_2 | z = (v_1, v_2)\}} \quad \text{INFER\_V\_PAIR}$$

$$\frac{\begin{array}{l}\mathbf{union}\ \mathit{tid} = \{\ \overline{C_i : \tau_i}^{\,i}\ \} \in \Pi \\ \Pi; \Gamma \vdash v \Leftarrow \tau\end{array}}{\Pi; \Gamma \vdash C_j\ v \Rightarrow \{z : \mathit{tid} | z = C_j\ v\}} \quad \text{INFER\_V\_DATA\_CONS}$$

$\boxed{\Pi; \Gamma \vdash v \Leftarrow \tau}$ Check that type of v is τ

$$\frac{\begin{array}{l}\Pi; \Gamma \vdash v \Rightarrow \{z_2 : b | \phi_2\} \\ \Pi; \Gamma \vdash \{z_2 : b | \phi_2\} \lesssim \{z_1 : b | \phi_1\}\end{array}}{\Pi; \Gamma \vdash v \Leftarrow \{z_1 : b | \phi_1\}} \quad \text{CHECK\_V\_VAL}$$

$\boxed{\Pi; \Gamma; \Delta \vdash e \Rightarrow \tau}$ Infer that type of e is τ

$$\frac{\begin{array}{l}\Pi;\Gamma\vdash\Delta\,\mathbf{ws}\\ \Pi;\Gamma\vdash v_1\Rightarrow\{z_1:\mathbf{int}|\phi_1\}\\ \Pi;\Gamma\vdash v_2\Rightarrow\{z_2:\mathbf{int}|\phi_2\}\end{array}}{\Pi;\Gamma;\Delta\vdash v_1+v_2\Rightarrow\{z_3:\mathbf{int}|z_3=v_1+v_2\}}\quad\text{INFER\_E\_PLUS}$$

$$\frac{\begin{array}{l}\Pi;\Gamma\vdash\Delta\,\mathbf{ws}\\ \Pi;\Gamma\vdash v_1\Rightarrow\{z_1:\mathbf{int}|\phi_1\}\\ \Pi;\Gamma\vdash v_2\Rightarrow\{z_2:\mathbf{int}|\phi_2\}\end{array}}{\Pi;\Gamma;\Delta\vdash v_1\leq v_2\Rightarrow\{z_3:\mathbf{bool}|z_3=v_1\leq v_2\}}\quad\text{INFER\_E\_LEQ}$$

$$\frac{\begin{array}{l}\Pi;\Gamma\vdash\Delta\,\mathbf{ws}\\ \mathbf{val}\,f:(x:b[\phi])\to\tau\ \in\ \Pi\\ \Pi;\Gamma\vdash v\Leftarrow\{z:b|\phi\}\end{array}}{\Pi;\Gamma;\Delta\vdash f\ v\Rightarrow\tau[v/x]}\quad\text{INFER\_E\_APP}$$

$$\frac{\begin{array}{l}\Pi;\Gamma\vdash\Delta\,\mathbf{ws}\\ \Pi;\Gamma\vdash v\Rightarrow\{z:b_1*b_2|\phi\}\end{array}}{\Pi;\Gamma;\Delta\vdash\mathbf{fst}\,v\Rightarrow\{z:b_1|z=\mathbf{fst}\,v\}}\quad\text{INFER\_E\_FST}$$

$$\frac{\begin{array}{l}\Pi;\Gamma\vdash\Delta\,\mathbf{ws}\\ \Pi;\Gamma\vdash v\Rightarrow\{z:b_1*b_2|\phi\}\end{array}}{\Pi;\Gamma;\Delta\vdash\mathbf{snd}\,v\Rightarrow\{z:b_2|z=\mathbf{snd}\,v\}}\quad\text{INFER\_E\_SND}$$

$$\frac{\begin{array}{l}\Pi;\Gamma\vdash\Delta\,\mathbf{ws}\\ u:\tau\in\Delta\end{array}}{\Pi;\Gamma;\Delta\vdash u\Rightarrow\tau}\quad\text{INFER\_E\_MVAR}$$

$\boxed{\Pi;\Gamma;\Delta\vdash e\Leftarrow\tau}$ Check that type of e is τ

$$\frac{\begin{array}{l}\Pi;\Gamma;\Delta\vdash e\Rightarrow\{z_2:b|\phi_2\}\\ \Pi;\Gamma\vdash\{z_2:b|\phi_2\}\lesssim\{z_1:b|\phi_1\}\end{array}}{\Pi;\Gamma;\Delta\vdash e\Leftarrow\{z_1:b|\phi_1\}}\quad\text{CHECK\_E\_EXPR}$$

$\boxed{\Pi;\Gamma;\Delta\vdash s\Leftarrow\tau}$ Check that type of s is τ

$$\frac{\begin{array}{l}\Pi;\Gamma\vdash\Delta\,\mathbf{ws}\\ \Pi;\Gamma\vdash v\Leftarrow\tau\end{array}}{\Pi;\Gamma;\Delta\vdash v\Leftarrow\tau}\quad\text{CHECK\_S\_VAL}$$

$$\frac{\begin{array}{l}u\notin\mathrm{dom}(\delta)\\ \Pi;\Gamma\vdash v\Leftarrow\tau\\ \Pi;\Gamma;\Delta,u:\tau\vdash s\Leftarrow\tau_2\end{array}}{\Pi;\Gamma;\Delta\vdash\mathbf{var}\,u:\tau:=v\,\mathbf{in}\,s\Leftarrow\tau_2}\quad\text{CHECK\_S\_VAR}$$

$$\frac{\begin{array}{l}\Pi;\Gamma\vdash\Delta\,\mathbf{ws}\\ u:\tau\in\Delta\\ \Pi;\Gamma\vdash v\Leftarrow\tau\end{array}}{\Pi;\Gamma;\Delta\vdash u:=v\Leftarrow\{z:\mathbf{unit}|\top\}}\quad\text{CHECK\_S\_ASSIGN}$$

$$\frac{\begin{array}{l}\Pi;\Gamma\vdash v\Rightarrow\{x:\mathbf{bool}|\phi_1\}\\ \Pi;\Gamma;\Delta\vdash s_1\Leftarrow\{z_1:b|(v=\mathbf{T}\wedge(\phi_1[v/x]))\implies(\phi[z_1/z])\}\\ \Pi;\Gamma;\Delta\vdash s_2\Leftarrow\{z_2:b|(v=\mathbf{F}\wedge(\phi_1[v/x]))\implies(\phi[z_2/z])\}\end{array}}{\Pi;\Gamma;\Delta\vdash\mathbf{if}\,v\,\mathbf{then}\,s_1\,\mathbf{else}\,s_2\Leftarrow\{z:b|\phi\}}\quad\text{CHECK\_S\_IF}$$

$$\frac{\begin{array}{c}\Pi;\Gamma;\Delta\vdash e\Rightarrow\{z:b|\phi\}\\ \Pi;\Gamma,x:b[\phi[x/z]];\Delta\vdash s\Leftarrow\tau\end{array}}{\Pi;\Gamma;\Delta\vdash\mathbf{let}\,x=e\,\mathbf{in}\,s\Leftarrow\tau}\ \textsc{check\_s\_let}$$

$$\frac{\begin{array}{c}\Pi;\Gamma;\Delta\vdash s_1\Leftarrow\{z:b|\phi\}\\ \Pi;\Gamma,x:b[\phi[x/z]];\Delta\vdash s_2\Leftarrow\tau\end{array}}{\Pi;\Gamma;\Delta\vdash\mathbf{let}\ x:\{z:b|\phi\}=s_1\ \mathbf{in}\ s_2\Leftarrow\tau}\ \textsc{check\_s\_let}2$$

$$\frac{\begin{array}{l}\mathbf{union}\,\mathit{tid}=\{\,\overline{C_i:\{z_i:b_i|\phi_i\}}^{\,i}\,\}\in\Pi\\ \Pi;\Gamma\vdash v\Rightarrow\{z:\mathit{tid}|\phi\}\\ \overline{\Pi;\Gamma,x_i:b_i[\phi_i[x_i/z_i]\wedge v=C_i\,x_i\wedge(\phi[v/z])];\Delta\vdash s_i\Leftarrow\tau}^{\,i}\end{array}}{\Pi;\Gamma;\Delta\vdash\mathbf{match}\,v\,\mathbf{of}\,\overline{C_i\,x_i\Rightarrow s_i}^{\,i}\Leftarrow\tau}\ \textsc{check\_s\_match}$$

$$\frac{\begin{array}{c}\Pi;\Gamma;\Delta\vdash e\Leftarrow\{z:\mathbf{bool}|\top\}\\ \Pi;\Gamma;\Delta\vdash s\Leftarrow\{z:\mathbf{unit}|\top\}\end{array}}{\Pi;\Gamma;\Delta\vdash\mathbf{while}\,(e)\,\mathbf{do}\,\{s\}\Leftarrow\{z:\mathbf{unit}|\top\}}\ \textsc{check\_s\_while}$$

$$\frac{\begin{array}{l}\Pi;\Gamma;\Delta\vdash s_1\Leftarrow\{z:\mathbf{unit}|\top\}\\ \Pi;\Gamma;\Delta\vdash s_2\Leftarrow\tau\end{array}}{\Pi;\Gamma;\Delta\vdash s_1;s_2\Leftarrow\tau}\ \textsc{check\_s\_seq}$$

$\boxed{\Pi;\Gamma\vdash \mathit{def}_1\,..\,\mathit{def}_n\leadsto\Pi'}$

$$\frac{\begin{array}{c}\mathbf{val}\,f:(x:b[\phi])\to\tau\in\Pi\\ \Pi;\Gamma,x:b[\phi];\epsilon\vdash s\Leftarrow\tau\end{array}}{\Pi;\Gamma\vdash\mathbf{function}\,f(x)=s\leadsto\Pi,\mathbf{function}\,f(x)=s}\ \textsc{check\_defs\_fundef}$$

$$\frac{}{\Pi;\Gamma\vdash\mathbf{val}\,f:(x:b[\phi])\to\tau\leadsto\Pi,\mathbf{val}\,f:(x:b[\phi])\to\tau}\ \textsc{check\_defs\_valspec}$$

$$\frac{}{\Pi;\Gamma\vdash\mathbf{union}\,\mathit{tid}=\{\,\overline{C_i:\tau_i}^{\,i}\,\}\leadsto\Pi,\mathbf{union}\,\mathit{tid}=\{\,\overline{C_i:\tau_i}^{\,i}\,\}}\ \textsc{check\_defs\_uniondef}$$

$$\frac{\begin{array}{c}\Pi;\Gamma\vdash\mathit{def}\leadsto\Pi'\\ \Pi';\Gamma\vdash\mathit{def}_1\,..\,\mathit{def}_n\leadsto\Pi''\end{array}}{\Pi;\Gamma\vdash\mathit{def}\ \mathit{def}_1\,..\,\mathit{def}_n\leadsto\Pi''}\ \textsc{check\_defs\_defs}$$

$\boxed{\Pi;\Gamma\vdash p}$

$$\frac{\begin{array}{c}\Pi;\Gamma\vdash\mathit{def}_1\,..\,\mathit{def}_n\leadsto\Pi'\\ \Pi';\Gamma;\epsilon\vdash s\Leftarrow\{z:\mathbf{int}|\top\}\end{array}}{\Pi;\Gamma\vdash\mathit{def}_1;\,..;\mathit{def}_n;s}\ \textsc{check\_program\_prog}$$

$\boxed{\Pi\vdash\Delta\sim\delta}$

$$\frac{\begin{array}{l}\delta=u_1\to v_1,\,..\,,u_n\to v_n\\ \Delta=u_1:\tau_1,\,..\,,u_n:\tau_n\\ \Pi;\cdot\vdash v_1\Leftarrow\tau_1\quad..\quad\Pi;\cdot\vdash v_n\Leftarrow\tau_n\end{array}}{\Pi\vdash\Delta\sim\delta}\ \textsc{dsim\_dsim}$$

$\boxed{\Pi;\Delta\vdash(\delta,s)\Leftarrow\tau}$ Program state typing judgement

$$\frac{\begin{array}{c}\Pi\vdash\Delta\sim\delta\\ \Pi;\cdot;\Delta\vdash s\Leftarrow\tau\end{array}}{\Pi;\Delta\vdash(\delta,s)\Leftarrow\tau}\ \textsc{check\_redex\_stmt}$$

$\boxed{\Pi\vdash\langle\delta,s\rangle\to\langle\delta',s'\rangle}$ One step reduction

$$\frac{}{\Pi \vdash \langle \delta, \mathbf{if}\ \mathbf{T}\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2 \rangle \rightarrow \langle \delta, s_1 \rangle} \quad \text{REDUCE\_IF\_TRUE}$$

$$\frac{}{\Pi \vdash \langle \delta, \mathbf{if}\ \mathbf{F}\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2 \rangle \rightarrow \langle \delta, s_2 \rangle} \quad \text{REDUCE\_IF\_FALSE}$$

$$\frac{}{\Pi \vdash \langle \delta, \mathbf{let}\ x = v\ \mathbf{in}\ s \rangle \rightarrow \langle \delta, s[v/x] \rangle} \quad \text{REDUCE\_LET\_VALUE}$$

$$\frac{v_1 + v_2 = v}{\Pi \vdash \langle \delta, \mathbf{let}\ x = v_1 + v_2\ \mathbf{in}\ s \rangle \rightarrow \langle \delta, \mathbf{let}\ x = v\ \mathbf{in}\ s \rangle} \quad \text{REDUCE\_LET\_PLUS}$$

$$\frac{v_1 \leq v_2 = v}{\Pi \vdash \langle \delta, \mathbf{let}\ x = v_1 \leq v_2\ \mathbf{in}\ s \rangle \rightarrow \langle \delta, \mathbf{let}\ x = v\ \mathbf{in}\ s \rangle} \quad \text{REDUCE\_LET\_LEQ}$$

$$\frac{\begin{array}{c}\mathbf{val}\, f : (x : b[\phi]) \rightarrow \tau \in \Pi \\ \mathbf{function}\, f(x) = s_1 \in \Pi\end{array}}{\Pi \vdash \langle \delta, \mathbf{let}\ y = f\ v\ \mathbf{in}\ s_2 \rangle \rightarrow \langle \delta, \mathbf{let}\ \ y : \tau[v/x] = s_1[v/x]\ \ \mathbf{in}\ \ s_2 \rangle} \quad \text{REDUCE\_LET\_APP}$$

$$\frac{}{\Pi \vdash \langle \delta, \mathbf{let}\ x = \mathbf{fst}\,(v_1, v_2)\ \mathbf{in}\ s \rangle \rightarrow \langle \delta, \mathbf{let}\ x = v_1\ \mathbf{in}\ s \rangle} \quad \text{REDUCE\_LET\_FST}$$

$$\frac{}{\Pi \vdash \langle \delta, \mathbf{let}\ x = \mathbf{snd}\,(v_1, v_2)\ \mathbf{in}\ s \rangle \rightarrow \langle \delta, \mathbf{let}\ x = v_2\ \mathbf{in}\ s \rangle} \quad \text{REDUCE\_LET\_SND}$$

$$\frac{v = \delta(u)}{\Pi \vdash \langle \delta, \mathbf{let}\ x = u\ \mathbf{in}\ s \rangle \rightarrow \langle \delta, \mathbf{let}\ x = v\ \mathbf{in}\ s \rangle} \quad \text{REDUCE\_LET\_MVAR}$$

$$\frac{u \notin \mathrm{dom}(\delta)}{\Pi \vdash \langle \delta, \mathbf{var}\ u : \tau := v\ \mathbf{in}\ s \rangle \rightarrow \langle \delta[u \mapsto v], s \rangle} \quad \text{REDUCE\_MVAR\_DECL}$$

$$\frac{\delta' = \delta[u \mapsto v]}{\Pi \vdash \langle \delta, u := v \rangle \rightarrow \langle \delta', () \rangle} \quad \text{REDUCE\_MVAR\_ASSIGN}$$

$$\frac{\Pi \vdash \langle \delta, s_1 \rangle \rightarrow \langle \delta', s_1' \rangle}{\Pi \vdash \langle \delta, s_1; s \rangle \rightarrow \langle \delta', s_1'; s \rangle} \quad \text{REDUCE\_SEQ1}$$

$$\frac{}{\Pi \vdash \langle \delta, (); s \rangle \rightarrow \langle \delta, s \rangle} \quad \text{REDUCE\_SEQ2}$$

$$\frac{}{\Pi \vdash \langle \delta, \mathbf{let}\ \ x : \tau = v\ \ \mathbf{in}\ \ s_2 \rangle \rightarrow \langle \delta, s_2[v/x] \rangle} \quad \text{REDUCE\_LET2\_VAL}$$

$$\frac{\Pi \vdash \langle \delta, s_1 \rangle \rightarrow \langle \delta', s_1' \rangle}{\Pi \vdash \langle \delta, \mathbf{let}\ \ x : \tau = s_1\ \ \mathbf{in}\ \ s_2 \rangle \rightarrow \langle \delta', \mathbf{let}\ \ x : \tau = s_1'\ \ \mathbf{in}\ \ s_2 \rangle} \quad \text{REDUCE\_LET2\_STMT}$$

$$\frac{}{\Pi \vdash \langle \delta, \mathbf{match}\,(C_j\ v)\ \mathbf{of}\ \overline{C_i\ x_i \Rightarrow s_i}^{\,i} \rangle \rightarrow \langle \delta, s_j[v/x_j] \rangle} \quad \text{REDUCE\_MATCH}$$

$$\frac{x\ \mathbf{fresh}}{\Pi \vdash \langle \delta, \mathbf{while}\,(e)\ \mathbf{do}\,\{s\} \rangle \rightarrow \langle \delta, \mathbf{let}\ x = e\ \mathbf{in}\ \mathbf{if}\ x\ \mathbf{then}\,(s; \mathbf{while}\,(e)\ \mathbf{do}\,\{s\})\ \mathbf{else}\,() \rangle} \quad \text{REDUCE\_WHILE}$$

$\boxed{\Pi \vdash \langle \delta_1, s_1 \rangle \xrightarrow{*} \langle \delta_2, s_2 \rangle}$ Multi-step reduction

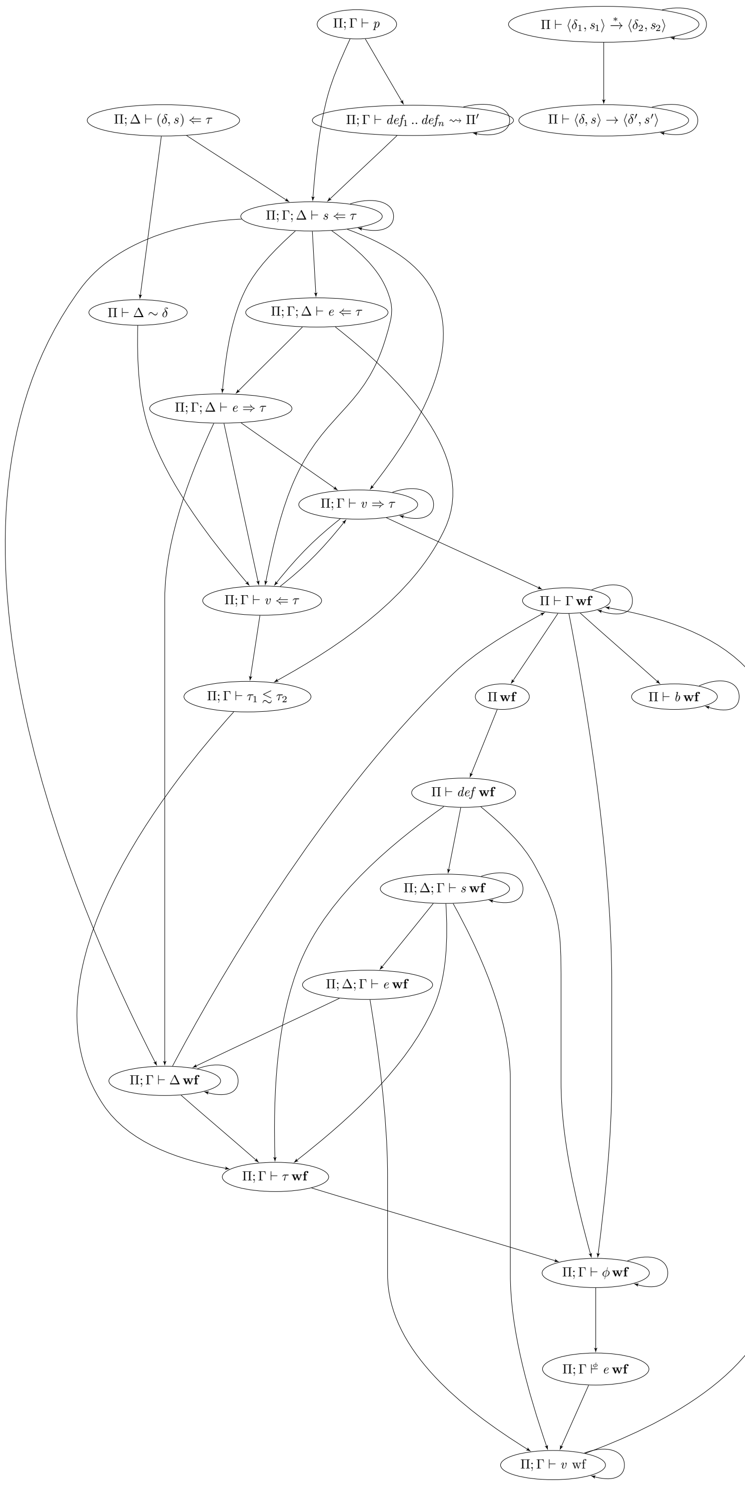
$$\frac{\Pi \vdash \langle \delta_1, s_1 \rangle \rightarrow \langle \delta_2, s_2 \rangle}{\Pi \vdash \langle \delta_1, s_1 \rangle \xrightarrow{*} \langle \delta_2, s_2 \rangle} \quad \text{REDUCE\_MANY\_SINGLE\_STEP}$$

$$\frac{\begin{array}{c}\Pi \vdash \langle \delta_1, s_1 \rangle \rightarrow \langle \delta_2, s_2 \rangle \\ \Pi \vdash \langle \delta_2, s_2 \rangle \xrightarrow{*} \langle \delta_3, s_3 \rangle\end{array}}{\Pi \vdash \langle \delta_1, s_1 \rangle \xrightarrow{*} \langle \delta_3, s_3 \rangle} \quad \text{REDUCE\_MANY\_MANY\_STEP}$$

13 Appendix 2 - List of Theorems

List of Theorems

14 Appendix 3 - Rule Dependency Graph


$\Pi; \Gamma \vdash p$
$\Pi \vdash \langle \delta_1, s_1 \rangle \xrightarrow{*} \langle \delta_2, s_2 \rangle$
$\Pi; \Delta \vdash \langle \delta, s \rangle \Leftarrow \tau$
$\Pi; \Gamma \vdash def_1 \,..\, def_n \rightsquigarrow \Pi'$
$\Pi \vdash \langle \delta, s \rangle \to \langle \delta', s' \rangle$
$\Pi; \Gamma; \Delta \vdash s \Leftarrow \tau$
$\Pi \vdash \Delta \sim \delta$
$\Pi; \Gamma; \Delta \vdash e \Leftarrow \tau$
$\Pi; \Gamma; \Delta \vdash e \Rightarrow \tau$
$\Pi; \Gamma \vdash v \Rightarrow \tau$
$\Pi; \Gamma \vdash v \Leftarrow \tau$
$\Pi \vdash \Gamma\ \mathbf{wf}$
$\Pi; \Gamma \vdash \tau_1 \lesssim \tau_2$
$\Pi\ \mathbf{wf}$
$\Pi \vdash b\ \mathbf{wf}$
$\Pi \vdash def\ \mathbf{wf}$
$\Pi; \Delta; \Gamma \vdash s\ \mathbf{wf}$
$\Pi; \Delta; \Gamma \vdash e\ \mathbf{wf}$
$\Pi; \Gamma \vdash \Delta\ \mathbf{wf}$
$\Pi; \Gamma \vdash \tau\ \mathbf{wf}$
$\Pi; \Gamma \vdash \phi\ \mathbf{wf}$
$\Pi; \Gamma \models^{\phi} e\ \mathbf{wf}$
$\Pi; \Gamma \vdash v\ \mathrm{wf}$

References

[1] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[2] Joshua Dunfield and Neelakantan R. Krishnaswami. Complete and easy bidirectional typechecking for higher-rank polymorphism. In *International Conference on Functional Programming (ICFP)*, September 2013. `arXiv:1306.6032[cs.PL]`.

[3] Patrick M. Rondon, Ming Kawaguci, and Ranjit Jhala. Liquid types. *SIGPLAN Not.*, 43(6):159–169, June 2008.

[4] Niki Vazou, Patrick M. Rondon, and Ranjit Jhala. Abstract refinement types. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7792 LNCS:209–228, 2013.