# Sums of Uncertainty: Refinements Go Gradual

Long version of paper to appear at POPL 2017, including supplementary material

Khurram A. Jafery Joshua Dunfield

University of British Columbia
Vancouver, Canada
{kjafery,joshdunf}@cs.ubc.ca

## Abstract

A long-standing shortcoming of statically typed functional languages is that type checking does not rule out pattern-matching failures (run-time match exceptions). Refinement types distinguish different values of datatypes; if a program annotated with refinements passes type checking, pattern-matching failures become impossible. Unfortunately, refinement is a monolithic property of a type, exacerbating the difficulty of adding refinement types to nontrivial programs.

Gradual typing has explored how to incrementally move between static typing and dynamic typing. We develop a type system of *gradual sums* that combines refinement with imprecision. Then, we develop a bidirectional version of the type system, which rules out excessive imprecision, and give a type-directed translation to a target language with explicit casts. We prove that the static sublanguage cannot have match failures, that a well-typed program remains well-typed if its type annotations are made less precise, and that making annotations less precise causes target programs to fail later. Several of these results correspond to criteria for gradual typing given by Siek et al. (2015).

***Categories and Subject Descriptors*** F.3.3 [*Mathematical Logic and Formal Languages*]: Studies of Program Constructs—Type structure

***Keywords*** gradual typing, refinement types

## 1. Introduction

A central feature of statically typed functional languages is pattern matching over user-defined datatypes that combine several fundamental constructs: sum types (for example, an element of a bool datatype can be *either* True or False), recursive types (such as lists), and polymorphic types. The aspect of ML datatypes that corresponds to sum types is the focus of this paper.

Static typing is said to catch run-time errors—at least, errors that would manifest in a dynamically typed language as *tag check failures*, such as subtracting a string from a number. Using the venerable encoding of dynamic typing as injections into a datatype Dynamic (Abadi et al. 1991), these tag check failures become errors raised in the "fall-through" arm of a case expression over Dynamic. The impossibility of such errors is a convincing argument in favour of static typing.



Yet Standard ML programmers frequently write code that is essentially the same as the scorned operations on Dynamic—and that has the same unfortunate risk of run-time errors. The definition of SML (Milner et al. 1997) requires compilers to accept *nonexhaustive* case expressions, which do not cover all the possible instances of the datatype. A nonexhaustive case expression is isomorphic to an implicit tag check over Dynamic: the non-error case is the only one written out explicitly, while an error case is inserted by the sneaky compiler.

In fairness, the definition encourages compilers to warn about nonexhaustive case expressions. But this only causes programmers to write their own "`raise Match`" arms, even when the fall-through case is impossible because of an invariant known by the programmer. This leads to verbose code. In response, Freeman and Pfenning (1991) developed datasort refinements that can encode many invariants about datatypes, allowing compilers to accept "nonexhaustive" case expressions when they are known to cover all *possible* cases. For case analyses of refined types, the nonexhaustiveness *warning* becomes a nonexhaustiveness *error*, which the programmer should solve by declaring and using refinements of the datatype.

Unfortunately, this approach is all-or-nothing: either a type is refined and the compiler rejects a nonexhaustive match over it, or the type is not refined and the compiler issues a noncommittal warning. In practice, programmers may want to migrate code written with unrefined types to code that uses refined types; doing this in a single pass over a nontrivial program is extremely difficult. Instead, programmers should be able to add type annotations *gradually*. This was essentially the motivation for gradual typing (Siek and Taha 2006), except that, where they contemplated migration from dynamically typed code to statically typed code, we are interested in migration from code that is statically typed (modulo nonexhaustiveness) to code that is *more* statically typed.

Gradual typing is about the possibility of uncertainty: in some cases, one knows exactly what type one has; in other cases, one does not even know whether something is an integer. In this paper, we always know whether something is an integer (or a function, etc.); uncertainty is possible, but only about sum types. This is like the uncertainty of SML datatypes, with one key difference: we allow SML-style uncertainty *and* refinement-style certainty.

As an example, consider a red-black tree library that passes the SML type checker, but does not use refinement types. Datasort refinements can express the colour invariant, which says that every red node's children must be black. By reasoning about how the library functions should work, a programmer can add annotations that say when the colour invariant should hold, which the refinement type checker will verify. With gradual refinements, this reasoning can be done gradually and in tandem with testing. In fact, the programmer could start by annotating a single function $r$. If all test cases use $r$ in accordance with its refinement type annotation, the programmer gains confidence that the annotation is correct; if any tests violate the annotation, then either the annotation is wrong, or there is a bug somewhere else. Thus, the more precise invariants guaranteed by refinements can be verified piecemeal.

***Contributions.*** We make the following contributions:

- We define a type assignment system of *gradual sums* that includes both static *refinement sums* and *dynamic sums*. Programs, and even individual types, can be partly static and partly dynamic. However, this system does not readily yield an algorithm, and it allows typing derivations that are *gratuitously* dynamic (more dynamic than indicated by the programmer's type annotations), which give rise to gratuitous run-time errors.
- We define a bidirectional type system that is easy to implement and suppresses gratuitous dynamism, and prove that it corresponds to the type assignment system. We also prove that a well-typed program remains well-typed if its type annotations are made less precise (more dynamic).
- We define a type-directed translation to a target language with explicit casts. We prove that, given one program with two sets of type annotations (one more precise than the other), the more precisely typed one "fails earlier": either they produce the same result, or they both fail, or the more precisely typed program fails earlier. (For technical reasons, part of this result uses a slightly different version of the translation.)
- We define static and dynamic fragments of the source type system. The static fragment is related to classic datasort refinement type systems; the dynamic fragment is related to Standard ML. We prove that translating a program in the static fragment yields a program that cannot raise `Match`.

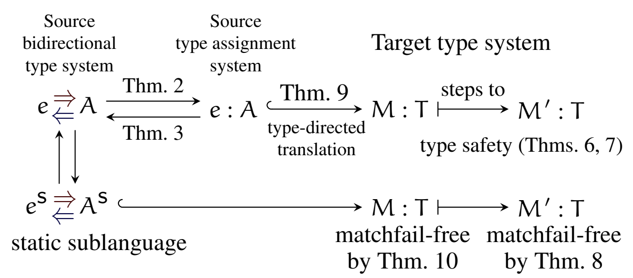



**Figure 1.** Some key results

Figure 1 depicts some of the results: source programs $e$ are translated to target terms $M$, which step to $M'$, preserving typing; source programs $e^S$ with only static types are translated to target terms with no match failures.

For space reasons, lemmas, proofs, and a few definitions can be found in the supplementary material.

## 2. Overview

We define a type system that has one of the essential capabilities of datasort refinements: the types can express the knowledge that a value is a *particular* alternative of a datatype; for example, that a value is not simply a list—either $\mathsf{Nil}$ or $\mathsf{Cons}(\dots)$—but specifically $\mathsf{Cons}(\dots)$. We represent this knowledge through sum types, not through the usual form of datasort refinements, but that is not the important difference.

- Like conventional datatype systems and datasort refinement systems, we can express that a value is either $\mathtt{inj}_1\, e_1$ where $e_1$ has type $A_1$ or $\mathtt{inj}_2\, e_2$ where $e_2$ has type $A_2$. Like datasort refinement systems, we only allow an exhaustive (two-armed) case expression over such a type: if we don't know which injection it is, the programmer must handle both cases. This is a standard sum type $A_1 + A_2$.
- Like datasort refinement systems, we can express that a value must be a particular injection. We use a *subscript sum* $A_1 +_k A_2$ for the type of the kth injection into $A_1 + A_2$. For example, $\mathtt{inj}_2\, \mathsf{True}$ has type $\mathsf{Int} +_2 \mathsf{Bool}$, but $\mathtt{inj}_1\, 5$ has type $\mathsf{Int} +_1 \mathsf{Bool}$. Also like datasort refinement systems, we allow case expressions over such types to have just one arm, because we know which injection we have; there is no need to handle an impossible case.
- Like conventional datatype systems, but unlike datasort refinement systems, we can also express that we don't know which injection we have, *but want to allow nonexhaustive matches*: the *dynamic sum* $A_1 +^? A_2$ can be deconstructed by a one-armed case expression. If, at run time, the specified arm does not match the scrutinee, it is a run-time error.

The three sum types $+$, $+_1$, and $+_2$ are essentially a datasort refinement system. Following datasort refinement systems, $A_1 +_1 A_2$ and $A_1 +_2 A_2$ are subtypes of $A_1 + A_2$.

We can also make $+^?$ a subtype of $+$: the only elimination form permitted for $+$ is a two-armed case, which is always safe. But $+^?$ must not be a subtype of $+_1$ and $+_2$, because $+^?$ contains both left and right injections; through subsumption, we could use a one-armed case on the left injection $\mathtt{inj}_1$ to eliminate a value of type $+_2$, which would fail at run time.

This yields the following subtype relation:

$$\begin{tikzcd}[column sep=small] & A_1 + A_2 \arrow[dl, dash] \arrow[dd, dash] \arrow[dr, dash] & \\ A_1 +_1 A_2 & & A_1 +_2 A_2 \\ & A_1 +^? A_2 & \end{tikzcd}$$

For brevity, we can omit $A_1$ and $A_2$ from the diagram.

$$\begin{tikzcd}[column sep=small] & + \arrow[dl, dash] \arrow[dd, dash] \arrow[dr, dash] & \\ +_1 & & +_2 \\ & +^? & \end{tikzcd}$$

***Comparison to datasort refinements.*** Our type $A_1 + A_2$ corresponds to the *top datasort* of a datatype—the datasort that contains all the values of that datatype. A case expression on $+$ must provide two arms, one for each injection.

Our type $A_1 +_1 A_2$ corresponds to a datasort that includes exactly the values of the form $c_1(v_1)$ where $v_1 : A_1$; similarly, $A_1 +_2 A_2$ corresponds to a datasort whose values are $c_2(v_2)$ where $v_2 : A_2$.

In contrast, our type $A_1 +^? A_2$ corresponds to the *unrefined* datatype. In datasort refinement systems, unrefined datatypes are part of the unrefined type system; the top datasort for a datatype contains the same values as the unrefined datatype, and is often notated in exactly the same way—but the unrefined datatype is not usable as a datasort. In contrast, both $+$ and $+^?$ are types in our system. Moreover, they can be freely combined.

### 2.1 Developing Typing and Subtyping

***Verificationists and pragmatists.*** In the *verificationist* approach to type theory, followed by Gentzen (1934) and Martin-Löf (1996), introduction forms are taken as the definition of a type; for example, a boolean type is defined by its constructors $\mathsf{True}$ and $\mathsf{False}$. The elimination forms are secondary. In the *pragmatist* approach considered by Dummett (1991) and Zeilberger (2009), elimination forms are taken as the definition, and the introduction forms are secondary. For example, a boolean type is defined primarily by its elimination form (say, an if-then-else expression).

In our setting, neither strict verificationism nor strict pragmatism seems adequate. Verificationism serves refinements well: the introduction rules directly express the intuition that refinements identify subsets of values. But introduction rules alone cannot distinguish $A_1 + A_2$ and $A_1 +^? A_2$, because they have identical sets of inhabiting values (namely, all $\mathtt{inj}_1\, v_1$ and $\mathtt{inj}_2\, v_2$ such that $v_1 : A_1$ and $v_2 : A_2$). The difference must lie in the elimination forms: only a two-armed case can eliminate $+$, while $+^?$ can be eliminated by a two-armed case *or* a one-armed case (since the point is to allow nonexhaustive matches). To start from a better-understood foundation, we begin with the introduction rules.

Designing a type system can require trading off simplicity in one set of rules for complexity in another. We choose to minimize the number of typing rules, even though it leads to more complicated subtyping.

***Introduction rules.*** Sum types need introduction forms. Since $+_1$ should contain only left injections, and $+_2$ should contain only right injections, we could have a rule

$$\frac{\Gamma \vdash e : A_k}{\Gamma \vdash (\mathtt{inj}_k\, e) : (A_1 +_k A_2)}\ +_k\text{Intro}$$

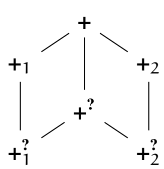
(This rule is really two rules, one for $(\mathtt{inj}_1\, v)$ with a premise $\Gamma \vdash e : A_1$ and one for $(\mathtt{inj}_2\, v)$ with a premise $\Gamma \vdash e : A_2$.)

Combined with subsumption, this rule gives the desired inhabitants to $+$, that is, both left and right injections. However, it does not add any inhabitants to $+^?$, so we could add another rule:

$$\frac{\Gamma \vdash e : A_k}{\Gamma \vdash (\mathtt{inj}_k\, e) : (A_1 +^? A_2)}\ +^?\text{Intro}$$

This goes against our goal of minimizing the number of typing rules: now there are *two* rules that type $\mathtt{inj}_k\, e$ directly, that is, without using subsumption. The types $+_k$ (given by $+_k$Intro) and $+^?$ (given by $+$Intro) are not in a subtyping relation with each other—neither is a subtype of the other. Hence, neither rule encompasses the other, and both are required.

We can avoid this nondeterminism by adding more sum types. By placing the additional sum types at the bottom of the subtyping relation, we can write a single introduction rule that will (through subsumption) populate all of our types with the desired injections.

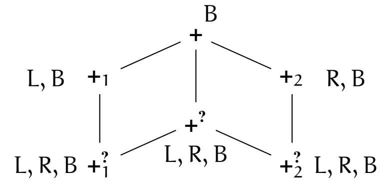



Now, we need only one introduction rule:

$$\frac{\Gamma \vdash e : A_k}{\Gamma \vdash (\mathtt{inj}_k\, e) : (A_1 +^?_k A_2)}\ +^?_k\text{Intro}$$

We can think of $+^?_1$ and $+^?_2$ as "innate" types: when an injection $\mathtt{inj}_k$ is created, it has type $+^?_k$. Through subtyping, we can interpret $+^?_k$ as $+_k$, or as the dynamic sum $+^?$.

***Elimination rules.*** To design the elimination rules, it is helpful to annotate the subtyping diagram with the elimination forms that each type should allow. We write L for a one-armed case expression on the left injection ($\mathtt{inj}_1$), R for a one-armed case on the right injection ($\mathtt{inj}_2$), and B for a two-armed case.




According to this diagram, all types support a two-armed case expression B. The types $+_1$ and $+^?_1$ are inhabited only by $\mathtt{inj}_1$, so they support the left one-armed case L; similarly, $+_2$ and $+^?_2$ support the right one-armed case R. However, $+^?_1$ and $+^?_2$ are subtypes of $+^?$, so by subsumption they also support the "wrong" one-armed cases. The dynamic sum $+^?$ supports all three eliminations, with the risk of failing at run time.

Handling the two-armed case expression is straightforward: all the sum types support that elimination form, and all the sum types are subtypes of $+$, so we can write a single rule that types the scrutinee with $+$. Given $e : (A_1\, \phi\, A_2)$ where $\phi$ is any of our sum types, subsumption can be used to derive $e : (A_1 + A_2)$.

$$\frac{\Gamma \vdash e : (A_1 + A_2) \qquad \begin{array}{l}\Gamma, x_1 : A_1 \vdash e_1 : B\\ \Gamma, x_2 : A_2 \vdash e_2 : B\end{array}}{\Gamma \vdash \mathtt{case}(e, \mathtt{inj}_1\, x_1.e_1, \mathtt{inj}_2\, x_2.e_2) : B}\ +\text{Elim}$$

One-armed case expressions are more troublesome. Consider a left one-armed case, which matches only values of the form $\mathtt{inj}_1\, v$. Any subtype of $+_1$ will work, so we can write a rule that handles $+_1$ and $+^?_1$ (and symmetrically, $+_2$ and $+^?_2$). However, $+^?$ should support a left one-armed case, but $+^?$ is not a subtype of $+_1$, leading us to a second rule that handles $+^?$.

Since $+^?$ supports one-armed cases, it violates a type-theoretic principle: the introduction and elimination rules of a logical connective should be in *harmony*—that is, they should be *locally sound* (Dummett 1991) and *locally complete* (Pfenning and Davies 2001). Local soundness holds when the elimination rules are not more powerful than the introduction rules. Consider some standard rules for pairs:

$$\frac{\Gamma \vdash e_1 : A_1 \qquad \Gamma \vdash e_2 : A_2}{\Gamma \vdash (e_1, e_2) : (A_1 \times A_2)} \qquad \frac{\Gamma \vdash e : (A_1 \times A_2)}{\Gamma \vdash (\mathtt{proj}_k\ e) : A_k}$$

These rules are locally sound: given something of type $(A_1 \times A_2)$, projection can only extract things of type $A_1$ and $A_2$.

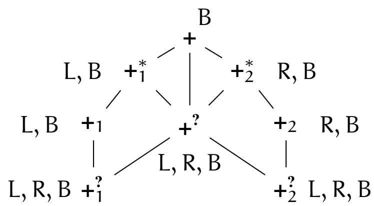
Dually, local completeness says that the elimination rules can extract all the information used in the introduction rules. (For a concise explanation of harmony, see Pfenning (2009).)

When the Curry–Howard correspondence holds, a type is inhabited iff the corresponding proposition is provable. Consider the following derivation (eliding empty contexts):

$$\dfrac{\dfrac{\dfrac{e : A_1}{(\mathtt{inj}_1\ e) : (A_1 +^?_1 A_2)} \quad (A_1 +^?_1 A_2) \leq (A_1 +^? A_2)}{(\mathtt{inj}_1\ e) : (A_1 +^? A_2)} \qquad x : A_2 \vdash x : A_2}{\mathtt{case}(\mathtt{inj}_1\, e, \mathtt{inj}_2\, x.x) : A_2}$$

By constructing $\mathtt{inj}_1\, e$, we have shown that $A_1$ is inhabited. By subsumption, $\mathtt{inj}_1\, e$ has type $A_1 +^? A_2$. An elimination rule for $+^?$ must permit a one-armed case on the second injection, ostensibly having type $A_2$. Simply returning $x$ as the result of the $\mathtt{case}$ should show that the proposition corresponding to $A_2$ is provable. But we never constructed something of type $A_2$, so $+^?$ does not satisfy local soundness.

As we did for the introduction forms, a single elimination rule *can* suffice: we just need more sum types. For the introduction forms, we added types at the bottom of the subtyping relation. Since eliminations should behave dually, we will add types at (or, at least, near) the *top* of the subtyping relation.




The types $+^*_1$ and $+^*_2$ support exactly the same eliminations as the subscript sums $+_1$ and $+_2$, but unlike the subscript sums, they are supertypes of the dynamic sum $+^?$.

Then the single elimination rule for one-armed cases is

$$\dfrac{\Gamma \vdash e : (A_1 +^*_k A_2) \qquad \Gamma, x : A_k \vdash e_k : B}{\Gamma \vdash \mathtt{case}(e, \mathtt{inj}_k\, x.e_k) : B}\ {+^*_k}\text{Elim}$$

We could simplify the diagram slightly by removing the edge from $+^?$ to $+$, since we now have an alternate routing via the $+^*_k$ types.

***The high-water mark.*** Have we added enough sum types? We believe so. First, the additional types (beyond $+$, $+_1$, $+_2$ and $+^?$) are motivated by limiting the number of typing rules. Second, there seem to be no other types that could be useful. Consider the following table:

| | elimination forms supported | | | |
|---|---|---|---|---|
| inhabitants | B only | B and L | B and R | B, L, and R |
| $\mathtt{inj}_1$ | note (a) | $+_1$ | note (b) | $+^?_1$ |
| $\mathtt{inj}_2$ | note (a) | note (b) | $+_2$ | $+^?_2$ |
| $\mathtt{inj}_1$ and $\mathtt{inj}_2$ | $+$ | $+^*_1$ | $+^*_2$ | $+^?$ |

In the spaces marked "note (a)", such a type would pointlessly restrict the possible elimination forms: the top left space would be a type that could only be eliminated by a two-armed case ("B only"), but was inhabited only by left injections $\mathtt{inj}_1$.

In the spaces marked "note (b)", such a type would allow one-armed cases that *always* fail: a left one-armed case L on $\mathtt{inj}_2$, or a right one-armed case R on $\mathtt{inj}_1$. We provide $+^?$ to give programmers the freedom to use one-armed cases that may fail; it seems pointless to give them one-armed cases that are *guaranteed* to fail.

If anything, we may have more sum types than we want in practice: having fewer typing rules is good, but showing $+^*_1$ or $+^?_2$ in a compiler error message seems unhelpful.

### 2.2 Developing Precision

Our ultimate goal is a language in which precisely typed code and imprecisely typed code can coexist. In precisely typed code, the impossibility of match failures is a consequence of typing. In imprecisely typed code, bugs may lead to match failures, but imprecisely typed code can be correct: a one-armed case expression may be exhaustive in practice, thanks to some invariant not expressed through the type system.

The approach to typing and subtyping, developed above, already permits some forms of coexistence. For example, if a function f expects a sum type $+$ and we have some $x$ of type $+^?$, we can pass $x$ to $f$. In the derivation below, $\Gamma = f : (A_1 + A_2) \to B, x : (A_1 +^? A_2)$.

$$\dfrac{\Gamma \vdash f : (A_1 + A_2) \to B \qquad \dfrac{\Gamma \vdash x : A_1 +^? A_2 \qquad A_1 +^? A_2 \le A_1 + A_2}{\Gamma \vdash x : A_1 + A_2}}{\Gamma \vdash f\,x : B}$$

What about the reverse situation? Suppose a function $g$ from the imprecisely typed part of the program expects $+^?$, and we want to pass something of type $+$. This is possible, but annoying: we have to use a two-armed case to decompose the sum, and immediately rebuild it at type $+^?$. Here, $\Gamma = g : (A_1 +^? A_2) \to B, y : (A_1 + A_2)$.

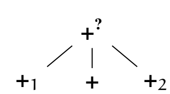
$$\dfrac{\dots \qquad \begin{array}{c}\Gamma, x_1 : A_1 \vdash \mathtt{inj}_1\, x_1 : (A_1 +^? A_2) \\ \Gamma, x_2 : A_2 \vdash \mathtt{inj}_2\, x_2 : (A_1 +^? A_2)\end{array}}{\Gamma \vdash g\ \big(\mathtt{case}(y, \mathtt{inj}_1\, x_1.\mathtt{inj}_1\, x_1, \mathtt{inj}_2\, x_2.\mathtt{inj}_2\, x_2)\big) : B}$$

To support directly calling imprecise code from precise code, we develop *precision relations* on sum constructors and types. These relations are inspired by precision relations developed in gradual typing, e.g. Siek and Vachharajani (2008) and Garcia et al. (2016), where ? (or $\star$) is an unknown, and thus very imprecise, type.

Our static sums $+$, $+_1$, $+_2$ are precise in the sense that the "reach" of their information is known. If we have a closed value $v$ of type $A_1 + A_2$, the type system "knows" only that $v$ is either a left or right injection, with no further information. So the type system rejects a one-armed case on $v$.

On the other hand, the dynamic sum $+^?$ is *imprecise*. Some programs that use $+^?$ will have run-time match failures, but some programs that use $+^?$ will *not* have such failures, even some that use one-armed cases. If such one-armed cases always succeed, it is because the program follows invariants that are not expressed in the types—but which may be known by the programmer.

So we would expect $+$ to be more precise than $+^?$, notated $+ \sqsubseteq +^?$ (which can also be read "$+$ is less imprecise than $+^?$"). What about $+_1$ and $+_2$? They should be more precise than $+^?$; indeed, $+^?$ should be more *imprecise* than everything else. How do $+_1$ and $+$ compare? It is true that $+_1$ has fewer inhabitants than $+$, but precision is not subtyping. All the static sums have the same degree of *certainty*: they are equally certain about different propositions (being a left injection, being a right injection, or being either). Thus, we will put $+_1$, $+_2$ and $+$ together at the bottom of the precision relation $\sqsubseteq$ (they are the *least imprecise*), with $+^?$ at the top:




What properties should precision have? In gradual typing, an important property of precision is that a program should remain well-typed when type annotations are made *less* precise. In the limit, we should be able to replace all static sums in annotations with $+^?$. We call this property *varying precision*; it is part of the "gradual guarantee" of Siek et al. (2015). (Making annotations *more* precise does not necessarily preserve typing: for example, changing a $+^?$ annotation on $\mathtt{inj}_2\,()$ to $+_1$.)

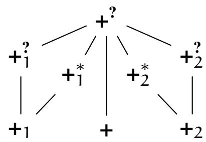
This property reinforces the intuition that $+^?$ should be at the top: this is what lets us substitute $+^?$ for more-precise sums. Dually, the static sums should be at the bottom: replacing a sum with a static sum should not, in general, preserve typing.

With this property in mind, how precise are $+^?_i$ and $+^*_i$, which we put in to reduce the number of typing rules? It doesn't make sense to "mix subscripts": moving between $+_2$ and $+^?_1$ in an annotation, or between $+_1$ to $+^*_2$, never preserves typing. Types with 1 subscripts should stay on the left of the edge from $+$ to $+^?$, and 2 subscripts should stay on the right.

Hence, we will place $+^?_1$ and $+^*_1$ left of the vertical edge (from $+$ to $+^?$), and $+^?_2$ and $+^*_2$ right of the vertical edge.

Moving to a less precise type should not lose inhabitants, because the lost inhabitants will become ill-typed. Suppose we put $+^*_1$ below $+^?_1$, making $+^*_1$ more precise. The sum $+^*_1$ contains both left and right injections (by the above subtyping relation, $+^?_2 \leq +^*_1$), meaning that $+^*_1$ has *more* inhabitants than $+^?_1$. Therefore, we should not have $+^*_1 \sqsubseteq +^?_1$.

The reverse, where $+^?_1 \sqsubseteq +^*_1$, is more plausible but would have unfortunate consequences (discussed at the end of this section). So we have no edge between $+^?_1$ and $+^*_1$.



Lifting this relation $\sqsubseteq$ on sum constructors to sum *types* is straightforward: if $\delta' \sqsubseteq \delta$ then $(A'_1\, \delta'\, A'_2) \sqsubseteq (A_1\, \delta\, A_2)$, provided $A'_1 \sqsubseteq A_1$ and $A'_2 \sqsubseteq A_2$. For function types, we diverge from subtyping: precision is covariant in the codomain *and* in the domain. This is consistent with precision in gradual typing, e.g. Siek and Vachharajani (2008) and Garcia et al. (2016), and with the refinement relations of Freeman (1994, p. 31) and Davies (2005).

Can we use this relation to type the above example $g\ y$, where we want to pass a value of type $+$ to a function expecting something of $+^?$ type? Subtyping is internalized through a subsumption rule (the rule on the left); we extend the rule to allow *loss of precision*: in addition to moving from $A$ to a supertype $B$, we can move from $B$ to a less-precise $B'$.

$$\frac{\Gamma \vdash e : A \quad A \leq B}{\Gamma \vdash e : B}\,\text{sub.} \qquad \frac{\Gamma \vdash e : A \quad A \leq B \quad B \sqsubseteq B'}{\Gamma \vdash e : B'}\,\text{sub.+loss}$$

Imprecision is fundamentally unsound: Using $B \sqsubseteq B'$, we move from a precise type (containing, say, $+$ and $+_2$) to an imprecise type containing $+^?$. Above, we showed that $+^?$ does not satisfy local soundness. The purpose of the $B \sqsubseteq B'$ premise is to allow more-precisely-typed code to interface with less-precisely-typed code. However, a type checker that lost precision wherever possible would behave like a type checker for a system that only had $+^?$.

In addition to losing precision after subtyping, we allow *gaining* precision *before* subtyping:

$$\frac{\Gamma \vdash e : A' \quad A \sqsubseteq A' \quad A \leq B \quad B \sqsubseteq B'}{\Gamma \vdash e : B'}\,\text{gain+sub.+loss}$$

Gaining precision is clearly unsound: $A \sqsubseteq A'$ allows moving from $+^?$ to $+_1$ or $+_2$. While unsound, this is needed for the property of varying precision: the typing of a single part of a program can become more or less precise, independent of the typing of the rest of the program.

We compose the three premises—gaining precision $A \sqsubseteq A'$, subtyping $A \leq B$, and losing precision $B \sqsubseteq B'$—into a relation $A' \rightsquigarrow B'$, called *directed consistency*.

With this relation, allowing $+^?_1 \sqsubseteq +^*_1$ would nearly erase the distinction between $+^*_1$ and $+^*_2$: first, $+^?_1 \sqsubseteq +^*_1$; second, $+^?_1 \leq +^*_2$; third, $+^*_2 \sqsubseteq +^*_2$. (An earlier version of our system did allow $+^?_1 \sqsubseteq +^*_1$—see Appendix C.)

Ideally, we should apply imprecision only when the programmer intends it. This goal motivates the bidirectional system in Section 4.

# 3. Source Type System

| | | |
|---|---|---|
| | $i$ ::= | $1 \mid 2$ |
| Source sums | $\delta$ ::= | $+ \mid +_i \mid +^? \mid +^?_i \mid +^*_i$ |
| Source expressions | $e$ ::= | $() \mid x \mid \lambda x.\, e \mid e_1\, e_2 \mid (e :: A)$ |
| | | $\mid \mathtt{inj}_i\, e$ |
| | | $\mid \mathsf{case}(e, \mathtt{inj}_1\, x_1.e_1, \mathtt{inj}_2\, x_2.e_2)$ |
| | | $\mid \mathsf{case}(e, \mathtt{inj}_i\, x.e_i)$ |
| Source types | $A, B$ ::= | $\mathsf{Unit} \mid A\, \delta\, B \mid A \to B$ |
| Source typing contexts | $\Gamma$ ::= | $\cdot \mid \Gamma, x : A$ |

**Figure 2.** Source syntax

The syntax of the source language is in Figure 2. Here, and throughout the paper, $i$ ranges over 1 and 2. The symbol $\delta$ ranges over the sum constructors: $+$ is the standard (static) sum, $+_1$ and $+_2$ are subscript sums denoting the $i$th injections, and $+^?$ is the gradual or dynamic sum. The final sum constructors, $+^?_i$ and $+^*_i$, are motivated by the desire to have the smallest number of introduction and elimination rules, as described in Section 2.

Source expressions are the unit $()$, variables $x$, abstraction $\lambda x.\, e$ and application $e_1\, e_2$, sum injection $\mathtt{inj}_i\, e$, annotation (or ascription) $(e :: A)$, a two-armed $\mathsf{case}$ that eliminates $+$, and a one-armed $\mathsf{case}$ that eliminates $+^*_i$.

Types $A$ and $B$ are $\mathsf{Unit}$, sums $A\, \delta\, B$, and functions $A \to B$. Typing contexts $\Gamma$ are unordered sets of typings $x : A$, where the $x$ are assumed to be distinct.

### 3.1 Subtyping and Precision

Figure 3 gives the rules for a *subsum* judgment on sum constructors, written $\delta' \leq \delta$. These rules follow the diagram in Section 2. The subtyping rule for sum types uses the subsum judgment. As is standard, the subtyping rule for functions is contravariant in the domain ($A_1 \leq A'_1$) and covariant in the codomain ($A'_2 \leq A_2$).

Precision on sum constructors (top of Figure 4) corresponds to the diagram from Section 2. On function types, precision is covariant in the domain, as discussed above.

In both subtyping and precision (for types), reflexivity and transitivity are admissible rules. Including transitivity rules would be fine on paper, but hard to implement since the middle type must be guessed. (The relations on sum constructors are a small finite set, so we do include transitivity rules; for an implementation, we would take the transitive closure.)

Subtyping and precision compose to form the directed consistency relation, which has a single rule, DirConsU, in Figure 5. The "U" in the name comes from the depiction to the right of the rule. Since precision is reflexive, DirConsU includes all pairs of types that are related by subtyping.

### 3.2 Typing Rules

Typing rules for the source language are shown in Figure 6. The rule for variables, SVar, is standard. Rules SAnno and SUnitIntro are standard, as are the rules S→Intro and S→Elim for functions.

Rule SCSub is a *consistent subsumption* rule: if $e$ has type $A'$ and $A'$ is directed consistent (Figure 5) with $A$, then $e$ has type $A$.

The rules for sums (SSumIntro, SSumElim1, SSumElim2) were developed in Section 2.1.

# 4. Bidirectional Source Typing

***Motivation.*** The type assignment system of Section 3 includes all the sensible sum types, along with subtyping and precision. By itself, the consistent subsumption rule SCSub makes type inference, and even type-checking, nontrivial: we should apply SCSub only

$\boxed{\delta' \leq \delta}$ Sum $\delta'$ is a subsum of $\delta$

$$\frac{}{\delta \leq \delta} \qquad \frac{}{+^?_i \leq +^?} \qquad \frac{}{+^? \leq +^*_i} \qquad \frac{}{+^?_i \leq +_i}$$

$$\frac{}{+_i \leq +^*_i} \qquad \frac{}{+^*_i \leq +} \qquad \frac{\delta' \leq \delta_1 \qquad \delta_1 \leq \delta}{\delta' \leq \delta}$$

$\boxed{A' \leq A}$ Type $A'$ is a subtype of $A$

$$\frac{}{\mathsf{Unit} \leq \mathsf{Unit}} \qquad \frac{A_1' \leq A_1 \qquad A_2' \leq A_2 \qquad \delta' \leq \delta}{(A_1' \,\delta'\, A_2') \leq (A_1 \,\delta\, A_2)}$$

$$\frac{A_1 \leq A_1' \qquad A_2' \leq A_2}{(A_1' \rightarrow A_2') \leq (A_1 \rightarrow A_2)}$$

**Figure 3.** Source subtyping

$\boxed{\delta' \sqsubseteq \delta}$ Sum $\delta'$ is more precise than $\delta$

$$\frac{}{\delta \sqsubseteq \delta} \qquad \frac{}{+_i \sqsubseteq +^?_i} \qquad \frac{}{+_i \sqsubseteq +^*_i} \qquad \frac{}{+^*_i \sqsubseteq +^?} \qquad \frac{}{+^?_i \sqsubseteq +^?} \qquad \frac{}{+ \sqsubseteq +^?}$$

$$\frac{\delta' \sqsubseteq \delta_1 \qquad \delta_1 \sqsubseteq \delta}{\delta' \sqsubseteq \delta}$$

$\boxed{A' \sqsubseteq A}$ Type $A'$ is more precise than $A$

$$\frac{}{\mathsf{Unit} \sqsubseteq \mathsf{Unit}} \qquad \frac{A_1' \sqsubseteq A_1 \qquad A_2' \sqsubseteq A_2 \qquad \delta' \sqsubseteq \delta}{(A_1' \,\delta'\, A_2') \sqsubseteq (A_1 \,\delta\, A_2)}$$

$$\frac{A_1' \sqsubseteq A_1 \qquad A_2' \sqsubseteq A_2}{(A_1' \rightarrow A_2') \sqsubseteq (A_1 \rightarrow A_2)}$$

**Figure 4.** Precision

$\boxed{A' \rightsquigarrow B'}$ Type $A'$ is directed consistent with $B'$

$$\frac{A \sqsubseteq A' \qquad A \leq B \qquad B \sqsubseteq B'}{A' \rightsquigarrow B'}\ \mathsf{DirConsU} \qquad \begin{array}{ccc} A' & & B' \\ \sqcup\!\!| & & \sqcup\!\!| \\ A & \leq & B \end{array}$$

**Figure 5.** Directed consistency

where necessary. This problem arises even with ordinary subsumption (subtyping, without changes of precision), which "forgets" that $e$ has a smaller type. Allowing changes of precision makes the problem worse: loss of precision "forgets" that $e$ has a more precise type, while gain of precision may add a downcast that fails at run time.

Such algorithmic difficulties could, perhaps, be resolved through careful design; the real problem with the type assignment system is that it types too many programs. Since SCSub is always applicable, any expression meant to be typed using only $+$ could be typed using $+^?$ instead.

A related problem is that our elimination rules for sums, while elegant, are excessively permissive: since $+^?_2$ is a subtype of $+^*_1$, an expression of type $+^?_2$ can be eliminated with a left-arm case—even though such an elimination is *guaranteed* to cause a match failure at run time. Since this is a consequence of the subtyping part of SCSub, it wouldn't help to remove the changes of precision from directed consistency.

$\boxed{\Gamma \vdash e : A}$ Under typing context $\Gamma$, expression $e$ has type $A$

$$\frac{\Gamma(x) = A}{\Gamma \vdash x : A}\ \mathsf{SVar} \qquad \frac{\Gamma \vdash e : A' \qquad A' \rightsquigarrow A}{\Gamma \vdash e : A}\ \mathsf{SCSub}$$

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash (e :: A) : A}\ \mathsf{SAnno} \qquad \frac{}{\Gamma \vdash () : \mathsf{Unit}}\ \mathsf{SUnitIntro}$$

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash (\lambda x.\, e) : (A \rightarrow B)}\ \mathsf{S{\rightarrow}Intro} \qquad \frac{\begin{array}{c}\Gamma \vdash e_1 : A \rightarrow B \\ \Gamma \vdash e_2 : A\end{array}}{\Gamma \vdash (e_1\, e_2) : B}\ \mathsf{S{\rightarrow}Elim}$$

$$\frac{\Gamma \vdash e : A_i}{\Gamma \vdash (\mathtt{inj}_i\, e) : (A_1 +^?_i A_2)}\ \mathsf{SSumIntro}$$

$$\frac{\Gamma \vdash e_0 : A_1 +^*_i A_2 \qquad \Gamma, x : A_i \vdash e : A}{\Gamma \vdash \mathsf{case}(e_0, \mathtt{inj}_i\, x.e) : A}\ \mathsf{SSumElim1}$$

$$\frac{\Gamma \vdash e_0 : A_1 + A_2 \qquad \begin{array}{c}\Gamma, x_1 : A_1 \vdash e_1 : A \\ \Gamma, x_2 : A_2 \vdash e_2 : A\end{array}}{\Gamma \vdash \mathsf{case}(e_0, \mathtt{inj}_1\, x_1.e_1, \mathtt{inj}_2\, x_2.e_2) : A}\ \mathsf{SSumElim2}$$

**Figure 6.** Source typing

We solve all of these problems via a bidirectional version of the system. In many settings, bidirectional typing has been chosen to overcome fundamental limitations of type inference, such as undecidability of inference for object-oriented subtyping (Pierce and Turner 1998), dependent types (Xi and Pfenning 1999; Pientka and Dunfield 2010) and first-class polymorphism (Dunfield and Krishnaswami 2013). It can also be motivated by better localization of type error messages. Our motivation is different: we want to stop the type-checker from doing certain things *unless* the programmer has signalled that they really want to do those things. Programmers signal their intent through type annotations, which are propagated through the bidirectional typing rules.

In Section 4.3, we show that the bidirectional system is sound and complete (under annotation) with respect to the type assignment system of Section 3.

***Checking and synthesis.*** Bidirectional typing splits typing into two judgments. The checking judgment $\Gamma \vdash e \Leftarrow A$ is read "$e$ checks against type $A$"; the synthesis judgment $\Gamma \vdash e \Rightarrow A$ is read "$e$ synthesizes type $A$". Both judgments can be interpreted as saying that $e$ has type $A$; the difference is that in checking, the type $A$ is already known, while synthesis infers $A$ from the available information ($\Gamma$ and $e$). The type in the checking judgment "flows" from some type annotation, either directly or (usually) indirectly.

An important advantage of the bidirectional system is a kind of subformula property (Gentzen 1934; Prawitz 1965). In our case, this property says that in a derivation of $\Gamma \vdash e \Rightarrow A$, every type synthesized or checked against is derived from types found in $\Gamma$ and $e$. For $\Gamma \vdash e \Leftarrow A$, every such type is derived from $\Gamma$, $e$, and $A$. Consequently, dynamic sums cannot appear out of nowhere: they result only from type annotations. We exploit this property in, for example, the proof of Theorem 5.

***From type assignment rules to bidirectional rules.*** As is often the case with bidirectional type systems, our bidirectional rules will strongly resemble our type assignment rules. In general, we construct a bidirectional rule by replacing ":" with "$\Leftarrow$" or "$\Rightarrow$". The main question is when to use checking, and when to use synthesis. Checking is more powerful than synthesis; for a premise, we generally prefer to make it a checking judgment, but a checking *conclusion* may increase the number of required type annotations.

$$\boxed{\begin{array}{l}\Gamma \vdash e \Leftarrow A \\ \Gamma \vdash e \Rightarrow A\end{array}}\quad \begin{array}{l}\text{Under context } \Gamma\text{, expr. } e \text{ checks against type } A \\ \text{Under context } \Gamma\text{, expr. } e \text{ synthesizes type } A\end{array}$$

$$\frac{\Gamma(x) = A}{\Gamma \vdash x \Rightarrow A}\ \mathsf{SynVar} \qquad \frac{\Gamma \vdash e \Rightarrow A' \qquad A' \rightsquigarrow A}{\Gamma \vdash e \Leftarrow A}\ \mathsf{ChkCSub}$$

$$\frac{\Gamma \vdash e \Leftarrow A}{\Gamma \vdash (e :: A) \Rightarrow A}\ \mathsf{SynAnno} \qquad \frac{}{\Gamma \vdash () \Leftarrow \mathsf{Unit}}\ \mathsf{ChkUnitIntro}$$

$$\frac{\Gamma, x : A \vdash e \Leftarrow B}{\Gamma \vdash (\lambda x.\, e) \Leftarrow (A \to B)}\ \mathsf{Chk{\to}Intro}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow (A \to B) \qquad \Gamma \vdash e_2 \Leftarrow A}{\Gamma \vdash (e_1\, e_2) \Rightarrow B}\ \mathsf{Syn{\to}Elim}$$

$$\frac{\Gamma \vdash e \Leftarrow A_i \qquad +^?_i \le \delta}{\Gamma \vdash (\mathtt{inj}_i\, e) \Leftarrow (A_1\, \delta\, A_2)}\ \mathsf{ChkSumIntro}$$

$$\frac{\begin{array}{ll}\Gamma \vdash e_0 \Rightarrow (A_1\, \delta\, A_2) & \\ \delta \Rrightarrow +^*_i & \Gamma, x : A_i \vdash e \Leftarrow A\end{array}}{\Gamma \vdash \mathsf{case}(e_0, \mathtt{inj}_i\, x.e) \Leftarrow A}\ \mathsf{ChkSumElim1}$$

$$\frac{\begin{array}{ll}\Gamma \vdash e_0 \Rightarrow (A_1\, \delta\, A_2) & \Gamma, x_1 : A_1 \vdash e_1 \Leftarrow A \\ \delta \Rrightarrow + & \Gamma, x_2 : A_2 \vdash e_2 \Leftarrow A\end{array}}{\Gamma \vdash \mathsf{case}(e_0, \mathtt{inj}_1\, x_1.e_1, \mathtt{inj}_2\, x_2.e_2) \Leftarrow A}\ \mathsf{ChkSumElim2}$$

$$\boxed{\delta \Rrightarrow \delta'}\quad \text{Sum } \delta \text{ synthesizes sum } \delta'$$

$$\frac{}{+^?_i \Rrightarrow +^*_i} \qquad \frac{}{+_i \Rrightarrow +^*_i} \qquad \frac{}{+^? \Rrightarrow +^*_i} \qquad \frac{}{+^*_i \Rrightarrow +^*_i} \qquad \frac{}{\delta \Rrightarrow +}$$

**Figure 7.** Bidirectional typing (source)

For the most part, we follow the recipe of Davies and Pfenning (2000); Dunfield and Pfenning (2004): introduction rules check, and elimination rules synthesize. More precisely, the judgment that includes the relevant connective—the *principal judgment*—should check for an introduction rule, and synthesize for an elimination rule.

Doing this step naturally determines the directions of many other judgments. For example, in rule Syn→Elim, the principal judgment is the first premise $\Gamma \vdash e_1 \Rightarrow (A_1 \to A_2)$. Since the type in a synthesis judgment is output, deriving this premise tells us what $A_1$ is, enabling us to make the second premise a checking judgment. The premise also tells us what $A_2$ is—so we can make the conclusion a synthesis judgment. Consequently, applications $e_1\, e_2$ will synthesize a type, without any local annotation, whenever the function $e_1$ synthesizes. In rule Chk→Intro, not following the recipe—by making the conclusion synthesize, $\Gamma \vdash \lambda x.\, e \Rightarrow (A_1 \to A_2)$—means that we don't know $A_1$, and cannot construct the context $\Gamma, x : A_1$ in the premise. (It may be possible to design a more complicated system in which $\lambda x.\, e$ *does* synthesize, as Dunfield and Krishnaswami (2013) did for a different type system.)

Rule ChkSumIntro says that $\mathtt{inj}_1\, e$ checks against $A_1\, \delta\, A_2$, where $\delta$ is any sum above $+^?_1$—that is, any sum constructor *except* $+^?_2$ and $+_2$. This is a checking rule for two reasons. First, it is an introduction form, so according to the recipe its principal judgment (the conclusion) should check. Second, the simplest synthesizing rule would synthesize $A_1 +^?_i A_2$. But that is a subtype of $A_1 +^? A_2$, introducing a possibly undesired dynamic sum.

In the (one-armed) elimination rule SSumElim1, the principal judgment is the premise $\Gamma \vdash e_0 : A_1 +^*_i A_2$. Following the recipe, the corresponding premise of ChkSumElim1 synthesizes. It would be unfortunate to require it to synthesize *exactly* $A_1 +^*_i A_2$: assuming programmers mostly write type annotations using $+_1$, $+_2$, $+$ and $+^?$, virtually no expressions will synthesize $+^*_i$. On the other hand, checking $e_0$ against $A_1 +^*_i A_2$ would be too permissive: if we have a left one-armed case $\mathsf{case}(e_0, \mathtt{inj}_1\, x.e)$, we would accept $e_0$ of type $+^?_2$, even though $+^?_2$ is a *right* injection, guaranteeing a run-time failure. Instead, we require that $e_0$ synthesize $A_1\, \delta\, A_2$ where $\delta \Rrightarrow +^*_i$. The judgment $\delta \Rrightarrow +^*_1$ is derivable when $\delta$ is $+^?_1$, $+_1$, $+^?$ or $+^*_1$.

For consistency with ChkSumElim1, our two-armed elimination rule ChkSumElim2 has a similar structure (with an additional premise for the second arm) and also uses the $\Rrightarrow$ judgment; however, $\delta \Rrightarrow +$ is *always* derivable, because a two-armed case is safe for every sum constructor. We include this premise anyway, to highlight the two rules' similarity.

Several rules are not tied to specific type connectives. An assumption $x : A$ in $\Gamma$ could be read "$x$ synthesizes $A$", so SynVar synthesizes its type. Rule SynAnno synthesizes the type given in an annotation $(e :: A)$, provided $e$ checks against $A$. Following earlier bidirectional systems (Davies and Pfenning 2000; Dunfield and Pfenning 2004), the subsumption rule has a checking conclusion and a synthesizing premise. The checking conclusion ensures that subsumption, which loses information, is applied only with the programmer's consent: the type being checked against is derived from a type annotation. The synthesizing premise ensures that we "make progress" as we move from the goal $e \Leftarrow A$ to the subgoal $e \Rightarrow A'$: we cannot use ChkCSub as the concluding rule of its own premise. In addition to subtyping and change of precision, ChkCSub with $A = A'$ (using reflexivity) allows us to use a derivation of $\Gamma \vdash e \Rightarrow A$ where we need a derivation of $\Gamma \vdash e \Leftarrow A$. For example, applying a function to a variable requires this rule: SynVar synthesizes, but Syn→Elim has a checking premise.

***Complexity.*** Typing in the bidirectional system takes polynomial time. With one exception, the bidirectional rules are in one-to-one correspondence with syntactic forms. The exception is ChkCSub, which can be used to check any synthesizing form. So bidirectional typing is syntax-directed in a slightly looser sense than the usual one: For each pair of a syntactic form and a direction (checking or synthesis), exactly one rule applies; if that rule is ChkCSub, then exactly one rule applies to derive its synthesizing premise. Thus, the size of a derivation (if one exists) is, at most, twice the size of the expression.

***Variations on a theme.*** Several checking rules could be supplemented with a synthesizing rule, or (in the case of ChkUnitIntro) replaced. A synthesizing version of ChkSumIntro, however, would be problematic: while we might synthesize the sum constructor $+_i$, synthesizing $e$ for $A_i$ tells us only one component of the sum. Our system enjoys uniqueness of synthesis: given $\Gamma$ and $e$, $e$ synthesizes (at most) one type. Synthesizing the other component of the sum would synthesize an infinite number of types. Moreover, a direct implementation would need to guess the other component.

A synthesizing version of ChkSumElim1 would be straightforward; for ChkSumElim2, we could synthesize $e_1 \Rightarrow B_1$ and $e_2 \Rightarrow B_2$ and synthesize their join $B_1 \vee B_2$ in the conclusion.

Except for ChkUnitIntro, all of these variations—while perhaps convenient in practice—would make the system larger and more complicated. This paper presents a core calculus; we leave exploration of such variations to future work.

### 4.1 Static System

Two restricted versions of the bidirectional system are of interest. The first is a *static* system: a simply typed $\lambda$-calculus with sums and refinements over sums, without any dynamic sums. The syntax (Figure 8) is the same as the source language, except for $\delta^{\mathsf{S}}$ which

Static sums $\delta^S ::= + \mid +_i$

Static expressions $e^S ::= () \mid x \mid \lambda x.\, e^S \mid e_1^S\, e_2^S \mid \mathsf{inj}_i\, e^S \mid (e^S :: A^S)$
$\quad \mid \mathsf{case}(e^S, \mathsf{inj}_1\, x_1.e_1^S, \mathsf{inj}_2\, x_2.e_2^S) \mid \mathsf{case}(e^S, \mathsf{inj}_i\, x.e_i^S)$

Static types $A^S ::= \mathsf{Unit} \mid A_1^S\, \delta^S\, A_2^S \mid A_1^S \to A_2^S$

Static typing contexts $\Gamma^S ::= \cdot \mid \Gamma^S, x : A^S$

$\boxed{\delta_1^S \le_S \delta_2^S}$ Static sum $\delta_1^S$ is a subsum of $\delta_2^S$

$\boxed{A_1^S \le_S A_2^S}$ Static type $A_1^S$ is a subtype of $A_2^S$

$$\frac{}{\delta^S \le_S \delta^S} \qquad \frac{}{+_i \le_S +} \qquad \frac{}{\mathsf{Unit} \le_S \mathsf{Unit}} \qquad \frac{A_{11}^S \le_S A_{12}^S \quad A_{21}^S \le_S A_{22}^S \quad \delta_1^S \le_S \delta_2^S}{(A_{11}^S\, \delta_1^S\, A_{21}^S) \le_S (A_{12}^S\, \delta_2^S\, A_{22}^S)} \qquad \frac{A_{12}^S \le_S A_{11}^S \quad A_{21}^S \le_S A_{22}^S}{(A_{11}^S \to A_{21}^S) \le_S (A_{12}^S \to A_{22}^S)}$$

$\boxed{\begin{array}{l}\Gamma^S \vdash_S e^S \Leftarrow A^S \\ \Gamma^S \vdash_S e^S \Rightarrow A^S\end{array}}$ Under typing context $\Gamma^S$, expression $e^S$ checks against type $A^S$
Under typing context $\Gamma^S$, expression $e^S$ synthesizes type $A^S$

$$\frac{\Gamma^S(x) = A^S}{\Gamma^S \vdash_S x \Rightarrow A^S}\,\mathsf{StVar} \qquad \frac{\Gamma^S \vdash_S e^S \Rightarrow A_0^S \quad A_0^S \le_S A^S}{\Gamma^S \vdash_S e^S \Leftarrow A^S}\,\mathsf{StSub} \qquad \frac{\Gamma^S \vdash_S e^S \Leftarrow A^S}{\Gamma^S \vdash_S (e^S :: A^S) \Rightarrow A^S}\,\mathsf{StAnno} \qquad \frac{}{\Gamma^S \vdash_S () \Leftarrow \mathsf{Unit}}\,\mathsf{StUnitIntro}$$

$$\frac{\Gamma^S, x : A_1^S \vdash_S e^S \Leftarrow A_2^S}{\Gamma^S \vdash_S \lambda x.\, e^S \Leftarrow A_1^S \to A_2^S}\,\mathsf{St{\to}Intro} \qquad \frac{\Gamma^S \vdash_S e_1^S \Rightarrow A_1^S \to A_2^S \quad \Gamma^S \vdash_S e_2^S \Leftarrow A_1^S}{\Gamma^S \vdash_S e_1^S\, e_2^S \Rightarrow A_2^S}\,\mathsf{St{\to}Elim} \qquad \frac{\Gamma^S \vdash_S e^S \Leftarrow A_i^S \quad +_i \le_S \delta^S}{\Gamma^S \vdash_S \mathsf{inj}_i\, e^S \Leftarrow (A_1^S\, \delta^S\, A_2^S)}\,\mathsf{StSumIntro}$$

$$\frac{\Gamma^S \vdash_S e_0^S \Rightarrow A_1^S +_i A_2^S \quad \Gamma^S, x : A_i^S \vdash_S e^S \Leftarrow A^S}{\Gamma^S \vdash_S \mathsf{case}(e_0^S, \mathsf{inj}_i\, x.e^S) \Leftarrow A^S}\,\mathsf{StSumElim1} \qquad \frac{\begin{array}{ll}\Gamma^S \vdash_S e_0^S \Rightarrow A_1^S\, \delta^S\, A_2^S & \Gamma^S, x_1 : A_1^S \vdash_S e_1^S \Leftarrow A^S \\ \delta^S \le_S + & \Gamma^S, x_2 : A_2^S \vdash_S e_2^S \Leftarrow A^S\end{array}}{\Gamma^S \vdash_S \mathsf{case}(e_0^S, \mathsf{inj}_1\, x_1.e_1^S, \mathsf{inj}_2\, x_2.e_2^S) \Leftarrow A^S}\,\mathsf{StSumElim2}$$

**Figure 8.** The static system: the bidirectional system restricted to $+$, $+_1$, $+_2$

can only be $+$, $+_1$, or $+_2$. We follow the bidirectional system in deriving rules for sub-sum, subtyping, and typing; the judgments are decorated with S for "static". The interesting difference is in the typing rules for sums: the introduction rule checks that the sum is above $+_i$ (instead of $+_i^?$), and the one-arm elimination StSumElim1 checks that the sum is below $+_i$ (instead of $+_i^*$), that is, the sum is exactly $+_i$.

### 4.2 Dynamic System

The static system omits dynamic sums; the dynamic system's only sum is the dynamic sum $+^?$. Since one-armed cases are allowed on type $+^?$, this corresponds to datatypes in Standard ML. The meta-variables and judgments are decorated with D for "dynamic". For space reasons, the definition of this system is in the supplementary material (Appendix A).

### 4.3 Metatheory

The bidirectional system is decidable. The $\delta' \le \delta$ judgment is immediately decidable (taking the transitive closure of the rules), and the $A' \le A$ judgment is decidable because each rule moves from larger type expressions to smaller ones. The same holds for $\sqsubseteq$, so directed consistency is decidable. The argument for the typing rules is slightly more interesting, as ChkCSub is a *stationary* rule (the premise and conclusion type the same expression). However, since this rule moves from checking to synthesis, and no stationary rule moves from synthesis to checking (in SynAnno, the expression becomes smaller), decidability holds.

**Theorem 1** (Decidability of bidirectional typing)**.**

1. *Given $\Gamma$, $e$ and $A$, the judgment $\Gamma \vdash e \Leftarrow A$ is decidable.*
2. *Given $\Gamma$ and $e$, the judgment $\Gamma \vdash e \Rightarrow A$ is decidable.*

The bidirectional system is sound with respect to the type assignment system: if $e$ is well-typed in the bidirectional system, it is well-typed in the type assignment system. (Proofs can be found in the supplementary material.)

**Theorem 2** (Bidirectional soundness)**.**
*If $\Gamma \vdash e \Leftarrow A$ or $\Gamma \vdash e \Rightarrow A$ then $\Gamma \vdash e : A$.*

The bidirectional system is also complete: given $e : A$ in the type assignment system, it is always possible to add annotations that make $e$ well-typed in the bidirectional system. We write $e =: e'$ when $e'$ is the same as $e$ except that $e'$ may have extra annotations.

**Theorem 3** (Annotatability)**.**
*If $\Gamma \vdash e : A$ then there exist $e'$ and $e''$ such that (1) $\Gamma \vdash e' \Leftarrow A$ where $e =: e'$, and (2) $\Gamma \vdash e'' \Rightarrow A$ where $e =: e''$.*

We also show that bidirectional typing derivations are robust under imprecision: if $e' \Leftarrow A'$, replacing annotations in $e'$ with more imprecise types preserves typing. This corresponds to part 1 of the *gradual guarantee* of Siek et al. (2015, Theorem 5 on p. 11). An example illustrating this theorem's significance appears below in Section 4.4.

First, $\Gamma' \sqsubseteq \Gamma$ is defined pointwise. Second, let $e' \sqsubseteq e$ if, for each annotation $(e_0' :: A')$ in $e'$, there is a corresponding annotation $(e_0 :: A)$ in $e$ where $A' \sqsubseteq A$. (For full inductive definitions, see Figures 15 and 16 in the supplementary material.)

**Theorem 4** (Varying precision of bidirectional typing)**.**

1. *If $\Gamma' \vdash e' \Leftarrow A'$ and $e' \sqsubseteq e$ and $\Gamma' \sqsubseteq \Gamma$ and $A' \sqsubseteq A$ then $\Gamma \vdash e \Leftarrow A$.*
2. *If $\Gamma' \vdash e' \Rightarrow A'$ and $e' \sqsubseteq e$ and $\Gamma' \sqsubseteq \Gamma$ then there exists $A$ such that $\Gamma \vdash e \Rightarrow A$ and $A' \sqsubseteq A$.*

The nonempty context is needed for the proof cases for rules whose premises add to $\Gamma'$, such as ChkSumElim1.

An earlier version of the system, which did not allow gain of precision, has a weaker property: in that system, the given expression $e$ is not necessarily typable, but there exists some "even more imprecise" expression $e_j$ that is typable. See Theorem 14 in Appendix C.

***Static system.*** As the static system is essentially a restriction of the bidirectional system, it is easy to turn a derivation in the static system into a derivation in the bidirectional system; this is the first part of the following theorem.

Completeness is more interesting: Given a bidirectional derivation whose *conclusion* is static—that is, the context $\Gamma$, expression $e$, and type $A$ are within the restricted static grammar—we can build a derivation in the static system. This holds because of a subformula property: if there are no dynamic sums in $\Gamma$, $e$ and $A$, then dynamic sums cannot appear anywhere in the bidirectional derivation.

**Theorem 5** (Static soundness and completeness)**.**

1. *Soundness:*
   (a) *If* $\Gamma^S \vdash_S e^S \Leftarrow A^S$ *then* $\Gamma^S \vdash e^S \Leftarrow A^S$
   (b) *If* $\Gamma^S \vdash_S e^S \Rightarrow A^S$ *then* $\Gamma^S \vdash e^S \Rightarrow A^S$.
2. *Completeness:*
   (a) *If* $\Gamma^S \vdash e^S \Leftarrow A^S$ *then* $\Gamma^S \vdash_S e^S \Leftarrow A^S$.
   (b) *If* $\Gamma^S \vdash e^S \Rightarrow A^S$ *then* $\Gamma^S \vdash_S e^S \Rightarrow A^S$.

This theorem directly corresponds to part 1 of Theorem 1 of Siek et al. (2015, p. 9) for "fully annotated" expressions. In that work, an expression is fully annotated if it has no gradual type annotations. In our system, expressions without annotations are static.

A corresponding theorem holds for the dynamic system and, in turn, corresponds to part 1 of Theorem 2 of Siek et al. (2015, p. 9). This is a rough correspondence: in our bidirectional system, dynamism is restricted to sum types and arises only through annotations. See Theorem 15 in the appendix.

### 4.4 Example

To see why Theorem 4 matters, consider the following example. Suppose we want to transform a program that uses dynamic sums into one that uses static sums. The program has a function $f$ of type $(\mathsf{Unit} +^? \mathsf{Int}) \to \mathsf{Int}$, which is called with an argument $x$ of type $\mathsf{Unit} +^? \mathsf{Int}$.

$$\begin{array}{l}\mathtt{let}\ f = (\lambda y.\ \cdots) :: (\mathsf{Unit} +^? \mathsf{Int}) \to \mathsf{Int}\ \mathtt{in}\ \ \ldots\\ \quad \mathtt{let}\ x = e_x :: (\mathsf{Unit} +^? \mathsf{Int})\ \mathtt{in}\\ \qquad f\ x\end{array}$$

(We assume that $e_x$ is a checking form that needs an annotation; if $e_x$ synthesizes $(\mathsf{Unit} +^? \mathsf{Int})$, the annotation could be removed.) The programmer realizes that $f$ only works with a right injection (perhaps its body is a one-armed case on $\mathtt{inj}_2$), and that $x$ should always be a right injection.

$$\begin{array}{l}\mathtt{let}\ f = (\lambda y.\ \cdots) :: (\mathsf{Unit} +_2 \mathsf{Int}) \to \mathsf{Int}\ \mathtt{in}\ \ \ldots\\ \quad \mathtt{let}\ x = e_x :: (\mathsf{Unit} +_2 \mathsf{Int})\ \mathtt{in}\\ \qquad f\ x\end{array}$$

If this program type-checks and contains no remaining dynamic sum annotations, we know that $f$ and $x$ actually satisfy their annotations, and that the application $f\ x$ will not cause any match or cast failures. Theorem 4 says that the annotations can be changed *one at a time*: the program with $+^?$ in the type of $f$ but $+_2$ in the type of $x$ is well-typed, as is the program with $+_2$ in the type of $f$ but $+^?$ in the type of $x$:

$$\begin{array}{l}\mathtt{let}\ f = (\lambda y.\ \cdots) :: (\mathsf{Unit} +_2 \mathsf{Int}) \to \mathsf{Int}\ \mathtt{in}\ \ \ldots\\ \quad \mathtt{let}\ x = e_x :: (\mathsf{Unit} +^? \mathsf{Int})\ \mathtt{in}\\ \qquad f\ x\end{array}$$

When synthesizing the type of $f\ x$, we use ChkCSub to gain precision in $x$:

$$\dfrac{\begin{array}{c}\Gamma \vdash f \Rightarrow\\ (\mathsf{Unit} +_2 \mathsf{Int}) \to \mathsf{Int}\end{array} \qquad \dfrac{\Gamma \vdash x \Rightarrow (\mathsf{Unit} +^? \mathsf{Int}) \qquad (\mathsf{Unit} +^? \mathsf{Int}) \rightsquigarrow (\mathsf{Unit} +_2 \mathsf{Int})}{\Gamma \vdash x \Leftarrow (\mathsf{Unit} +_2 \mathsf{Int})}\ \mathsf{ChkCSub}}{\Gamma \vdash f\ x \Rightarrow \mathsf{Int}}\ \mathsf{Syn}{\to}\mathsf{Elim}$$

A precise annotation that differs from the correct one, such as $\mathsf{Unit} +_1 \mathsf{Int}$ on $x$, may cause an error—either at type-checking time, or at run time. But a precise annotation that is correct will not cause an error, and constitutes a step towards a completely static program.

## 5. Target Language and Translation

### 5.1 Target Syntax and Semantics

| | | |
|---|---|---|
| | $i$ | $::= 1 \mid 2$ |
| Target sums | $\phi$ | $::= + \mid +_i$ |
| Target terms | $M$ | $::= () \mid x \mid \lambda x.\, M \mid M_1\, M_2 \mid \mathtt{inj}_i\, M$ |
| | | $\mid \mathtt{case}(M, \mathtt{inj}_1\, x_1.M_1, \mathtt{inj}_2\, x_2.M_2)$ |
| | | $\mid \mathtt{case}(M, \mathtt{inj}_i\, x.M_i)$ |
| | | $\mid \langle \phi_2 \Leftarrow \phi_1 \rangle M \mid \mathtt{matchfail}$ |
| Values | $W$ | $::= () \mid x \mid \lambda x.\, M \mid \mathtt{inj}_i\, W$ |
| Target types | $T$ | $::= \mathsf{Unit} \mid T_1\, \phi\, T_2 \mid T_1 \to T_2$ |
| Target typing contexts | $\Theta$ | $::= \cdot \mid \Theta, x : T$ |

**Figure 9.** Target syntax

Our target language is a statically typed $\lambda$-calculus with static sum types and a cast construct. The syntax is shown in Figure 9. We write $M$ for target terms (expressions), $W$ for values, and $T$ for target types. The target sum constructors are all the static sum types from the source language: $+$, $+_1$, and $+_2$. In addition, we have a cast construct $\langle \phi_2 \Leftarrow \phi_1 \rangle M$, which casts from sum $\phi_1$ to $\phi_2$. A failing cast, such as $\langle +_2 \Leftarrow + \rangle (\mathtt{inj}_1\ ())$, steps to the error term $\mathtt{matchfail}$.

Much of the target type system (Figure 10) follows the source type assignment system, if that system were restricted to static sum types. Since the target lacks any dynamic sum constructors (like $+^?$), target subtyping says only that $+_1$ and $+_2$ are subtypes of $+$; this corresponds to datasort refinement systems, where every datasort is a subsort of a "top" datasort for the type being refined. Our type-directed translation (Section 5.2) transforms the gradual property of types into dynamic checks at the term level; rule TCast casts between sum constructors, and rule TMatchfail gives any type to $\mathtt{matchfail}$, which represents the failure of a cast.

Our target language (Figure 11) has a standard call-by-value small-step semantics, extended with casts. Evaluation contexts $\mathcal{E}$ are terms with a hole $[]$, where the hole represents a term in an evaluation position: if target term $M = \mathcal{E}[M_0]$, and $M_0$ *reduces*—written $M_0 \mapsto_R M_0'$—then the larger term $M$ steps to $\mathcal{E}[M_0']$.

The cast reduction rules represent the three relevant situations: (1) an *upcast* to a supertype succeeds (ReduceUpcast); (2) a downcast from $+$ to $+_i$ succeeds if $i$ matches the injection (ReduceCastSuccess); (3) a downcast from $+$ to $+_i$ fails, reducing to $\mathtt{matchfail}$, if $i$ doesn't match the injection (ReduceCastFailure).

### 5.2 Type-Directed Translation $\hookrightarrow$

To translate source programs into target programs with explicit casts between sum types, we use a judgment $\Gamma \vdash e : A \hookrightarrow M$. Most of the rules (in Figure 12) follow the type assignment rules, with the addition of $\hookrightarrow M$. Given $e$ of type $A$, the rules produce a target term $M$ of type $T$ where $T$ is the translation of $A$, written $|A|$.

$\boxed{\phi' \le \phi}$ Sum $\phi'$ is a subsum of $\phi$

$$\frac{}{\phi \le \phi} \qquad \frac{}{+_i \le +}$$

$\boxed{T' \le T}$ Target type $T'$ is a subtype of $T$

$$\frac{}{\mathsf{Unit} \le \mathsf{Unit}} \qquad \frac{T_1' \le T_1 \quad T_2' \le T_2 \quad \phi' \le \phi}{(T_1'\,\phi'\,T_2') \le (T_1\,\phi\,T_2)} \qquad \frac{T_1 \le T_1' \quad T_2' \le T_2}{(T_1' \to T_2') \le (T_1 \to T_2)}$$

$\boxed{\Theta \vdash M : T}$ Under context $\Theta$, target term $M$ has target type $T$

$$\frac{\Theta(x) = T}{\Theta \vdash x : T}\ \mathsf{TVar} \qquad \frac{\Theta \vdash M : T' \quad T' \le T}{\Theta \vdash M : T}\ \mathsf{TSub} \qquad \frac{\Theta \vdash M : (T_1\,\phi'\,T_2)}{\Theta \vdash \langle \phi \Leftarrow \phi' \rangle M : (T_1\,\phi\,T_2)}\ \mathsf{TCast}$$

$$\frac{}{\Theta \vdash \mathtt{matchfail} : T}\ \mathsf{TMatchfail} \qquad \frac{}{\Theta \vdash () : \mathsf{Unit}}\ \mathsf{TUnitIntro} \qquad \frac{\Theta \vdash M : T_i}{\Theta \vdash \mathtt{inj}_i\, M : (T_1 +_i T_2)}\ \mathsf{T}{+_i}\mathsf{Intro}$$

$$\frac{\Theta \vdash M_0 : T_1 +_i T_2 \quad \Theta, x : T_i \vdash M : T}{\Theta \vdash \mathtt{case}(M_0, \mathtt{inj}_i\, x.M) : T}\ \mathsf{T}{+_i}\mathsf{Elim} \qquad \frac{\Theta \vdash M_0 : T_1 + T_2 \quad \begin{array}{c}\Theta, x_1 : T_1 \vdash M_1 : T \\ \Theta, x_2 : T_2 \vdash M_2 : T\end{array}}{\Theta \vdash \mathtt{case}(M_0, \mathtt{inj}_1\, x_1.M_1, \mathtt{inj}_2\, x_2.M_2) : T}\ \mathsf{T{+}Elim}$$

$$\frac{\Theta, x : T_1 \vdash M : T_2}{\Theta \vdash \lambda x.\, M : (T_1 \to T_2)}\ \mathsf{T{\to}Intro} \qquad \frac{\Theta \vdash M_1 : T' \to T \quad \Theta \vdash M_2 : T'}{\Theta \vdash M_1\, M_2 : T}\ \mathsf{T{\to}Elim}$$

**Figure 10.** Target subtyping and typing

Evaluation contexts

$$\begin{array}{rl}\mathcal{E} ::= & [\,] \\ & \mid \mathtt{inj}_i\, \mathcal{E} \\ & \mid \mathtt{case}(\mathcal{E}, \mathtt{inj}_i\, x.M) \\ & \mid \mathtt{case}(\mathcal{E}, \mathtt{inj}_1\, x_1.M_1, \mathtt{inj}_2\, x_2.M_2) \\ & \mid \langle \phi \Leftarrow \phi' \rangle \mathcal{E} \\ & \mid \mathcal{E}\, M_2 \mid W_1\, \mathcal{E}\end{array}$$

$\boxed{M \mapsto_R M'}$ Target term $M$ reduces to $M'$

$$\begin{array}{rll}\langle \phi \Leftarrow \phi' \rangle W \mapsto_R W & & \\ \text{where } \phi' \le \phi & & \mathsf{ReduceUpcast} \\ \langle +_i \Leftarrow + \rangle (\mathtt{inj}_i\, W) \mapsto_R \mathtt{inj}_i\, W & & \mathsf{ReduceCastSuccess} \\ \langle +_k \Leftarrow \phi' \rangle (\mathtt{inj}_i\, W) \mapsto_R \mathtt{matchfail} & & \\ \text{where } \phi' \in \{+_i, +\} \text{ and } i \ne k & & \mathsf{ReduceCastFailure} \\ \mathtt{case}(\mathtt{inj}_i\, W, \mathtt{inj}_i\, x.M) \mapsto_R [W/x]M & & \mathsf{ReduceCase1} \\ \mathtt{case}(\mathtt{inj}_i\, W, \mathtt{inj}_1\, x_1.M_1, \mathtt{inj}_2\, x_2.M_2) \mapsto_R [W/x_i]M_i & & \mathsf{ReduceCase2} \\ (\lambda x.\, M)\, W \mapsto_R [W/x]M & & \mathsf{Reduce}\beta\end{array}$$

$\boxed{M \mapsto M'}$ Target term $M$ steps to $M'$

$$\frac{M \mapsto_R M'}{\mathcal{E}[M] \mapsto \mathcal{E}[M']}\ \mathsf{StepContext} \qquad \frac{\mathcal{E} \ne [\,]}{\mathcal{E}[\mathtt{matchfail}] \mapsto \mathtt{matchfail}}\ \mathsf{StepMatchfail}$$

**Figure 11.** Small-step semantics of the target language

This translation (Figure 12, top) maps the source sums $+$ and $\overset{?}{+}$ to the target sum $+$, and maps the other source sums to $+_i$.

We extend type assignment, rather than the bidirectional system, because translation should be independent of bidirectionality: Type assignment is stable under variations in the bidirectional "recipe", so if we decided to synthesize a type for $()$, we could leave the translation untouched. That said, an implementation would be based on a bidirectional version of the translation—replacing ":" with "$\Leftarrow$" or "$\Rightarrow$", following Figure 7.

The interesting translation rule is STCSub, which inserts a *coercion context* $\mathcal{C}$. This context coerces between two directed-consistent types, so it composes up to three coercions (cf. Figure 5): from a more imprecise type to a less imprecise type, from that type to a supertype, and from the supertype to a more imprecise type.

Our coercion judgment $A' \Rightarrow A \hookrightarrow \mathcal{C}$ produces a context $\mathcal{C}$, a target term containing a hole such that, if $M$ has type $T' = |A'|$, then $\mathcal{C}[M]$ has type $T = |A|$. Rule CoeUnit produces a hole, which behaves as the identity function. Rule Coe$\to$ produces a function: given a hole $[\,]$ filled by a function of type $T_1' \to T_2'$, it constructs $\lambda x. \mathcal{C}_2\big[[\,]\ \mathcal{C}_1[x]\big]$. This function has type $T_1 \to T_2$: it applies cast $\mathcal{C}_1$ to $x$, yielding a value of type $T_1'$. Applying the original function yields an $T_2'$, which cast $\mathcal{C}_2$ transforms into an $T_2$.

Three rules generate coercions between sum types: CoeCase1L, CoeCase1R, and CoeCase2. The first two rules handle sums that are definitely a left injection, or definitely a right injection: we apply CoeCase1L whenever we are coercing from $A_1'\ \delta'\ A_2'$ where $\delta'$ is $+_1$ or $\overset{?}{+}_1$, and CoeCase1R when $\delta'$ is $+_2$ or $\overset{?}{+}_2$.

In CoeCase1L, we recursively generate a coercion $\mathcal{C}_1$ from $A_1'$, and a cast $\mathcal{C}_3$ from $\delta'$. The conclusion generates a coercion by matching the given value (replacing $[\,]$) against $\mathtt{inj}_1\, x_1$, constructing $\mathtt{inj}_1\, (\mathcal{C}_1[x_1])$, to which we apply $\mathcal{C}_3$. CoeCase1R is symmetric.

CoeCase2 handles the cases not covered by the previous two rules. In addition to doing the work of the previous two rules, it generates casts $\mathcal{C}_1'$ and $\mathcal{C}_2'$, applying them in each arm. According to STSumIntro, an injection $\mathtt{inj}_1$ has a type whose sum constructor is $\overset{?}{+}_1$, so CoeCase2 applies $\mathcal{C}_1'$ which takes $\overset{?}{+}_1$ to $\delta'$. Similarly, the rule applies $\mathcal{C}_2'$, which takes $\overset{?}{+}_2$ to $\delta'$. Since CoeCase2 applies $\mathcal{C}_3$ (from $\delta'$ to $\delta$) to the entire case, the result will be $\delta$.

### 5.3 Target Precision $\preccurlyeq$

We will prove that more precise source typings—differently annotated versions of the same source expression—produce more precise target terms. We will also prove that precision of the target terms is preserved by stepping, and that if a more precise target term converges (steps to a value), so does a less precise target term.

Sum translation $|\delta| = \phi$

$$|+| = |+^?| = +$$
$$|+_i| = |+_i^?| = |+_i^*| = +_i$$

Type translation $|A| = T$

$$|\mathsf{Unit}| = \mathsf{Unit}$$
$$|A_1\,\delta\,A_2| = |A_1|\;|\delta|\;|A_2|$$
$$|A_1 \to A_2| = |A_1| \to |A_2|$$

Typing context trans. $|\Gamma| = \Theta$

$$|\cdot| = \cdot$$
$$|\Gamma, x : A| = |\Gamma|, x : |A|$$

Coercion contexts

$$\begin{aligned}\mathcal{C} ::= {} & [\,] \\ & \mid \mathsf{case}(\mathcal{C}, \mathsf{inj}_i\, x.M_i) \\ & \mid \mathsf{case}(\mathcal{C}, \mathsf{inj}_1\, x_1.M_1, \mathsf{inj}_2\, x_2.M_2) \\ & \mid \langle \phi \Leftarrow \phi' \rangle \mathcal{C} \\ & \mid \lambda x.\,\mathcal{C} \mid \mathcal{C}\, M_2\end{aligned}$$

$\boxed{\delta' \Rightarrow \delta \hookrightarrow \mathcal{C}}$ Coercion $\mathcal{C}$ coerces sum $|\delta'|$ to sum $|\delta|$

$$\frac{|\delta'| \le |\delta|}{\delta' \Rightarrow \delta \hookrightarrow [\,]}\;\mathsf{CoeSub} \qquad \frac{|\delta'| \not\le |\delta|}{\delta' \Rightarrow \delta \hookrightarrow \langle |\delta| \Leftarrow |\delta'| \rangle [\,]}\;\mathsf{CoeCast}$$

$\boxed{A' \Rightarrow A \hookrightarrow \mathcal{C}}$ Coercion $\mathcal{C}$ coerces target type $|A'|$ to $|A|$

$$\frac{}{\mathsf{Unit} \Rightarrow \mathsf{Unit} \hookrightarrow [\,]}\;\mathsf{CoeUnit} \qquad \frac{A_1 \Rightarrow A_1' \hookrightarrow \mathcal{C}_1 \qquad A_2' \Rightarrow A_2 \hookrightarrow \mathcal{C}_2}{(A_1' \to A_2') \Rightarrow (A_1 \to A_2) \hookrightarrow \lambda x.\,\mathcal{C}_2\big[[\,]\;\mathcal{C}_1[x]\big]}\;\mathsf{Coe}{\to}$$

$$\frac{\delta' \in \{+_1^?, +_1\} \qquad A_1' \Rightarrow A_1 \hookrightarrow \mathcal{C}_1 \qquad \delta' \Rightarrow \delta \hookrightarrow \mathcal{C}_3}{\begin{aligned}&(A_1'\,\delta'\,A_2') \Rightarrow (A_1\,\delta\,A_2) \\ &\quad\hookrightarrow \mathcal{C}_3\big[\mathsf{case}([\,], \mathsf{inj}_1\, x_1.\mathsf{inj}_1\, \mathcal{C}_1[x_1])\big]\end{aligned}}\;\mathsf{CoeCase1L}$$

$$\frac{\delta' \in \{+_2^?, +_2\} \qquad A_2' \Rightarrow A_2 \hookrightarrow \mathcal{C}_2 \qquad \delta' \Rightarrow \delta \hookrightarrow \mathcal{C}_3}{\begin{aligned}&(A_1'\,\delta'\,A_2') \Rightarrow (A_1\,\delta\,A_2) \\ &\quad\hookrightarrow \mathcal{C}_3\big[\mathsf{case}([\,], \mathsf{inj}_2\, x_2.\mathsf{inj}_2\, \mathcal{C}_2[x_2])\big]\end{aligned}}\;\mathsf{CoeCase1R}$$

$$\frac{\begin{array}{cccc} & +_1^? \Rightarrow \delta' \hookrightarrow \mathcal{C}_1' & +_2^? \Rightarrow \delta' \hookrightarrow \mathcal{C}_2' & \\ \delta' \in \{+^?, +_1^*, +_2^*, +\} & A_1' \Rightarrow A_1 \hookrightarrow \mathcal{C}_1 & A_2' \Rightarrow A_2 \hookrightarrow \mathcal{C}_2 & \delta' \Rightarrow \delta \hookrightarrow \mathcal{C}_3\end{array}}{(A_1'\,\delta'\,A_2') \Rightarrow (A_1\,\delta\,A_2) \;\hookrightarrow\; \mathcal{C}_3\big[\mathsf{case}([\,], \mathsf{inj}_1\, x_1.\mathcal{C}_1'[\mathsf{inj}_1\, \mathcal{C}_1[x_1]], \mathsf{inj}_2\, x_2.\mathcal{C}_2'[\mathsf{inj}_2\, \mathcal{C}_2[x_2]])\big]}\;\mathsf{CoeCase2}$$

$\boxed{\Gamma \vdash e : A \hookrightarrow M}$ Under typing context $\Gamma$, expression $e$ has type $A$ and translates to target term $M$

$$\frac{\Gamma(x) = A}{\Gamma \vdash x : A \hookrightarrow x}\;\mathsf{STVar} \qquad \frac{\begin{array}{cc} & A' \rightsquigarrow A \\ \Gamma \vdash e : A' \hookrightarrow M' & A' \Rightarrow A \hookrightarrow \mathcal{C}\end{array}}{\Gamma \vdash e : A \hookrightarrow \mathcal{C}[M']}\;\mathsf{STCSub} \qquad \frac{\Gamma \vdash e : A \hookrightarrow M}{\Gamma \vdash (e :: A) : A \hookrightarrow M}\;\mathsf{STAnno} \qquad \frac{}{\Gamma \vdash () : \mathsf{Unit} \hookrightarrow ()}\;\mathsf{STUnitIntro}$$

$$\frac{\Gamma \vdash e : A_i \hookrightarrow M}{\Gamma \vdash \mathsf{inj}_i\, e : (A_1 +_i^? A_2) \hookrightarrow \mathsf{inj}_i\, M}\;\mathsf{STSumIntro}$$

$$\frac{\Gamma \vdash e_0 : A_1 +_i^* A_2 \hookrightarrow M_0 \qquad \Gamma, x : A_i \vdash e : A \hookrightarrow M}{\Gamma \vdash \mathsf{case}(e_0, \mathsf{inj}_i\, x.e) : A \hookrightarrow \mathsf{case}(M_0, \mathsf{inj}_i\, x.M)}\;\mathsf{STSumElim1}$$

$$\frac{\begin{array}{cc} & \Gamma, x_1 : A_1 \vdash e_1 : A \hookrightarrow M_1 \\ \Gamma \vdash e_0 : A_1 + A_2 \hookrightarrow M_0 & \Gamma, x_2 : A_2 \vdash e_2 : A \hookrightarrow M_2\end{array}}{\begin{aligned}&\Gamma \vdash \mathsf{case}(e_0, \mathsf{inj}_1\, x_1.e_1, \mathsf{inj}_2\, x_2.e_2) : A \\ &\quad\hookrightarrow \mathsf{case}(M_0, \mathsf{inj}_1\, x_1.M_1, \mathsf{inj}_2\, x_2.M_2)\end{aligned}}\;\mathsf{STSumElim2}$$

$$\frac{\Gamma, x : A_1 \vdash e : A_2 \hookrightarrow M}{\Gamma \vdash \lambda x.\, e : A_1 \to A_2 \hookrightarrow \lambda x.\, M}\;\mathsf{ST}{\to}\mathsf{Intro} \qquad \frac{\Gamma \vdash e_1 : A_1 \to A_2 \hookrightarrow M_1 \qquad \Gamma \vdash e_2 : A_1 \hookrightarrow M_2}{\Gamma \vdash e_1\, e_2 : A_2 \hookrightarrow M_1\, M_2}\;\mathsf{ST}{\to}\mathsf{Elim}$$

**Figure 12.** Type-directed translation

Our relation, and the form of the result, were inspired by the approximation relation of Ahmed et al. (2011), as well as the term precision relation of Siek et al. (2015).

For source expressions, we defined $e' \sqsubseteq e$ simply by applying $\sqsubseteq$ to the types in annotations. For target terms, we have no type precision relation; the target type system only has static sums, so $T' \sqsubseteq T$ would degenerate to $T' = T$. Instead, we define target precision $\preccurlyeq$ for terms only.

If $e' \sqsubseteq e$, and these expressions translate to $M'$ and $M$ respectively, we want to show $M' \preccurlyeq M$. The difference between $e'$ and $e$ is only in their annotations, so $M'$ and $M$ must share a lot of structure—except that different annotations may lead to different casts. Thus, most of the rules in Figure 13 are homomorphic.

What about casts, which can step to $\mathtt{matchfail}$? A static source typing is very precise, and the target term it produces never fails, so we might expect a more precisely typed term to "fail less"—but this would lead us astray. A better intuition is that imprecisely typed code "doesn't care", so it tends *not* to fail—while precisely typed code *can* fail, if it collides with imprecisely typed code. Therefore, terms with casts should be *more* precise than terms without. In addition, since casts can step to $\mathtt{matchfail}$, and we want stepping to preserve precision, $\mathtt{matchfail} \preccurlyeq M$ for any $M$.

Given two terms with casts $M' = \langle \phi_2' \Leftarrow \phi_1' \rangle$ and $M = \langle \phi_2 \Leftarrow \phi_1 \rangle$, we will consider $M'$ more precise than $M$ if the cast in $M'$ is more precise: $\langle \phi_2' \Leftarrow \phi_1' \rangle \preccurlyeq \langle \phi_2 \Leftarrow \phi_1 \rangle$. Let $\mathtt{ac}$ be a cast; it must be either a safe cast $\mathtt{sc}$ like $\langle + \Leftarrow + \rangle$ or $\langle + \Leftarrow +_1 \rangle$, a backward cast $\mathtt{bc}$ of the form $\langle +_i \Leftarrow + \rangle$, or a (doomed) match-failure cast $\mathtt{mc}$—$\langle +_2 \Leftarrow +_1 \rangle$ or $\langle +_1 \Leftarrow +_2 \rangle$. These are classified by the grammar in Figure 13.

Equal casts should be equally precise, so rule $\mathsf{Cast}{\preccurlyeq}\mathsf{Refl}$ makes the relation $\mathtt{ac}' \preccurlyeq \mathtt{ac}$ reflexive. Following the idea that the more precisely typed term should "fail more", a safer cast should be *less* precise; this leads to $\mathsf{CastM}{\preccurlyeq}\mathsf{B}$, $\mathsf{CastB}{\preccurlyeq}\mathsf{S}$, and $\mathsf{CastM}{\preccurlyeq}\mathsf{S}$.

The other rules are subtle. They compare *particular* safe casts and/or backward casts, relying implicitly on typing. For example, the last rule says (with $i = 1$) that $\langle + \Leftarrow + \rangle \preccurlyeq \langle + \Leftarrow +_1 \rangle$. We will ultimately need to show that if the cast on the left succeeds, so does the cast on the right. The left-hand cast is $\langle + \Leftarrow + \rangle$, which always succeeds. The right-hand cast succeeds if it is given $\mathsf{inj}_1$. If the value being cast is well-typed, then (by $\mathsf{TCast}$) it will indeed have type $+_1$.

Finally, note that a more precise source typing may result in a one-armed case in a coercion, while the less precise typing results in a two-armed case. For example, $+^?$ is less precise than $+_1$;

| | | |
|---|---|---|
| Safe casts | $\mathsf{sc}$ ::= | $\langle +_1 \Leftarrow +_1 \rangle \mid \langle + \Leftarrow +_1 \rangle$ $\mid \langle +_2 \Leftarrow +_2 \rangle \mid \langle + \Leftarrow +_2 \rangle$ $\mid \langle + \Leftarrow + \rangle$ |
| Backward casts | $\mathsf{bc}$ ::= | $\langle +_1 \Leftarrow + \rangle \mid \langle +_2 \Leftarrow + \rangle$ |
| Match-failure casts | $\mathsf{mc}$ ::= | $\langle +_2 \Leftarrow +_1 \rangle \mid \langle +_1 \Leftarrow +_2 \rangle$ |
| Casts | $\mathsf{ac}$ ::= | $\mathsf{sc} \mid \mathsf{bc} \mid \mathsf{mc}$ |

$\boxed{\mathsf{ac}' \preccurlyeq \mathsf{ac}}$ Cast $\mathsf{ac}'$ is more precise than $\mathsf{ac}$

$$\frac{}{\mathsf{ac} \preccurlyeq \mathsf{ac}}\ \mathsf{Cast{\preccurlyeq}Refl} \qquad \frac{}{\mathsf{mc} \preccurlyeq \mathsf{bc}}\ \mathsf{CastM{\preccurlyeq}B} \qquad \frac{}{\mathsf{bc} \preccurlyeq \mathsf{sc}}\ \mathsf{CastB{\preccurlyeq}S} \qquad \frac{}{\mathsf{mc} \preccurlyeq \mathsf{sc}}\ \mathsf{CastM{\preccurlyeq}S}$$

$$\frac{}{\langle +_i \Leftarrow +_i \rangle \preccurlyeq \langle +_i \Leftarrow + \rangle} \qquad \frac{\mathsf{sc} \in \{\langle + \Leftarrow +_i \rangle, \langle + \Leftarrow + \rangle\}}{\langle +_i \Leftarrow +_i \rangle \preccurlyeq \mathsf{sc}}$$

$$\frac{\mathsf{sc} \in \{\langle + \Leftarrow + \rangle, \langle +_i \Leftarrow +_i \rangle\}}{\langle + \Leftarrow +_i \rangle \preccurlyeq \mathsf{sc}} \qquad \frac{\mathsf{sc} \in \{\langle + \Leftarrow +_i \rangle, \langle +_i \Leftarrow +_i \rangle\}}{\langle + \Leftarrow + \rangle \preccurlyeq \mathsf{sc}}$$

$\boxed{M' \preccurlyeq M}$ Target term $M'$ is more precise than $M$

$$\frac{}{() \preccurlyeq ()} \qquad \frac{}{x \preccurlyeq x} \qquad \frac{M' \preccurlyeq M}{\lambda x.\, M' \preccurlyeq \lambda x.\, M} \qquad \frac{M_1' \preccurlyeq M_1 \qquad M_2' \preccurlyeq M_2}{M_1'\, M_2' \preccurlyeq M_1\, M_2} \qquad \frac{M' \preccurlyeq M}{(\mathsf{inj}_i\, M') \preccurlyeq (\mathsf{inj}_i\, M)}$$

$$\frac{M' \preccurlyeq M \qquad \langle \phi_2' \Leftarrow \phi_1' \rangle \preccurlyeq \langle \phi_2 \Leftarrow \phi_1 \rangle}{\langle \phi_2' \Leftarrow \phi_1' \rangle M' \preccurlyeq \langle \phi_2 \Leftarrow \phi_1 \rangle M} \qquad \frac{M' \preccurlyeq M \qquad M \neq \langle \phi_2 \Leftarrow \phi_1 \rangle \cdots}{\langle \phi_2' \Leftarrow \phi_1' \rangle M' \preccurlyeq M} \qquad \frac{}{\mathsf{matchfail} \preccurlyeq M}$$

$$\frac{M' \preccurlyeq M \qquad M_i' \preccurlyeq M_i}{\mathsf{case}(M', \mathsf{inj}_i\, x.M_i') \preccurlyeq \mathsf{case}(M, \mathsf{inj}_i\, x.M_i)} \qquad \frac{M' \preccurlyeq M \qquad M_i' \preccurlyeq M_i}{\mathsf{case}(M', \mathsf{inj}_i\, x_i.M_i') \preccurlyeq \mathsf{case}(M, \mathsf{inj}_1\, x_1.M_1, \mathsf{inj}_2\, x_2.M_2)}$$

$$\frac{M' \preccurlyeq M \qquad M_1' \preccurlyeq M_1 \qquad M_2' \preccurlyeq M_2}{\mathsf{case}(M', \mathsf{inj}_1\, x_1.M_1', \mathsf{inj}_2\, x_2.M_2') \preccurlyeq \mathsf{case}(M, \mathsf{inj}_1\, x_1.M_1, \mathsf{inj}_2\, x_2.M_2)}$$

**Figure 13.** Precision $\preccurlyeq$ on target terms

coercing $+_1$ to $+$ results in one-armed case, and coercing $+^?$ to $+$ results in a two-armed case. Hence, a one-armed case can be more precise than a two-armed case.

### 5.4 Metatheory

The target system satisfies preservation and progress:

**Theorem 6** (Type preservation)**.**
*If* $\cdot \vdash M : T$ *and* $M \mapsto M'$ *then* $\cdot \vdash M' : T$.

**Theorem 7** (Progress)**.**
*If* $\cdot \vdash M : T$ *then either (a)* $M$ *is a value, or (b) there exists* $M'$ *such that* $M \mapsto M'$*, or (c)* $M = \mathsf{matchfail}$.

By itself, the above progress statement leaves open the possibility that a well-typed target term $M$ will step to $\mathsf{matchfail}$. However, if $M$ has no casts, it will not step to $\mathsf{matchfail}$.

**Theorem 8** ($\mathsf{matchfail}$-freeness)**.**
*If* $M$ *is cast-free and* $\mathsf{matchfail}$*-free and* $M \mapsto M'$ *then* $M'$ *is cast-free and* $\mathsf{matchfail}$*-free.*

For cast-free terms, combining Theorems 7 and 8 gives a version of progress without the possibility of match failure.

**Corollary.** *If* $M$ *is cast-free and* $\mathsf{matchfail}$*-free and* $\cdot \vdash M : T$ *then either (a)* $M$ *is a value, or (b) there exists* $M'$ *such that* $M \mapsto M'$.

We also prove that the translation takes well-typed source programs to well-typed target programs. The theorem takes a type assignment derivation, but Theorem 2 can produce such a derivation from a bidirectional typing derivation.

**Theorem 9** (Translation soundness)**.**
*If* $\Gamma \vdash e : A$ *then there exists* $M$ *such that* $\Gamma \vdash e : A \hookrightarrow M$ *and* $|\Gamma| \vdash M : |A|$.

The proof relies on several lemmas, e.g. that the generated coercions $\mathcal{C}$ are well-typed; see the supplementary material.

A great advantage of static typing is that, for a suitable definition of "wrong", static programs don't go wrong. The theorem below proves that translating a static program yields a target term $M$ that has no casts; by Theorem 8, $M$ will never step to $\mathsf{matchfail}$.

**Theorem 10** (Static derivations don't have match failures)**.**
*If* $\Gamma^S \vdash e^S \Leftarrow A^S$ *or* $\Gamma^S \vdash e^S \Rightarrow A^S$
*then there exists* $M$ *such that* $\Gamma^S \vdash e^S : A^S \hookrightarrow M$
*and* $M$ *is free of casts and* $\mathsf{matchfail}$.

Together, preservation and progress correspond to Theorem 3 (type safety) of Siek et al. (2015, p. 9). Their *blame-subtyping* Theorem 4 says that safe casts (casts from a subtype to a supertype) cannot be blamed (cannot fail); our translation does not insert safe casts at all, and our Theorem 10 shows that expressions without dynamic sums produce target terms without casts.

The remaining results concern precision. We show that more precise annotations translate to more precise terms, that target precision is preserved by stepping, and that if a target term converges, then a less precise version also converges.

We must note that the first of these results, Theorem 11, uses a modified version of the translation: one that always inserts casts, even safe ones; this simplifies part of the proof. In effect, the modified translation (Figure 21 in the appendix) does not have rule $\mathsf{CoeSub}$ and always uses rule $\mathsf{CoeCast}$. Similarly, we modify $\mathsf{CoeCase1L}$ and $\mathsf{CoeCase1R}$ to always insert casts within each arm, like $\mathcal{C}_1'$ and $\mathcal{C}_2'$ in $\mathsf{CoeCase2}$. Since the only difference is the presence of casts that cannot fail, the terms generated by either translation must both step to the same value, or both generate $\mathsf{matchfail}$.

**Theorem 11** (Translation preserves precision)**.**
*Suppose* $\Gamma' \sqsubseteq \Gamma$ *and* $e' \sqsubseteq e$.

1. *If* $\Gamma' \vdash e' \Leftarrow A'$ *and* $\Gamma \vdash e \Leftarrow A$ *and* $A' \sqsubseteq A$ *then* $\Gamma' \vdash e' : A' \hookrightarrow M'$ *and* $\Gamma \vdash e : A \hookrightarrow M$ *where* $M' \preccurlyeq M$.
2. *If* $\Gamma' \vdash e' \Rightarrow A'$ *and* $\Gamma \vdash e \Rightarrow A$ *then* $\Gamma' \vdash e' : A' \hookrightarrow M'$ *and* $\Gamma \vdash e : A \hookrightarrow M$ *where* $A' \sqsubseteq A$ *and* $M' \preccurlyeq M$.

**Theorem 12** (Stepping preserves precision)**.**
*If* $\cdot \vdash M_1' : T_1'$ *and* $\cdot \vdash M_1 : T_1$ *and* $M_1' \preccurlyeq M_1$ *and* $M_1' \mapsto M_2'$ *then either*
*(a)* $M_1$ *is a value and* $M_2' \preccurlyeq M_1$*, or*
*(b) there exists* $M_2$ *such that* $M_1 \mapsto M_2$ *and* $M_2' \preccurlyeq M_2$*, or*
*(c)* $M_1 = \mathsf{matchfail}$ *and* $M_2' \preccurlyeq M_1$.

**Definition 1.** *A closed term* $M$ converges *if* $M \mapsto^* W$ *for some value* $W$*, and* diverges *if the stepping sequence never terminates.*

Note that `matchfail` neither converges nor diverges, and that divergence is not possible in our language.

**Theorem 13** ($\preccurlyeq$ respects convergence)**.**
*If* $M' \preccurlyeq M$ *where* $\cdot \vdash M' : T'$ *and* $\cdot \vdash M : T$
*and* $M'$ *converges then* $M$ *also converges.*

If $M' \preccurlyeq M$, and they converge to injections $\mathtt{inj}_i\, W'$ and $\mathtt{inj}_k\, W$, then Theorem 13 gives $\mathtt{inj}_i\, W' \preccurlyeq \mathtt{inj}_k\, W$. By inversion on the definition of $\preccurlyeq$, we have $i = k$. Similar results would hold if $\preccurlyeq$ were extended for base types.

Together with Theorem 11, this means that if we translate two source expressions $e' \sqsubseteq e$ to $M'$ and $M$, and $M'$ converges to a value of base type, $M$ will converge to the same value. This corresponds to Theorem 5 (gradual guarantee), part 2, of Siek et al. (2015).

## 6. Related Work

***Sums and refinements.*** Sum types are well-established in a variety of programming languages, though practical languages tend to embed them within larger mechanisms: ML datatypes can encode sums, but also recursion. Refinement type systems, such as datasort refinements (Freeman and Pfenning 1991; Davies 2005) and indexed types (Xi and Pfenning 1999), have been built on these larger mechanisms. This gives a close connection to practice, but needs additional machinery such as constructor types and signatures. Such machinery is not central to our investigation; in contrast, we distill datasort refinements to one essential feature: distinguishing whether we have a left or right injection.

These systems often have a refinement relation $\sqsubset$: if $A$ is a sort (refined type) and $\tau$ is an unrefined type, $A \sqsubset \tau$ says that $A$ refines $\tau$. Both the symbol and the high-level concept resemble our relation $A' \sqsubseteq A$, but the refinement relation is more rigid: it cannot compare two sorts, or two unrefined types, and it certainly cannot derive $(A_1 \to A) \sqsubset (A_1 \to \tau)$, where $(A_1 \to \tau)$ mixes a refined type $A_1$ with an unrefined type $\tau$. Nonetheless, the covariance of this relation on function types—in contrast to subtyping, which must be contravariant—made us more confident that our precision relation should be covariant.

Koot and Hage (2015) formulate a constraint-based type system that analyzes pattern matches, using a characterization of data somewhat reminiscent of datasort refinements. Their system needs no type annotations, but is (necessarily) incomplete.

***Gradual typing.*** Our approach to expressing uncertainty in a type system was inspired by gradual typing, introduced by Siek and Taha (2006), in which ? (often written $\star$) is an uncertain type (it could be $\mathsf{Int}$, a function type, or anything else). We confine uncertainty to refinement properties of sum types, making the effect on the overall type system less dramatic; still, several mechanisms of gradual typing appear in our work. For example, we also have precision relations on types and (through annotations) expressions.

Our directed consistency is somewhat similar to consistent subtyping for gradual object-based languages (Siek and Taha 2007). Consistent subtyping augments subsumption with consistent equality (roughly, gain *and* loss of precision) on either the subtype or supertype, but not both. Drawing on abstract interpretation, Garcia et al. (2016) give a different but equivalent formulation of consistent subtyping. In these systems, the underlying subtyping relation is defined over static types only. Allende et al. (2014) also have a notion of directed consistency, but the connection to our relation is less clear.

Siek et al. (2015) propose several criteria as desirable for gradual type systems. We prove properties that correspond to some of their criteria: Theorems 5 and 15 correspond to the first parts of Theorems 1 and 2 of Siek et al. (2015), our Theorem 10 corresponds to their Theorem 4, our Theorem 4 corresponds to part 1 of their Theorem 5 (gradual guarantee), and our Theorems 11 and 13 corresponds to part 2 of their Theorem 5.

Some systems of gradual typing include a notion of *blame* (Wadler and Findler 2009), associating program labels to casts so that a failing cast "blames" some program location. It may be possible to incorporate blame into our approach; we omit it to focus on other issues.

We are not the first to apply ideas from gradual typing to less-traditional areas: for example, Bañados Schwerter et al. (2014) develop a gradual effect system, and McDonell et al. (2016) develop a tool for moving between ADTs and more precise GADTs.

***Bidirectional typing.*** Originating as folklore and first discussed explicitly by Pierce and Turner (1998), bidirectional typing has been used extensively in type systems for which full inference is undecidable or otherwise problematic (Freeman and Pfenning 1991; Coquand 1996; Xi and Pfenning 1999; Davies and Pfenning 2000; Pientka 2008). A strength of many bidirectional type systems, sometimes overlooked, is that they have some variety of subformula property. In some systems, this property serves to make type checking more feasible—for example, for Davies (2005) and Dunfield (2007), it controls the spread of intersection types. For Dunfield (2015), where evaluation order is implicit in terms and explicit in types, it prevents the spontaneous generation of by-name types; in our system, it prevents the spontaneous generation of gradual sum types.

The gradual type system of Garcia and Cimini (2015, p. 306) is not bidirectional, but enjoys a similar property: "dynamicity [the uncertain type ?] is introduced only via program annotations". However, their rules can be viewed as a bidirectional system that always synthesizes, except at annotations.

## 7. Future Work

We plan to implement the bidirectional type system, which will allow us to test whether our approach is practical. We are particularly interested in whether our formulation of precision, combined with the annotation discipline of bidirectional typing, strikes a good balance: the annotation burden should be reasonable, but imprecision should not appear out of nowhere. Also, it is unclear whether programmers would have any use for the sum types $+^?_i$ and $+^*_i$; if not, error messages should read "expected $+_1$ or $+^?$" rather than "expected $+^*_1$", for example.

We would also like to enrich the language with intersection types, recursive types, and polymorphism. Intersection types are important for datasort refinements: for example, if we encode booleans as $\mathsf{Unit} + \mathsf{Unit}$, the datasorts $\mathsf{True}$ and $\mathsf{False}$ are $\mathsf{Unit} +_1 \mathsf{Unit}$ and $\mathsf{Unit} +_2 \mathsf{Unit}$. Then negation should have type $(\mathsf{True} \to \mathsf{False}) \cap (\mathsf{False} \to \mathsf{True})$. We also want to evaluate the run-time efficiency of coercions—a common concern in gradual type systems.


## Acknowledgments

We would like to thank Ronald Garcia, Felipe Bañados Schwerter, Joey Eremondi, Rui Ge, Jodi Spacek, Alec Thériault, and the anonymous reviewers for their feedback on several versions of this work.


## References


Martín Abadi, Luca Cardelli, Benjamin Pierce, and Gordon Plotkin. Dynamic typing in a statically typed language. *ACM Trans. Prog. Lang. Syst.*, 13(2):237–268, 1991.

Amal Ahmed, Robert Bruce Findler, Jeremy G. Siek, and Philip Wadler. Blame for all. In *Principles of Programming Languages*, pages 201–214, 2011.

Esteban Allende, Johan Fabry, Ronald Garcia, and Éric Tanter. Confined gradual typing. In *OOPSLA*, pages 251–270, 2014.

Felipe Bañados Schwerter, Ronald Garcia, and Éric Tanter. A theory of gradual effect systems. In *ICFP*, pages 283–295, 2014.

Thierry Coquand. An algorithm for type-checking dependent types. *Science of Computer Programming*, 26(1–3):167–177, 1996.

Rowan Davies. *Practical Refinement-Type Checking*. PhD thesis, Carnegie Mellon University, 2005. CMU-CS-05-110.

Rowan Davies and Frank Pfenning. Intersection types and computational effects. In *ICFP*, pages 198–208, 2000.

Michael Dummett. *The Logical Basis of Metaphysics*. Harvard University Press, 1991. The William James Lectures, 1976.

Joshua Dunfield. *A Unified System of Type Refinements*. PhD thesis, Carnegie Mellon University, 2007. CMU-CS-07-129.

Joshua Dunfield. Elaborating evaluation-order polymorphism. In *Int'l Conf. Functional Programming*, 2015. arXiv:`1504.07680 [cs.PL]`.

Joshua Dunfield and Neelakantan R. Krishnaswami. Complete and easy bidirectional typechecking for higher-rank polymorphism. In *ICFP*, 2013. arXiv:`1306.6032 [cs.PL]`.

Joshua Dunfield and Frank Pfenning. Tridirectional typechecking. In *Principles of Programming Languages*, pages 281–292, 2004.

Tim Freeman. *Refinement Types for ML*. PhD thesis, Carnegie Mellon University, 1994. CMU-CS-94-110.

Tim Freeman and Frank Pfenning. Refinement types for ML. In *Programming Language Design and Implementation*, pages 268–277, 1991.

Ronald Garcia and Matteo Cimini. Principal type schemes for gradual programs. In *Principles of Programming Languages*, pages 303–315, 2015.

Ronald Garcia, Alison M. Clark, and Éric Tanter. Abstracting gradual typing. In *Principles of Programming Languages*, pages 429–442, 2016.

Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1934. English translation, *Investigations into logical deduction*, in M. Szabo, editor, *Collected papers of Gerhard Gentzen* (North-Holland, 1969), pages 68–131.

Ruud Koot and Jurriaan Hage. Type-based exception analysis for non-strict higher-order functional languages with imprecise exception semantics. In *Proceedings of the 2015 Workshop on Partial Evaluation and Program Manipulation*, pages 127–138, 2015.

Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, 1(1):11–60, 1996. Notes for lectures given in 1983 in Siena, Italy.

Trevor L. McDonell, Timothy A. K. Zakian, Matteo Cimini, and Ryan R. Newton. Ghostbuster: A tool for simplifying and converting GADTs. In *ICFP*, pages 338–350, 2016.

Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.

Frank Pfenning. Lecture notes on harmony. Lecture notes for 15–317: Constructive Logic, Carnegie Mellon University, September 2009. `www.cs.cmu.edu/~fp/courses/15317-f09/lectures/03-harmony.pdf`.

Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11(4):511–540, 2001.

Brigitte Pientka. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *Principles of Programming Languages*, pages 371–382, 2008.

Brigitte Pientka and Joshua Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In *Int'l Joint Conference on Automated Reasoning (IJCAR)*, pages 15–21, 2010.

Benjamin C. Pierce and David N. Turner. Local type inference. In *Principles of Programming Languages*, pages 252–265, 1998. Full version in *ACM Trans. Prog. Lang. Sys.*, 22(1):1–44, 2000.

Dag Prawitz. *Natural Deduction*. Almqvist & Wiksells, 1965.

Jeremy Siek and Walid Taha. Gradual typing for objects. In *European Conference on Object-Oriented Programming*, pages 2–27. Springer, 2007.

Jeremy G. Siek and Walid Taha. Gradual typing for functional languages. In *Proceedings of the Scheme and Functional Programming Workshop*, pages 81–92, September 2006.

Jeremy G. Siek and Manish Vachharajani. Gradual typing with unification-based inference. In *Symposium on Dynamic Languages (DLS)*, pages 7:1–7:12, 2008.

Jeremy G. Siek, Michael M. Vitousek, Matteo Cimini, and John Tang Boyland. Refined criteria for gradual typing. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 32. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.

Philip Wadler and Robert Bruce Findler. Well-typed programs can't be blamed. In *European Symposium on Programming*, pages 1–16, 2009.

Hongwei Xi and Frank Pfenning. Dependent types in practical programming. In *Principles of Programming Languages*, pages 214–227, 1999.

Noam Zeilberger. *The Logical Basis of Evaluation Order and Pattern-Matching*. PhD thesis, Carnegie Mellon University, 2009. CMU-CS-09-122.

# Appendix to "Sums of Uncertainty: Refinements go gradual" (POPL 2017)

## A. Dynamic System

$$
\begin{array}{lrcl}
\text{Dynamic expressions} & e^{\mathsf{D}} & ::= & () \mid x \mid \lambda x.\, e^{\mathsf{D}} \mid e_1^{\mathsf{D}}\, e_2^{\mathsf{D}} \mid \mathsf{inj}_i\, e^{\mathsf{D}} \mid (e^{\mathsf{D}} :: A^{\mathsf{D}}) \\
& & & \mid \mathsf{case}(e^{\mathsf{D}}, \mathsf{inj}_1\, x_1.e_1^{\mathsf{D}}, \mathsf{inj}_2\, x_2.e_2^{\mathsf{D}}) \mid \mathsf{case}(e^{\mathsf{D}}, \mathsf{inj}_i\, x.e_i^{\mathsf{D}}) \\
\text{Dynamic types} & A^{\mathsf{D}} & ::= & \mathsf{Unit} \mid A_1^{\mathsf{D}} +^{?} A_2^{\mathsf{D}} \mid A_1^{\mathsf{D}} \to A_2^{\mathsf{D}} \\
\text{Dynamic typing contexts} & \Gamma^{\mathsf{D}} & ::= & \cdot \mid \Gamma^{\mathsf{D}}, x : A^{\mathsf{D}}
\end{array}
$$

$\boxed{\begin{array}{l}\Gamma^{\mathsf{D}} \vdash_{\mathsf{D}} e^{\mathsf{D}} \Leftarrow A^{\mathsf{D}} \\ \Gamma^{\mathsf{D}} \vdash_{\mathsf{D}} e^{\mathsf{D}} \Rightarrow A^{\mathsf{D}}\end{array}}$ Under typing context $\Gamma^{\mathsf{D}}$, expression $e^{\mathsf{D}}$ checks against type $A^{\mathsf{D}}$
Under typing context $\Gamma^{\mathsf{D}}$, expression $e^{\mathsf{D}}$ synthesizes type $A^{\mathsf{D}}$

$$
\frac{\Gamma^{\mathsf{D}}(x) = A^{\mathsf{D}}}{\Gamma^{\mathsf{D}} \vdash_{\mathsf{D}} x \Rightarrow A^{\mathsf{D}}}\ \mathsf{DVar}
\qquad
\frac{\Gamma^{\mathsf{D}} \vdash_{\mathsf{D}} e^{\mathsf{D}} \Rightarrow A^{\mathsf{D}}}{\Gamma^{\mathsf{D}} \vdash_{\mathsf{D}} e^{\mathsf{D}} \Leftarrow A^{\mathsf{D}}}\ \mathsf{DSub}
\qquad
\frac{\Gamma^{\mathsf{D}} \vdash_{\mathsf{D}} e^{\mathsf{D}} \Leftarrow A^{\mathsf{D}}}{\Gamma^{\mathsf{D}} \vdash_{\mathsf{D}} (e^{\mathsf{D}} :: A^{\mathsf{D}}) \Rightarrow A^{\mathsf{D}}}\ \mathsf{DAnno}
\qquad
\frac{}{\Gamma^{\mathsf{D}} \vdash_{\mathsf{D}} () \Leftarrow \mathsf{Unit}}\ \mathsf{DUnitIntro}
$$

$$
\frac{\Gamma^{\mathsf{D}}, x : A_1^{\mathsf{D}} \vdash_{\mathsf{D}} e^{\mathsf{D}} \Leftarrow A_2^{\mathsf{D}}}{\Gamma^{\mathsf{D}} \vdash_{\mathsf{D}} \lambda x.\, e^{\mathsf{D}} \Leftarrow A_1^{\mathsf{D}} \to A_2^{\mathsf{D}}}\ \mathsf{D{\to}Intro}
\qquad
\frac{\Gamma^{\mathsf{D}} \vdash_{\mathsf{D}} e_1^{\mathsf{D}} \Rightarrow A_1^{\mathsf{D}} \to A_2^{\mathsf{D}} \qquad \Gamma^{\mathsf{D}} \vdash_{\mathsf{D}} e_2^{\mathsf{D}} \Leftarrow A_1^{\mathsf{D}}}{\Gamma^{\mathsf{D}} \vdash_{\mathsf{D}} e_1^{\mathsf{D}}\, e_2^{\mathsf{D}} \Rightarrow A_2^{\mathsf{D}}}\ \mathsf{D{\to}Elim}
\qquad
\frac{\Gamma^{\mathsf{D}} \vdash_{\mathsf{D}} e^{\mathsf{D}} \Leftarrow A_i^{\mathsf{D}}}{\Gamma^{\mathsf{D}} \vdash_{\mathsf{D}} \mathsf{inj}_i\, e^{\mathsf{D}} \Leftarrow (A_1^{\mathsf{D}} +^{?} A_2^{\mathsf{D}})}\ \mathsf{D{+}^{?}Intro}
$$

$$
\frac{\Gamma^{\mathsf{D}} \vdash_{\mathsf{D}} e_0^{\mathsf{D}} \Rightarrow A_1^{\mathsf{D}} +^{?} A_2^{\mathsf{D}} \qquad \Gamma^{\mathsf{D}}, x : A_i^{\mathsf{D}} \vdash_{\mathsf{D}} e^{\mathsf{D}} \Leftarrow A^{\mathsf{D}}}{\Gamma^{\mathsf{D}} \vdash_{\mathsf{D}} \mathsf{case}(e_0^{\mathsf{D}}, \mathsf{inj}_i\, x.e^{\mathsf{D}}) \Leftarrow A^{\mathsf{D}}}\ \mathsf{D{+}^{?}Elim1}
\qquad
\frac{\Gamma^{\mathsf{D}} \vdash_{\mathsf{D}} e_0^{\mathsf{D}} \Rightarrow A_1^{\mathsf{D}} +^{?} A_2^{\mathsf{D}} \qquad \begin{array}{c}\Gamma^{\mathsf{D}}, x_1 : A_1^{\mathsf{D}} \vdash_{\mathsf{D}} e_1^{\mathsf{D}} \Leftarrow A^{\mathsf{D}} \\ \Gamma^{\mathsf{D}}, x_2 : A_2^{\mathsf{D}} \vdash_{\mathsf{D}} e_2^{\mathsf{D}} \Leftarrow A^{\mathsf{D}}\end{array}}{\Gamma^{\mathsf{D}} \vdash_{\mathsf{D}} \mathsf{case}(e_0^{\mathsf{D}}, \mathsf{inj}_1\, x_1.e_1^{\mathsf{D}}, \mathsf{inj}_2\, x_2.e_2^{\mathsf{D}}) \Leftarrow A^{\mathsf{D}}}\ \mathsf{D{+}^{?}Elim2}
$$

**Figure 14.** The dynamic system: the bidirectional system restricted to $+^{?}$

Figure 14 shows the syntax and typing rules for the dynamic system—the restriction of the bidirectional type system to the dynamic sum $+^{?}$.

## B. Omitted Definitions

$\boxed{e' \sqsubseteq e}$ Expression $e'$ is more precise than $e$

$$
\frac{}{() \sqsubseteq ()}
\qquad
\frac{}{x \sqsubseteq x}
\qquad
\frac{e' \sqsubseteq e}{\lambda x.\, e' \sqsubseteq \lambda x.\, e}
\qquad
\frac{e_1' \sqsubseteq e_1 \qquad e_2' \sqsubseteq e_2}{e_1'\, e_2' \sqsubseteq e_1\, e_2}
\qquad
\frac{e' \sqsubseteq e}{(\mathsf{inj}_i\, e') \sqsubseteq (\mathsf{inj}_i\, e)}
\qquad
\frac{e' \sqsubseteq e \qquad A' \sqsubseteq A}{(e' :: A') \sqsubseteq (e :: A)}
$$

$$
\frac{e' \sqsubseteq e \qquad e_1' \sqsubseteq e_1 \qquad e_2' \sqsubseteq e_2}{\mathsf{case}(e', \mathsf{inj}_1\, x_1.e_1', \mathsf{inj}_2\, x_2.e_2') \sqsubseteq \mathsf{case}(e, \mathsf{inj}_1\, x_1.e_1, \mathsf{inj}_2\, x_2.e_2)}
\qquad
\frac{e' \sqsubseteq e \qquad e_i' \sqsubseteq e_i}{\mathsf{case}(e', \mathsf{inj}_i\, x.e_i') \sqsubseteq \mathsf{case}(e, \mathsf{inj}_i\, x.e_i)}
$$

$\boxed{\Gamma' \sqsubseteq \Gamma}$ Typing context $\Gamma'$ is more precise than $\Gamma$

$$
\frac{}{\cdot \sqsubseteq \cdot}
\qquad
\frac{\Gamma' \sqsubseteq \Gamma \qquad A' \sqsubseteq A}{(\Gamma', x : A') \sqsubseteq (\Gamma, x : A)}
$$

**Figure 15.** Precision on expressions and contexts

Several results involve precision of expressions and typing contexts, shown in Figure 15; these are the straightforward lifting of type precision (Figure 4).

$\boxed{e' =: e}$ Expression $e'$ is annotative-ly equivalent to $e$

$$
\frac{}{() =: ()}
\qquad
\frac{}{x =: x}
\qquad
\frac{e' =: e}{e' =: (e :: A)}
$$

$$
\frac{e' =: e}{\lambda x.\, e' =: \lambda x.\, e}
\qquad
\frac{e_1' =: e_1 \qquad e_2' =: e_2}{e_1'\, e_2' =: e_1\, e_2}
\qquad
\frac{e' =: e}{(\mathsf{inj}_i\, e') =: (\mathsf{inj}_i\, e)}
\qquad
\frac{e' =: e \qquad A' = A}{(e' :: A') =: (e :: A)}
$$

$$
\frac{e' =: e \qquad e_1' =: e_1 \qquad e_2' =: e_2}{\mathsf{case}(e', \mathsf{inj}_1\, x_1.e_1', \mathsf{inj}_2\, x_2.e_2') =: \mathsf{case}(e, \mathsf{inj}_1\, x_1.e_1, \mathsf{inj}_2\, x_2.e_2)}
\qquad
\frac{e' =: e \qquad e_i' =: e_i}{\mathsf{case}(e', \mathsf{inj}_i\, x.e_i') =: \mathsf{case}(e, \mathsf{inj}_i\, x.e_i)}
$$

**Figure 16.** Annotation equivalence

# C. Differences from the Original Version

The paper that was submitted to POPL differs in two important ways from the final version.

***No directed consistency.*** In the final version, ChkCSub, SCSub, etc. allow (a) gain of precision, (b) subtyping, and (c) loss of precision, formulated via directed consistency. In contrast, the original system had (in each system) two rules: one rule that allowed subtyping (exactly like a traditional subsumption rule), and one rule that allowed loss of precision. For example, the bidirectional system had

$$\frac{\Gamma \vdash e \Rightarrow A' \qquad A' \leq A}{\Gamma \vdash e \Leftarrow A}\ \text{**ChkSub} \qquad\qquad \frac{\Gamma \vdash e \Rightarrow A' \qquad A' \sqsubseteq A}{\Gamma \vdash e \Leftarrow A}\ \text{**ChkImp}$$

These rules could not type the same expression without an extra annotation (to transition from the checking conclusion of one rule to the synthesizing conclusion of the other).

Moreover, there was no rule to gain precision. In a traditional gradual type system, this would be completely untenable: the point of the "unknown type" in a gradual system is that it can be downcasted to a static type. In the previous version of our system, programmers could write coercions "by hand":

$$f : (A_1 +_1 A_2) \to B, y : (A_1 +^{?} A_2) \vdash f\ \big(\mathsf{case}(y, \mathtt{inj}_1\ x.x)\big) \Rightarrow B$$

But this requires a change to the expression that goes beyond changing an annotation: the expression itself is being changed.

The lack of a way to gain precision, combined with the need for an extra annotation to use subtyping *and* loss of precision, meant that the varying precision property—Theorem 4 in the final version—did not hold. A weaker property—Theorem 14, below—did hold, but this property only provides that some expression $e_j$, which could be more imprecise than $e$, is well typed.

***Different definition of imprecision.*** In Section 2, we explained why $+_1^* \sqsubseteq +_1^?$ doesn't make sense. We also argued against $+_1^? \sqsubseteq +_1^*$, on the basis that in directed consistency (SCSub) one could gain precision from $+_1^*$ to $+_1^?$, then use subtyping from $+_1^?$ to $+_2^*$. In the old system, there was no gain of precision, and even loss of precision could not be combined with subtyping (without extra annotation). Thus, we saw no clear argument against $+_1^? \sqsubseteq +_1^*$, and included it in the relation. However, in the absence of gain of precision, the only way the type system could use this was by moving from $+_i^?$ to $+_i^*$, which was also possible via subtyping.

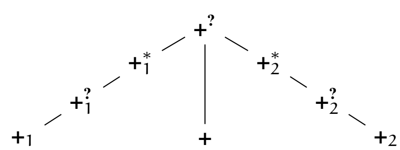



**Figure 17.** Original, obsolete definition of precision

## C.1 Original, weak version of varying precision

Theorem 4 does not hold for the original system. Instead, the following holds, where $e \sqsubseteq: e_j$ means that $e_j$ is a version of $e$ with more-imprecise annotations (like $e' \sqsubseteq e$) *and* extra annotations. For example, $x \sqsubseteq: (x :: A)$.

**Theorem 14** (Weak version of varying precision)**.**

1. *If* $\Gamma' \vdash e' \Leftarrow A'$ *and* $e' \sqsubseteq e$ *and* $\Gamma' \sqsubseteq \Gamma$
   *then there exist* $e_j$ *and* $A$ *such that* $\Gamma \vdash e_j \Leftarrow A$ *and* $e \sqsubseteq: e_j$ *and* $A' \sqsubseteq A$.
2. *If* $\Gamma' \vdash e' \Rightarrow A'$ *and* $e' \sqsubseteq e$ *and* $\Gamma' \sqsubseteq \Gamma$
   *then there exist* $e_j$ *and* $A$
   *such that* $\Gamma \vdash e_j \Rightarrow A$ *and* $e \sqsubseteq: e_j$ *and* $A' \sqsubseteq A$.

Given $e' \sqsubseteq e$, this weak version of varying precision yields some $e_j$ that may be more imprecise, $e' \sqsubseteq e \sqsubseteq: e_j$. This is needed because—in the absence of ChkCSub, which allows precision to be adjusted whenever subsumption is used—a more imprecise annotation may require changing other annotations to make them more imprecise. For example, suppose we are given

$$\begin{aligned} e' &= \big((\lambda x.\, (x :: B +_2 B)) :: (B +_2 B) \to (B +_2 B)\big) \\ e &= \big((\lambda x.\, (x :: B +^{?} B)) :: (B +_2 B) \to (B +_2 B)\big) \end{aligned}$$

We can synthesize $A' = (B +_2 B) \to (B +_2 B)$ for $e'$, but not for $e$, because the inner annotation on $x$ makes the $\lambda$ fail to check against the outer annotation. But we can produce $e_j = \big((\lambda x.\, (x :: B +^{?} B)) :: (B +_2 B) \to (B +^{?} B)\big)$. Now the uses of $+^{?}$ match, and $e_j$ synthesizes $A = (B +_2 B) \to (B +^{?} B)$. The remaining $+_2$ is okay, because of **ChkImp: in $(x :: B +^{?} B)$, we have $\Gamma(x) = B +_2 B$, which is less imprecise than $B +^{?} B$.

# D. Proofs

## D.1 Source System

### D.1.1 Subtyping

**Lemma 1** (Subtyping inversion)**.**

1. *If* $\mathsf{Unit} \le A$ *then* $A = \mathsf{Unit}$.
2. *If* $A' \le \mathsf{Unit}$ *then* $A' = \mathsf{Unit}$.
3. *If* $A_1' \,\delta'\, A_2' \le A$ *then* $A = A_1 \,\delta\, A_2$ *where* $A_1' \le A_1$ *and* $A_2' \le A_2$ *and* $\delta' \le \delta$.
4. *If* $A' \le A_1 \,\delta\, A_2$ *then* $A' = A_1' \,\delta'\, A_2'$ *where* $A_1' \le A_1$ *and* $A_2' \le A_2$ *and* $\delta' \le \delta$.
5. *If* $A_1' \to A_2' \le A$ *then* $A = A_1 \to A_2$ *where* $A_1 \le A_1'$ *and* $A_2' \le A_2$.
6. *If* $A' \le A_1 \to A_2$ *then* $A' = A_1' \to A_2'$ *where* $A_1 \le A_1'$ *and* $A_2' \le A_2$.

*Proof.*

1. By case analysis on $\mathsf{Unit} \le A$.
   - **Case** $\mathsf{Unit} \le \mathsf{Unit}$: Immediate that $A = \mathsf{Unit}$.
2. Symmetric to the previous statement, hence omitted.
3. By case analysis on $A_1' \,\delta'\, A_2' \le A$.
   - **Case** $A_1' \,\delta'\, A_2' \le A_1 \,\delta\, A_2$: Immediate as $A' = A_1' \,\delta'\, A_2'$ and subderivations are $A_1' \le A_1$ and $A_2' \le A_2$ and $\delta' \le \delta$.
4. Symmetric to the previous statement, hence omitted.
5. By case analysis on $A_1' \to A_2' \le A$.
   - **Case** $A_1' \to A_2' \le A_1 \to A_2$: Immediate as $A' = A_1' \to A_2'$ and subderivations are $A_1 \le A_1'$ and $A_2' \le A_2$.
6. Symmetric to the previous statement, hence omitted. □

**Lemma 2** (Reflexivity of subtyping)**.**
*For all types* $A$, *it is the case that* $A \le A$.

*Proof.* By induction on the structure of $A$.

- **Case** $A = \mathsf{Unit}$: By the definition of precision, $A \le A$.
- **Case** $A = A_1 \,\delta\, A_2$: By the induction hypothesis, $A_1 \le A_1$ and $A_2 \le A_2$. By the reflexivity of subsum, $\delta \le \delta$. Thus, by the definition of subtyping, $A \le A$.
- **Case** $A = A_2 \to A_2$: By the induction hypothesis, $A_1 \le A_1$ and $A_2 \le A_2$. Thus, by the definition of subtyping, $A \le A$. □

**Lemma 3** (Transitivity of subtyping)**.**
*If* $A_1 \le A_2$ *and* $A_2 \le A_3$ *then* $A_1 \le A_2$

*Proof.* By induction on the structure of $A_2$.

- **Case** $A_2 = \mathsf{Unit}$:

| | |
|---|---|
| $A_1 \le \mathsf{Unit}$ | Given |
| $\mathsf{Unit} \le A_3$ | Given |
| $A_1 = \mathsf{Unit}$ | By Lemma 1 (Subtyping inversion) |
| $A_3 = \mathsf{Unit}$ | By Lemma 1 (Subtyping inversion) |
| $\mathsf{Unit} \le \mathsf{Unit}$ | By Lemma 2 (Reflexivity of subtyping) |
| $A_1 \le A_3$ | Equivalent |

- **Case** $A_2 = A_{12} \,\delta_2\, A_{22}$:

| | |
|---|---|
| $A_1 \le A_{12} \,\delta_2\, A_{22}$ | Given |
| $A_1 = A_{11} \,\delta_1\, A_{21}$ | By Lemma 1 (Subtyping inversion) |
| $A_{11} \le A_{12}$ | $''$ |
| $A_{21} \le A_{22}$ | $''$ |
| $\delta_1 \le \delta_2$ | $''$ |
| $A_{12} \,\delta_2\, A_{22} \le A_3$ | Given |
| $A_3 = A_{13} \,\delta_3\, A_{23}$ | By Lemma 1 (Subtyping inversion) |
| $A_{12} \le A_{13}$ | $''$ |
| $A_{22} \le A_{23}$ | $''$ |
| $\delta_2 \le \delta_3$ | $''$ |

$$
\begin{array}{rl}
A_{11} \leq A_{13} & \text{By the induction hypothesis} \\
A_{21} \leq A_{23} & \text{By the induction hypothesis} \\
\delta_1 \leq \delta_3 & \text{By the transitivity of } \leq \\
A_{11}\,\delta_1\,A_{21} \leq A_{13}\,\delta_3\,A_{23} & \text{By the definition of } \leq \\
A_1 \leq A_3 & \text{Equivalent}
\end{array}
$$

- **Case** $A_2 = A_{12} \to A_{22}$:

$$
\begin{array}{rl}
A_1 \leq A_{12} \to A_{22} & \text{Given} \\
A_1 = A_{11} \to A_{21} & \text{By Lemma 1 (Subtyping inversion)} \\
A_{12} \leq A_{11} & '' \\
A_{21} \leq A_{22} & '' \\
 & \\
A_{12} \to A_{22} \leq A_3 & \text{Given} \\
A_3 = A_{13} \to A_{23} & \text{By Lemma 1 (Subtyping inversion)} \\
A_{13} \leq A_{12} & '' \\
A_{22} \leq A_{23} & '' \\
 & \\
A_{13} \leq A_{11} & \text{By the induction hypothesis} \\
A_{21} \leq A_{23} & \text{By the induction hypothesis} \\
A_{11} \to A_{21} \leq A_{13} \to A_{23} & \text{By the definition of } \leq \\
A_1 \leq A_3 & \text{Equivalent}
\end{array}
$$

□

### D.1.2 Precision

**Lemma 4** (Precision inversion)**.**

1. *If* $\mathsf{Unit} \sqsubseteq A$ *then* $A = \mathsf{Unit}$.
2. *If* $A' \sqsubseteq \mathsf{Unit}$ *then* $A' = \mathsf{Unit}$.
3. *If* $A'_1\,\delta'\,A'_2 \sqsubseteq A$ *then* $A = A_1\,\delta\,A_2$ *where* $A'_1 \sqsubseteq A_1$ *and* $A'_2 \sqsubseteq A_2$ *and* $\delta' \sqsubseteq \delta$.
4. *If* $A' \sqsubseteq A_1\,\delta\,A_2$ *then* $A' = A'_1\,\delta'\,A'_2$ *where* $A'_1 \sqsubseteq A_1$ *and* $A'_2 \sqsubseteq A_2$ *and* $\delta' \sqsubseteq \delta$.
5. *If* $A'_1 \to A'_2 \sqsubseteq A$ *then* $A = A_1 \to A_2$ *where* $A'_1 \sqsubseteq A_1$ *and* $A'_2 \sqsubseteq A_2$.
6. *If* $A' \sqsubseteq A_1 \to A_2$ *then* $A' = A'_1 \to A'_2$ *where* $A'_1 \sqsubseteq A_1$ *and* $A'_2 \sqsubseteq A_2$.

*Proof.*

1. By case analysis on $\mathsf{Unit} \sqsubseteq A$.
   - **Case** $\mathsf{Unit} \sqsubseteq \mathsf{Unit}$: Immediate that $A = \mathsf{Unit}$.
2. Symmetric to the previous statement, hence omitted.
3. By case analysis on $A'_1\,\delta'\,A'_2 \sqsubseteq A$.
   - **Case** $A'_1\,\delta'\,A'_2 \sqsubseteq A_1\,\delta\,A_2$: Immediate as $A' = A'_1\,\delta'\,A'_2$ and subderivations are $A'_1 \sqsubseteq A_1$ and $A'_2 \sqsubseteq A_2$ and $\delta' \sqsubseteq \delta$.
4. Symmetric to the previous statement, hence omitted.
5. By case analysis on $A'_1 \to A'_2 \sqsubseteq A$.
   - **Case** $A'_1 \to A'_2 \sqsubseteq A_1 \to A_2$: Immediate as $A' = A'_1 \to A'_2$ and subderivations are $A'_1 \sqsubseteq A_1$ and $A'_2 \sqsubseteq A_2$.
6. Symmetric to the previous statement, hence omitted. □

**Lemma 5** (Reflexivity of precision)**.**
*For all types* $A$*, it is the case that* $A \sqsubseteq A$.

*Proof.* By induction on the structure of $A$.

- **Case** $A = \mathsf{Unit}$: By the definition of precision, $A \sqsubseteq A$.
- **Case** $A = A_1\,\delta\,A_2$: By the induction hypothesis, $A_1 \sqsubseteq A_1$ and $A_2 \sqsubseteq A_2$. By the reflexivity of precision on sums, $\delta \sqsubseteq \delta$. Thus, by the definition of subtyping, $A \sqsubseteq A$.
- **Case** $A = A_2 \to A_2$: By the induction hypothesis, $A_1 \sqsubseteq A_1$ and $A_2 \sqsubseteq A_2$. Thus, by the definition of subtyping, $A \sqsubseteq A$. □

**Lemma 6** (Transitivity of precision)**.**
*If* $A_1 \sqsubseteq A_2$ *and* $A_2 \sqsubseteq A_3$ *then* $A_1 \sqsubseteq A_2$.

*Proof.* By induction on the structure of $A_2$.

- **Case** $A_2 = \mathsf{Unit}$:

| | |
|---|---|
| $A_1 \sqsubseteq \mathsf{Unit}$ | Given |
| $\mathsf{Unit} \sqsubseteq A_3$ | Given |
| $A_1 = \mathsf{Unit}$ | By Lemma 4 (Precision inversion) |
| $A_3 = \mathsf{Unit}$ | By Lemma 4 (Precision inversion) |
| $\mathsf{Unit} \sqsubseteq \mathsf{Unit}$ | By Lemma 5 (Reflexivity of precision) |
| $A_1 \sqsubseteq A_3$ | Equivalent |

- **Case** $A_2 = A_{12}\,\delta_2\,A_{22}$:

| | |
|---|---|
| $A_1 \sqsubseteq A_{12}\,\delta_2\,A_{22}$ | Given |
| $A_1 = A_{11}\,\delta_1\,A_{21}$ | By Lemma 4 (Precision inversion) |
| $A_{11} \sqsubseteq A_{12}$ | ″ |
| $A_{21} \sqsubseteq A_{22}$ | ″ |
| $\delta_1 \sqsubseteq \delta_2$ | ″ |
| | |
| $A_{12}\,\delta_2\,A_{22} \sqsubseteq A_3$ | Given |
| $A_3 = A_{13}\,\delta_3\,A_{23}$ | By Lemma 4 (Precision inversion) |
| $A_{12} \sqsubseteq A_{13}$ | ″ |
| $A_{22} \sqsubseteq A_{23}$ | ″ |
| $\delta_2 \sqsubseteq \delta_3$ | ″ |
| | |
| $A_{11} \sqsubseteq A_{13}$ | By the induction hypothesis |
| $A_{21} \sqsubseteq A_{23}$ | By the induction hypothesis |
| $\delta_1 \sqsubseteq \delta_3$ | By transitivity of $\sqsubseteq$ |
| $A_{11}\,\delta_1\,A_{21} \sqsubseteq A_{13}\,\delta_3\,A_{23}$ | By the definition of $\sqsubseteq$ |
| $A_1 \sqsubseteq A_3$ | Equivalent |

- **Case** $A_2 = A_{12} \to A_{22}$:

| | |
|---|---|
| $A_1 \sqsubseteq A_{12} \to A_{22}$ | Given |
| $A_1 = A_{11} \to A_{21}$ | By Lemma 4 (Precision inversion) |
| $A_{11} \sqsubseteq A_{12}$ | ″ |
| $A_{21} \sqsubseteq A_{22}$ | ″ |
| | |
| $A_{12} \to A_{22} \sqsubseteq A_3$ | Given |
| $A_3 = A_{13} \to A_{23}$ | By Lemma 4 (Precision inversion) |
| $A_{12} \sqsubseteq A_{13}$ | ″ |
| $A_{22} \sqsubseteq A_{23}$ | ″ |
| | |
| $A_{11} \sqsubseteq A_{13}$ | By the induction hypothesis |
| $A_{21} \sqsubseteq A_{23}$ | By the induction hypothesis |
| $A_{11} \to A_{21} \sqsubseteq A_{13} \to A_{23}$ | By the definition of $\sqsubseteq$ |
| $A_1 \sqsubseteq A_3$ | Equivalent |

□

### D.1.3 Directed Consistency

**Lemma 7** (Reflexivity of directed consistency)**.**
*For all types* $A$, *it is the case that* $A \leadsto A$.

*Proof.* Immediate from Lemma 5 (Reflexivity of precision), Lemma 2 (Reflexivity of subtyping) and rule $\mathsf{DirConsU}$. □

**Lemma 8** (Subtyping obeys directed consistency)**.**
*If* $A \leq B$ *then* $A \leadsto B$.

*Proof.* By Lemma 5 (Reflexivity of precision), $A \sqsubseteq A$ and $B \sqsubseteq B$. It is given that $A \leq B$. Therefore, by rule $\mathsf{DirConsU}$, $A \leadsto B$. □

**Lemma 9** (Loss in precision obeys directed consistency)**.**
*If* $A \sqsubseteq B$ *then* $A \leadsto B$.

*Proof.* By Lemma 5 (Reflexivity of precision), $A \sqsubseteq A$. By Lemma 2 (Reflexivity of subtyping), $A \leq A$. It is given that $A \sqsubseteq B$. Therefore, by rule $\mathsf{DirConsU}$, $A \leadsto B$. □

**Lemma 10** (Gain in precision obeys directed consistency)**.**
*If* $A \sqsubseteq B$ *then* $B \leadsto A$.

*Proof.* It is given that $A \sqsubseteq B$. By Lemma 2 (Reflexivity of subtyping), $A \leq A$. By Lemma 5 (Reflexivity of precision), $A \sqsubseteq A$. Therefore, by rule $\mathsf{DirConsU}$, $B \leadsto A$. □

$\boxed{A' \simeq A}$ Type $A'$ is structurally equivalent to $A$

$$\frac{}{\mathsf{Unit} \simeq \mathsf{Unit}} \qquad \frac{A'_1 \simeq A_1 \qquad A'_2 \simeq A_2}{(A'_1\, \delta'\, A'_2) \simeq (A_1\, \delta\, A_2)} \qquad \frac{A'_1 \simeq A_1 \qquad A'_2 \simeq A_2}{(A'_1 \to A'_2) \simeq (A_1 \to A_2)}$$

**Figure 18.** Source structural equivalence

### D.1.4 Structural Equivalence

**Lemma 11** (Reflexivity of Structural Equivalence)**.**
*For all types $A$, it is the case that $A \simeq A$.*

*Proof.* By induction on the structure of $A$. All cases are immediate by the induction hypothesis and the definition of $\simeq$. □

**Lemma 12** (Symmetry of Structural Equivalence)**.**
*If $A' \simeq A$ then $A \simeq A'$.*

*Proof.* By structural induction on the derivation of $A' \simeq A$. All cases are immediate by the induction hypothesis and the definition of $\simeq$. □

**Lemma 13** (Transitivity of Structural Equivalence)**.**
*If $A_1 \simeq A_2$ and $A_2 \simeq A_3$ then $A_1 \simeq A_3$.*

*Proof.* By induction on the structure of the type $A_2$. All cases are immediate from inversion on structural equivalence, the induction hypothesis, and the definition of $\simeq$. □

**Corollary 14** (Structural Equivalence is an equivalence relation)**.**
*The binary relation $\simeq$ on types is an equivalence relation.*

*Proof.* Immediate from Lemma 11 (Reflexivity of Structural Equivalence), Lemma 12 (Symmetry of Structural Equivalence), and Lemma 13 (Transitivity of Structural Equivalence). □

**Lemma 15** (Subtyping obeys Structural Equivalence)**.**
*If $A' \leq A$ then $A' \simeq A$.*

*Proof.* By induction on the structure of the derivation of $A' \leq A$.

- **Case** $\mathsf{Unit} \leq \mathsf{Unit}$: By definition of structural equivalence, $\mathsf{Unit} \simeq \mathsf{Unit}$.
- **Case** $\dfrac{A'_1 \leq A_1 \qquad A'_2 \leq A_2 \qquad \delta' \leq \delta}{(A'_1\, \delta'\, A'_2) \leq (A_1\, \delta\, A_2)}$

| | |
|---|---|
| $A'_1 \leq A_1$ | Subderivation |
| $A'_2 \leq A_2$ | Subderivation |
| $A'_1 \simeq A_1$ | By the induction hypothesis |
| $A'_2 \simeq A_2$ | By the induction hypothesis |
| $A'_1\, \delta'\, A'_2 \simeq A_1\, \delta\, A_2$ | By definition of $\simeq$ |

- **Case** $\dfrac{A_1 \leq A'_1 \qquad A'_2 \leq A_2}{(A'_1 \to A'_2) \leq (A_1 \to A_2)}$

| | |
|---|---|
| $A_1 \leq A'_1$ | Subderivation |
| $A'_2 \leq A_2$ | Subderivation |
| $A_1 \simeq A'_1$ | By the induction hypothesis |
| $A'_1 \simeq A_1$ | By Lemma 12 (Symmetry of Structural Equivalence) |
| $A'_2 \simeq A_2$ | By the induction hypothesis |
| $A'_1 \to A'_2 \simeq A_1 \to A_2$ | By definition of $\simeq$ |

□

**Lemma 16** (Precision obeys Structural Equivalence)**.**
*If $A' \sqsubseteq A$ then $A' \simeq A$.*

*Proof.* By induction on the structure of the derivation of $A' \sqsubseteq A$. All cases are immediate by the induction hypothesis and the definition of structural equivalence. □

**Lemma 17** (Directed consistency obeys Structural Equivalence)**.**
*If $A \rightsquigarrow B$ then $A \simeq B$.*

*Proof.* It is given that $A \rightsquigarrow B$. By inversion on $\mathsf{DirConsU}$, there exist $A'$ and $B'$ such that $A' \sqsubseteq A$ and $A' \le B'$ and $B' \sqsubseteq B$. By Lemma 16 (Precision obeys Structural Equivalence), $A' \simeq A$ and $B' \simeq B$. By Lemma 11 (Reflexivity of Structural Equivalence), $A \simeq A'$. By Lemma 15 (Subtyping obeys Structural Equivalence), $A' \simeq B'$. Therefore, by Lemma 13 (Transitivity of Structural Equivalence), $A \simeq B$. ☐

### D.1.5 Decidability

In this section, we write $\mathcal{J}$ `decidable` in proofs to indicate that the associated judgment form $\mathcal{J}$ is decidable.

$\boxed{\delta' \le \delta}$ Sum $\delta'$ is a sub-sum of $\delta$

$$\frac{}{+^?_i \le +^?_i} \qquad \frac{}{+^?_i \le +^?} \qquad \frac{}{+^?_i \le +_i} \qquad \frac{}{+^?_i \le +^*_k} \qquad \frac{}{+^?_i \le +} \qquad \frac{}{+^? \le +^?} \qquad \frac{}{+^? \le +^*_i}$$

$$\frac{}{+^? \le +} \qquad \frac{}{+_i \le +_i} \qquad \frac{}{+_i \le +^*_i} \qquad \frac{}{+_i \le +} \qquad \frac{}{+^*_i \le +^*_i} \qquad \frac{}{+^*_i \le +} \qquad \frac{}{+ \le +}$$

**Figure 19.** Reflexive, transitive closure of source subsum

**Lemma 18** (Decidability of subsum)**.**
*Given $\delta'$ and $\delta$, the judgment $\delta' \le \delta$ is decidable.*

*Proof.* We present the reflexive, transitive closure of the subsum relation on source sums in Figure 19. We can view this relation as a finite set of ordered sums. Thus, the decidability of the subsum relation is equivalent to a membership check on this set. ☐

**Lemma 19** (Decidability of subtyping)**.**
*Given $A'$ and $A$, the judgment $A' \le A$ is decidable.*

*Proof.* By simultaneous induction on the structure of $A'$ and $A$.
Proceed by case analysis on the head constructors of $A'$ and $A$. Either they agree or they disagree.
If they disagree, then no rule can possibly derive $A' \le A$.
If they agree, then:

- **Case** $A' = \mathsf{Unit}$ and $A = \mathsf{Unit}$: By definition of subtyping, $\mathsf{Unit} \le \mathsf{Unit}$.
- **Case** $A' = A'_1\,\delta'\,A'_2$ and $A = A_1\,\delta\,A_2$:

$$\begin{array}{rl}
A'_1 \le A_1 \ \texttt{decidable} & \text{By the induction hypothesis} \\
A'_2 \le A_2 \ \texttt{decidable} & \text{By the induction hypothesis} \\
\delta' \le \delta \ \texttt{decidable} & \text{By Lemma 18 (Decidability of subsum)} \\
A'_1\,\delta'\,A'_2 \le A_1\,\delta\,A_2 \ \texttt{decidable} & \text{By decidability of premises}
\end{array}$$

- **Case** $A' = A'_1 \to A'_2$ and $A = A_1 \to A_2$:

$$\begin{array}{rl}
A_1 \le A'_1 \ \texttt{decidable} & \text{By the induction hypothesis} \\
A'_2 \le A_2 \ \texttt{decidable} & \text{By the induction hypothesis} \\
A'_1 \to A'_2 \le A_1 \to A_2 \ \texttt{decidable} & \text{By decidability of premises}
\end{array}$$

☐

$\boxed{\delta' \sqsubseteq \delta}$ Sum $\delta'$ is more precise than $\delta$

$$\frac{}{+_i \sqsubseteq +_i} \qquad \frac{}{+_i \sqsubseteq +^?_i} \qquad \frac{}{+_i \sqsubseteq +^*_i} \qquad \frac{}{+_i \sqsubseteq +^?} \qquad \frac{}{+ \sqsubseteq +} \qquad \frac{}{+ \sqsubseteq +^?}$$

$$\frac{}{+^?_i \sqsubseteq +^?_i} \qquad \frac{}{+^?_i \sqsubseteq +^?} \qquad \frac{}{+^*_i \sqsubseteq +^*_i} \qquad \frac{}{+^*_i \sqsubseteq +^?} \qquad \frac{}{+^? \sqsubseteq +^?}$$

**Figure 20.** Reflexive, transitive closure of precision on sums

**Lemma 20** (Decidability of precision on sums)**.**
*Given $\delta'$ and $\delta$, the judgment $\delta' \sqsubseteq \delta$ is decidable.*

*Proof.* We present the reflexive, transitive closure of the precision relation on source sums in Figure 20. We could view this relation as a finite set of ordered sums. Thus, the decidability of the precision relation is equivalent to a membership check on this set. Therefore, given $\delta'$ and $\delta$, check whether or not $(\delta', \delta) \in \sqsubseteq$. ☐

**Lemma 21** (Decidability of precision on types)**.**
*Given $A'$ and $A$, the judgment $A' \sqsubseteq A$ is decidable.*

*Proof.* By simultaneous induction on the structure of $A'$ and $A$.

Proceed by case analysis on the head constructors of $A'$ and $A$. Either they agree or they disagree.

If they disagree, then no rule can possibly derive $A' \sqsubseteq A$.

If they agree, then:

- **Case** $A' = \mathsf{Unit}$ and $A = \mathsf{Unit}$: By definition of precision, $\mathsf{Unit} \sqsubseteq \mathsf{Unit}$ and therefore derivablity is decidable.
- **Case** $A' = A_1' \, \delta' \, A_2'$ and $A = A_1 \, \delta \, A_2$:

| | |
|---|---|
| $A_1' \sqsubseteq A_1$ decidable | By the induction hypothesis |
| $A_2' \sqsubseteq A_2$ decidable | By the induction hypothesis |
| $\delta' \sqsubseteq \delta$ decidable | By Lemma 20 (Decidability of precision on sums) |
| $A_1' \, \delta' \, A_2' \sqsubseteq A_1 \, \delta \, A_2$ decidable | By decidability of premises |

- **Case** $A' = A_1' \to A_2'$ and $A = A_1 \to A_2$:

| | |
|---|---|
| $A_1' \sqsubseteq A_1$ decidable | By the induction hypothesis |
| $A_2' \sqsubseteq A_2$ decidable | By the induction hypothesis |
| $A_1' \to A_2' \sqsubseteq A_1 \to A_2$ decidable | By decidability of premises |

□

**Lemma 22** (Decidability of directed consistency)**.**
*Given $A'$ and $B'$, the relation $A' \rightsquigarrow B'$ is decidable.*

*Proof.* We have $A' \rightsquigarrow B'$ if and only if there exist $A$ and $B$ such that $A \sqsubseteq A'$ and $A \leq B$ and $B \sqsubseteq B'$. We are given $A'$; there are only finitely many types such that $A \sqsubseteq A'$. Each such $A$ has only finitely many supertypes, that is, types $B$ such that $A \leq B$. Since these two relations are decidable, $A' \rightsquigarrow B'$ is decidable. □

**Theorem 1** (Decidability of bidirectional typing)**.**

1. *Given $\Gamma$, $e$ and $A$, the judgment $\Gamma \vdash e \Leftarrow A$ is decidable.*
2. *Given $\Gamma$ and $e$, the judgment $\Gamma \vdash e \Rightarrow A$ is decidable.*

*Proof.* By lexicographic induction on (1) the expression $e$, then on (2) the judgment form, with $\Rightarrow$ smaller than $\Leftarrow$.

In most rules, the expression gets smaller in all the premises: SynAnno, Chk→Intro, Syn→Elim, ChkSumIntro, ChkSumElim1, and ChkSumElim2.

In ChkCSub, the premise types the same expression but is a synthesizing judgment, which is smaller under our induction measure. By Lemma 22, the second premise of ChkCSub is decidable. □

#### D.1.6 Equivalence of type assignment and bidirectional system

**Lemma 23** (All sums below $+$)**.**
*For all source sums $\delta$, it is the case that $\delta \leq +$.*

*Proof.* By case analysis on $\delta$.

- **Case** $\delta = +_i^*$: By the definition of subtyping, $+_i^* \leq +$.
- **Case** $\delta = +_i$: By the definition of subtyping, $+_i \leq +_i^*$. By the previous case, $+_i^* \leq +$. By the transitivity of subtyping, $+_i \leq +$.
- **Case** $\delta = +_i^?$: By the definition of subtyping, $+_i^? \leq +_i$. By the previous case, $+_i \leq +$. By the transitivity of subtyping, $+_i^? \leq +$.
- **Case** $\delta = +^?$: By the definition of subtyping, $+^? \leq +_i^*$. By the definition of subtyping, $+_i^* \leq +$. By the transitivity of subtyping, $+^? \leq +$.
- **Case** $\delta = +$: By the reflexivity of subtyping, $+ \leq +$. □

**Lemma 24** ($\Rrightarrow$ implies subsum)**.**
*If $\delta' \Rrightarrow \delta$ then $\delta' \leq \delta$.*

*Proof.* By case analysis on $\delta' \Rrightarrow \delta$.

- **Case** $+_i^? \Rrightarrow +_i^*$: By definition of subtyping, $+_i^? \leq +_i$. By definition of subtyping, $+_i \leq +_i^*$. By transitivity of subtyping, $+_i^? \leq +_i^*$.
- **Case** $+_i \Rrightarrow +_i^*$: By definition of subtyping, $+_i \leq +_i^*$.
- **Case** $+^? \Rrightarrow +_i^*$: By definition of subtyping, $+^? \leq +_i^*$.
- **Case** $+_i^* \Rrightarrow +_i^*$: By reflexivity of subtyping, $+_i^* \leq +_i^*$.
- **Case** $\delta' \Rrightarrow +$: By Lemma 23 (All sums below $+$), $\delta' \leq +$. □

**Theorem 2** (Bidirectional soundness)**.**
*If $\Gamma \vdash e \Leftarrow A$ or $\Gamma \vdash e \Rightarrow A$ then $\Gamma \vdash e : A$.*

*Proof.* By induction on the structure of the given derivation.

- **Case** SynVar: Apply rule SVar.
- **Case** ChkCSub: Use the induction hypothesis and apply rule SCSub.

- **Case** SynAnno: Use the induction hypothesis, and apply rule SAnno.
- **Case** ChkUnitIntro: Apply rule SUnitIntro.
- **Case** $\dfrac{\Gamma \vdash e_0 \Leftarrow A_i \qquad +^?_i \leq \delta}{\Gamma \vdash \mathtt{inj}_i\, e_0 \Leftarrow (A_1\, \delta\, A_2)}$ ChkSumIntro

| | |
|---|---|
| $\Gamma \vdash e_0 \Leftarrow A_i$ | Subderivation |
| $\Gamma \vdash e_0 : A_i$ | By the induction hypothesis |
| $\Gamma \vdash \mathtt{inj}_i\, e_0 : (A_1 +^?_i A_2)$ | By rule SSumIntro |
| $A_1 \leq A_1$ | By Lemma 2 (Reflexivity of subtyping) |
| $A_2 \leq A_2$ | By Lemma 2 (Reflexivity of subtyping) |
| $+^?_i \leq \delta$ | Subderivation |
| $A_1 +^?_i A_2 \leq A_1\, \delta\, A_2$ | By definition of $\leq$ |
| $A_1 +^?_i A_2 \rightsquigarrow A_1\, \delta\, A_2$ | By Lemma 8 (Subtyping obeys directed consistency) |
| $\Gamma \vdash \mathtt{inj}_i\, e_0 : (A_1\, \delta\, A_2)$ | By rule SCSub |

- **Case** $\dfrac{\begin{array}{l}\Gamma \vdash e_0 \Rightarrow (A_1\, \delta\, A_2) \\ \delta \Rrightarrow +^*_i \qquad\qquad \Gamma, x : A_i \vdash e_i \Leftarrow A\end{array}}{\Gamma \vdash \mathtt{case}(e_0, \mathtt{inj}_i\, x.e_i) \Leftarrow A}$ ChkSumElim1

| | |
|---|---|
| $\Gamma \vdash e_0 \Rightarrow (A_1\, \delta\, A_2)$ | Subderivation |
| $\Gamma \vdash e_0 : (A_1\, \delta\, A_2)$ | By the induction hypothesis |
| $\delta \Rrightarrow +^*_i$ | Subderivation |
| $\delta \leq +^*_i$ | By Lemma 24 ($\Rrightarrow$ implies subsum) |
| $A_1 \leq A_1$ | By Lemma 2 (Reflexivity of subtyping) |
| $A_2 \leq A_2$ | By Lemma 2 (Reflexivity of subtyping) |
| $A_1\, \delta\, A_2 \leq A_1 +^*_i A_2$ | By definition of $\leq$ |
| $A_1\, \delta\, A_2 \rightsquigarrow A_1 +^*_i A_2$ | By Lemma 8 (Subtyping obeys directed consistency) |
| $\Gamma \vdash e_0 : (A_1 +^*_i A_2)$ | By rule SCSub |
| $\Gamma, x : A_i \vdash e_i \Leftarrow A$ | Subderivation |
| $\Gamma, x : A_i \vdash e_i : A$ | By the induction hypothesis |
| $\Gamma \vdash \mathtt{case}(e_0, \mathtt{inj}_i\, x.e_i) : A$ | By rule ChkSumElim1 |

- **Case** ChkSumElim2: Similar to the ChkSumElim1 case, hence omitted.
- **Case** Chk$\rightarrow$Intro: Use the induction hypothesis, and apply rule S$\rightarrow$Intro.
- **Case** Syn$\rightarrow$Elim: Use the induction hypothesis, and apply rule S$\rightarrow$Elim. □

**Lemma 25** (Reflexivity of annotation equivalence)**.** *For all expressions* $e$, $e =: e$.

*Proof.* By induction on the structure of $e$.
All cases either hold directly by definition or by first using the induction hypothesis. □

**Lemma 26** (Synthesis also checks)**.** *If* $\Gamma \vdash e \Rightarrow A$ *then* $\Gamma \vdash e \Leftarrow A$.

*Proof.* Apply rule ChkCSub as $A \rightsquigarrow A$ holds by Lemma 5 (Reflexivity of precision). □

**Theorem 3** (Annotatability)**.**
*If* $\Gamma \vdash e : A$ *then there exist* $e'$ *and* $e''$ *such that (1)* $\Gamma \vdash e' \Leftarrow A$ *where* $e =: e'$*, and (2)* $\Gamma \vdash e'' \Rightarrow A$ *where* $e =: e''$.

*Proof.* By induction on the structure of the derivation of $\Gamma \vdash e : A$.

- **Case** $\dfrac{\Gamma(x) = A}{\Gamma \vdash x : A}$ SVar

| | | |
|---|---|---|
| | $\Gamma(x) = A$ | Premise |
| ☞ | $\Gamma \vdash x \Rightarrow A$ | By rule SynVar |
| ☞ | $\Gamma \vdash x \Leftarrow A$ | By Lemma 26 (Synthesis also checks) |
| ☞ | $x =: x$ | By definition of $=:$ |

- **Case** $\dfrac{\Gamma \vdash e : A' \qquad A' \rightsquigarrow A}{\Gamma \vdash e : A}$ SCSub

| | | |
|---|---|---|
| | $\Gamma \vdash e : A'$ | Subderivation |
| | $\Gamma \vdash e' \Rightarrow A'$ | By the induction hypothesis |
| ☞ | $e =: e'$ | '' |
| | $A' \rightsquigarrow A$ | Subderivation |
| ☞ | $\Gamma \vdash e' \Leftarrow A$ | By rule ChkCSub |
| ☞ | $\Gamma \vdash (e' :: A) \Rightarrow A$ | By rule SynAnno |
| ☞ | $e =: (e' :: A)$ | By definition of $=:$ |

- **Case** $\dfrac{\Gamma \vdash e_0 : A}{\Gamma \vdash (e_0 :: A) : A}$ SAnno

| | | |
|---|---|---|
| | $\Gamma \vdash e_0 : A$ | Subderivation |
| | $\Gamma \vdash e_0' \Leftarrow A$ | By the induction hypothesis |
| | $e_0 =: e_0'$ | '' |
| ☞ | $\Gamma \vdash (e_0' :: A) \Rightarrow A$ | By rule SynAnno |
| ☞ | $\Gamma \vdash (e_0' :: A) \Leftarrow A$ | By Lemma 26 (Synthesis also checks) |
| ☞ | $e_0 =: (e_0' :: A)$ | By definition of $=:$ |

- **Case** $\dfrac{}{\Gamma \vdash () : \mathsf{Unit}}$ SUnitIntro

| | | |
|---|---|---|
| ☞ | $\Gamma \vdash () \Leftarrow \mathsf{Unit}$ | By rule ChkUnitIntro |
| ☞ | $\Gamma \vdash (() :: \mathsf{Unit}) \Rightarrow \mathsf{Unit}$ | By rule SynAnno |
| ☞ | $() =: ()$ | By definition of $=:$ |
| ☞ | $() =: (() :: \mathsf{Unit})$ | By definition of $=:$ |

- **Case** $\dfrac{\Gamma \vdash e_0 : A_i}{\Gamma \vdash \mathbf{inj}_i\, e_0 : (A_1 +_i^? A_2)}$ SSumIntro

| | | |
|---|---|---|
| | $\Gamma \vdash e_0 : A_i$ | Subderivation |
| | $\Gamma \vdash e_0' \Leftarrow A_i$ | By the induction hypothesis |
| | $e_0 =: e_0'$ | '' |
| | $+_i^? \leq +_i^?$ | By definition of $\leq$ |
| ☞ | $\Gamma \vdash \mathbf{inj}_i\, e_0' \Leftarrow (A_1 +_i^? A_2)$ | By rule ChkSumIntro |
| ☞ | $\Gamma \vdash (\mathbf{inj}_i\, e_0' :: A_1 +_i^? A_2) \Rightarrow (A_1 +_i^? A_2)$ | By rule SynAnno |
| ☞ | $\mathbf{inj}_i\, e_0 =: \mathbf{inj}_i\, e_0'$ | By definition of $=:$ |
| ☞ | $\mathbf{inj}_i\, e_0 =: (\mathbf{inj}_i\, e_0' :: A_1 +_i^? A_2)$ | By definition of $=:$ |

- **Case** $\dfrac{\Gamma \vdash e_0 : A_1 +_i^* A_2 \qquad \Gamma, x : A_i \vdash e_i : A}{\Gamma \vdash \mathsf{case}(e_0, \mathbf{inj}_i\, x.e_i) : A}$ SSumElim1

| | | |
|---|---|---|
| | $\Gamma \vdash e_0 : A_1 +_i^* A_2$ | Subderivation |
| | $\Gamma \vdash e_0' \Rightarrow A_1 +_i^* A_2$ | By the induction hypothesis |
| | $e_0 =: e_0'$ | '' |
| | $\Gamma, x : A_i \vdash e_i : A$ | Subderivation |
| | $\Gamma, x : A_i \vdash e_i' \Leftarrow A$ | By the induction hypothesis |
| | $e_i =: e_i'$ | '' |
| | $+_i^* \Rrightarrow +_i^*$ | By definition of $\Rrightarrow$ |
| ☞ | $\Gamma \vdash \mathsf{case}(e_0', \mathbf{inj}_i\, x.e_i') \Leftarrow A$ | By rule ChkSumElim1 |
| ☞ | $\Gamma \vdash (\mathsf{case}(e_0', \mathbf{inj}_i\, x.e_i') :: A) \Rightarrow A$ | By rule SynAnno |
| ☞ | $\mathsf{case}(e_0, \mathbf{inj}_i\, x.e_i) =: \mathsf{case}(e_0', \mathbf{inj}_i\, x.e_i')$ | By definition of $=:$ |
| ☞ | $\mathsf{case}(e_0, \mathbf{inj}_i\, x.e_i) =: (\mathsf{case}(e_0', \mathbf{inj}_i\, x.e_i') :: A)$ | By definition of $=:$ |

- **Case** SSumElim2: Similar to the SSumElim1 case, hence omitted.

- **Case** $\dfrac{\Gamma, x : A_1 \vdash e_0 : A_2}{\Gamma \vdash \lambda x.\, e_0 : A_1 \to A_2}$ S$\to$Intro

$\Gamma, x : A_1 \vdash e_0 : A_2$ Subderivation
$\Gamma, x : A_1 \vdash e_0' \Leftarrow A_2$ By the induction hypothesis
$e_0 =: e_0'$ $''$

☞ $\Gamma \vdash \lambda x.\, e_0' \Leftarrow (A_1 \to A_2)$ By rule Chk$\to$Intro
☞ $\Gamma \vdash (\lambda x.\, e_0' :: A_1 \to A_2) \Rightarrow (A_1 \to A_2)$ By rule SynAnno
☞ $\lambda x.\, e_0 =: \lambda x.\, e_0'$ By definition of $=:$
☞ $\lambda x.\, e_0 =: (\lambda x.\, e_0' :: A_1 \to A_2)$ By definition of $=:$

- **Case** $\dfrac{\Gamma \vdash e_1 : A_1 \to A_2 \qquad \Gamma \vdash e_2 : A_1}{\Gamma \vdash e_1\, e_2 : A_2}$ S$\to$Elim

$\Gamma \vdash e_1 : A_1 \to A_2$ Subderivation
$\Gamma \vdash e_1' \Rightarrow A_1 \to A_2$ By the induction hypothesis
$e_1 =: e_1'$ $''$

$\Gamma \vdash e_2 : A_1$ Subderivation
$\Gamma \vdash e_2' \Leftarrow A_1$ By the induction hypothesis
$e_2 =: e_2'$ $''$

☞ $\Gamma \vdash e_1'\, e_2' \Rightarrow A_2$ By rule Syn$\to$Elim
☞ $\Gamma \vdash e_1'\, e_2' \Leftarrow A_2$ By Lemma 26 (Synthesis also checks)
☞ $e_1\, e_2 =: e_1'\, e_2'$ By definition of $=:$ □

### D.2 Typability under varying precision

**Lemma 27** (Pointwise precision preserves domain).
*If* $\Gamma' \sqsubseteq \Gamma$ *then* $\mathsf{dom}(\Gamma') = \mathsf{dom}(\Gamma)$.

*Proof.* By induction on the structure of $\Gamma' \sqsubseteq \Gamma$. □

**Lemma 28** (Context strengthening).
*If* $\Gamma, y : A' \vdash e : A_0$ *and* $A \sqsubseteq A'$ *then* $\Gamma, y : A \vdash e : A_0$.

*Proof.* By induction on the structure of the derivation of $\Gamma, y : A' \vdash e : A_0$.

- **Case** $\dfrac{(\Gamma, y : A')(e) = A_0}{\Gamma, y : A' \vdash e : A_0}$ SVar

  Either $e = y$, or $e \neq y$.
  In the first case:

  $(\Gamma, y : A')(y) = A_0$ Premise
  $A' = A_0$ By definition
  $\Gamma, y : A \vdash y : A$ By rule SVar
  $A \sqsubseteq A'$ Given
  $A \rightsquigarrow A'$ By Lemma 9 (Loss in precision obeys directed consistency)
  $\Gamma, y : A \vdash y : A'$ By rule SCSub
  $\Gamma, y : A \vdash e : A_0$ By above equalities

  In the second case:

  $\Gamma, y : A \vdash e : A_0$ By rule SVar

- **Case** SCSub: Use the induction hypothesis and apply rule SCSub.
- **Case** SUnitIntro: Immediate from rule SUnitIntro.
- **Case** SSumIntro: Use the induction hypothesis and apply rule SSumIntro.
- **Case** SSumElim1: Use the induction hypothesis and apply rule SSumElim1.
- **Case** SSumElim2: Use the induction hypothesis and apply rule SSumElim2.
- **Case** S$\to$Intro: Use the induction hypothesis and apply rule S$\to$Intro.
- **Case** S$\to$Elim: Use the induction hypothesis and apply rule S$\to$Elim. □

**Corollary 29.**
*If* $\Gamma' \vdash e : A$ *and* $\Gamma \sqsubseteq \Gamma'$ *then* $\Gamma \vdash e : A$.

*Proof.* By induction on the number of variables $x$ such that $x \in \mathsf{dom}(\Gamma')$ but $\Gamma'(x) \neq \Gamma(x)$.

Note that we don't impose $x \in \mathsf{dom}(\Gamma)$ as $\mathsf{dom}(\Gamma) = \mathsf{dom}(\Gamma')$ by Lemma 27 (Pointwise precision preserves domain).

If $\Gamma'(x) = \Gamma(x)$ for all $x \in \mathsf{dom}(\Gamma')$, then $\Gamma = \Gamma'$ so we already have the result.

Otherwise, use the induction hypothesis, and apply Lemma 28 (Context strengthening). □

**Lemma 30** (Relating $+_i^?$-subsum and precision)**.**
*If $+_i^? \leq \delta'$ and $\delta' \sqsubseteq \delta$ then $+_i^? \leq \delta$.*

*Proof.* Proceed by case analysis on $+_i^? \leq \delta'$.

- **Case** $+_i^? \leq +_i^?$: From the definition of precision, either $\delta = +_i^?$ or $\delta = +^?$. In both cases, there exists a derivation for $+_i^? \leq \delta$.
- **Case** $+_i^? \leq +_i$: From the definition of precision, either $\delta = +_i$, $\delta = +_i^?$, $\delta = +_i^*$ or $\delta = +^?$. In all cases, there exists a derivation for $+_i^? \leq \delta$.
- **Case** $+_i^? \leq +^?$: From the definition of precision, $\delta = +^?$. We are given a derivation for $+_i^? \leq +^?$.
- **Case** $+_i^? \leq +_k^*$: From the definition of precision, either $\delta = +_k^*$ or $\delta = +^?$. In both cases, there exists a derivation for $+_i^? \leq \delta$.
- **Case** $+_i^? \leq +$: From the definition of precision, either $\delta = +$ or $\delta = +^?$. In both cases, there exists a derivation for $+_i^? \leq \delta$. □

**Lemma 31** (Bidirectional sum precision)**.**
*If $\delta' \Rrightarrow \delta_1$ and $\delta' \sqsubseteq \delta$ then $\delta \Rrightarrow \delta_1$.*

*Proof.* Proceed by case analysis on $\delta' \Rrightarrow \delta_1$.

- **Case** $+_i^? \Rrightarrow +_i^*$: From the definition of precision, either $\delta = +_i^?$ or $\delta = +^?$. In both cases, there exists a derivation for $\delta \Rrightarrow +_i^*$.
- **Case** $+_i \Rrightarrow +_i^*$: From the definition of precision, either $\delta = +_i$, $\delta = +_i^?$, $\delta = +_i^*$, or $\delta = +^?$. In all cases, there exists a derivations for $\delta \Rrightarrow +_i^*$.
- **Case** $+^? \Rrightarrow +_i^*$: From the definition of precision, $\delta = +^?$. We are given a derivation for $+^? \Rrightarrow +_i^*$.
- **Case** $+_i^* \Rrightarrow +_i^*$: From the definition of precision, either $\delta = +_i^*$ or $\delta = +^?$. In both cases, there exists a derivation for $\delta \Rrightarrow +_i^*$.
- **Case** $\delta' \Rrightarrow +$: There exists a derivation for $\delta \Rrightarrow +$ for all $\delta$. □

**Theorem 4** (Varying precision of bidirectional typing)**.**

1. *If $\Gamma' \vdash e' \Leftarrow A'$ and $e' \sqsubseteq e$ and $\Gamma' \sqsubseteq \Gamma$ and $A' \sqsubseteq A$ then $\Gamma \vdash e \Leftarrow A$.*
2. *If $\Gamma' \vdash e' \Rightarrow A'$ and $e' \sqsubseteq e$ and $\Gamma' \sqsubseteq \Gamma$ then there exists $A$ such that $\Gamma \vdash e \Rightarrow A$ and $A' \sqsubseteq A$.*

*Proof.* By induction on the structure of the given derivation.

1. By case analysis on the rule concluding $\Gamma' \vdash e' \Leftarrow A'$.

   - **Case**
   $$\frac{}{\Gamma' \vdash \underbrace{()}_{e'} \Leftarrow \underbrace{\mathsf{Unit}}_{A'}}\ \mathsf{ChkUnitIntro}$$

     | | |
     |---|---|
     | $() \sqsubseteq e$ | Given |
     | $e = ()$ | From definition of $\sqsubseteq$ |
     | $\mathsf{Unit} \sqsubseteq A$ | Given |
     | $A = \mathsf{Unit}$ | By Lemma 4 (Precision inversion) |
     | $\Gamma \vdash e \Leftarrow \mathsf{Unit}$ | By rule ChkUnitIntro |

   - **Case**
   $$\frac{\Gamma' \vdash e' \Rightarrow A'_0 \qquad A'_0 \rightsquigarrow A'}{\Gamma' \vdash e' \Leftarrow A'}\ \mathsf{ChkCSub}$$

$$
\begin{array}{ll}
\Gamma' \vdash e' \Rightarrow A'_0 & \text{Subderivation} \\
e' \sqsubseteq e & \text{Given} \\
\Gamma' \sqsubseteq \Gamma & \text{Given} \\
\Gamma \vdash e \Rightarrow A_0 & \text{By the induction hypothesis} \\
A'_0 \sqsubseteq A_0 & '' \\
\\
A'_0 \rightsquigarrow A' & \text{Subderivation} \\
B'_0 \sqsubseteq A'_0 & \text{By inversion on } \mathsf{DirConsU} \\
B'_0 \leq B' & '' \\
B' \sqsubseteq A' & '' \\
\\
B'_0 \sqsubseteq A_0 & \text{By Lemma 6 (Transitivity of precision)} \\
A' \sqsubseteq A & \text{Given} \\
B' \sqsubseteq A & \text{By Lemma 6 (Transitivity of precision)} \\
A_0 \rightsquigarrow A & \text{By rule } \mathsf{DirConsU} \\
\\
\Gamma \vdash e \Leftarrow A & \text{By rule } \mathsf{ChkCSub}
\end{array}
$$

- **Case** $\dfrac{\Gamma', x : A'_1 \vdash e'_0 \Leftarrow A'_2}{\Gamma' \vdash \underbrace{\lambda x.\, e'_0}_{e'} \Leftarrow \underbrace{A'_1 \to A'_2}_{A'}}\ \mathsf{Chk{\to}Intro}$

$$
\begin{array}{ll}
\lambda x.\, e'_0 \sqsubseteq e & \text{Given} \\
e = \lambda x.\, e_0 & \text{From definition of } \sqsubseteq \\
e'_0 \sqsubseteq e_0 & '' \\
\\
A'_1 \to A'_2 \sqsubseteq A & \text{Given} \\
A = A_1 \to A_2 & \text{By Lemma 4 (Precision inversion)} \\
A'_1 \sqsubseteq A_1 & '' \\
A'_2 \sqsubseteq A_2 & '' \\
\\
\Gamma' \sqsubseteq \Gamma & \text{Given} \\
\Gamma', x : A'_1 \sqsubseteq \Gamma, x : A_1 & \text{By definition of } \sqsubseteq \\
\Gamma', x : A'_1 \vdash e'_0 \Leftarrow A'_2 & \text{Subderivation} \\
\Gamma, x : A_1 \vdash e_0 \Leftarrow A_2 & \text{By the induction hypothesis} \\
\Gamma \vdash \lambda x.\, e_0 \Leftarrow A_1 \to A_2 & \text{By rule } \mathsf{Chk{\to}Intro}
\end{array}
$$

- **Case** $\dfrac{\Gamma' \vdash e'_0 \Leftarrow A'_i \qquad +^?_i \leq \delta'}{\Gamma' \vdash \underbrace{\mathtt{inj}_i\, e'_0}_{e'} \Leftarrow \underbrace{A'_1\, \delta'\, A'_2}_{A'}}\ \mathsf{ChkSumIntro}$

$$
\begin{array}{ll}
\mathtt{inj}_i\, e'_0 \sqsubseteq e & \text{Given} \\
e = \mathtt{inj}_i\, e_0 & \text{From definition of } \sqsubseteq \\
e'_0 \sqsubseteq e_0 & '' \\
\\
A'_1\, \delta'\, A'_2 \sqsubseteq A & \text{Given} \\
A = A_1\, \delta\, A_2 & \text{By Lemma 4 (Precision inversion)} \\
A'_i \sqsubseteq A_i & '' \\
\delta' \sqsubseteq \delta & '' \\
\\
\Gamma' \vdash e'_0 \Leftarrow A'_i & \text{Subderivation} \\
\Gamma' \sqsubseteq \Gamma & \text{Given} \\
\Gamma \vdash e_0 \Leftarrow A_i & \text{By the induction hypothesis} \\
\\
+^?_i \leq \delta' & \text{Subderivation} \\
+^?_i \leq \delta & \text{By Lemma 30 (Relating } +^?_i\text{-subsum and precision)} \\
\Gamma \vdash \mathtt{inj}_i\, e_0 \Leftarrow (A_1\, \delta\, A_2) & \text{By rule } \mathsf{ChkSumIntro}
\end{array}
$$

- **Case** $\dfrac{\begin{array}{l}\Gamma' \vdash e'_0 \Rightarrow A'_1\, \delta'\, A'_2 \\ \delta' \Rrightarrow +^*_i \qquad\qquad \Gamma', x : A'_i \vdash e'_i \Leftarrow A'\end{array}}{\Gamma' \vdash \underbrace{\mathtt{case}(e'_0, \mathtt{inj}_i\, x.e'_i)}_{e'} \Leftarrow A'}\ \mathsf{ChkSumElim1}$

$$\begin{array}{rll}
e' \sqsubseteq e & & \text{Given} \\
e = \mathsf{case}(e_0, \mathtt{inj}_i\, x.e_i) & & \text{From definition of } \sqsubseteq \\
e'_0 \sqsubseteq e_0 & & '' \\
e'_i \sqsubseteq e_i & & '' \\[1ex]
\Gamma' \vdash e'_0 \Rightarrow A'_1\, \delta'\, A'_2 & & \text{Subderivation} \\
\Gamma' \sqsubseteq \Gamma & & \text{Given} \\
\Gamma \vdash e_0 \Rightarrow A_1\, \delta\, A_2 & & \text{By the induction hypothesis} \\
A'_1\, \delta'\, A'_2 \sqsubseteq A_1\, \delta\, A_2 & & '' \\
A'_i \sqsubseteq A_i & & \text{From definition of } \sqsubseteq \\
\delta' \sqsubseteq \delta & & '' \\[1ex]
\delta' \Rrightarrow +^*_i & & \text{Subderivation} \\
\delta \Rrightarrow +^*_i & & \text{By Lemma 31 (Bidirectional sum precision)} \\[1ex]
\Gamma', x : A'_i \sqsubseteq \Gamma, x : A_i & & \text{By definition of } \sqsubseteq \\
A' \sqsubseteq A & & \text{Given} \\
\Gamma', x : A'_i \vdash e'_i \Leftarrow A' & & \text{Subderivation} \\
\Gamma, x : A_i \vdash e_i \Leftarrow A & & \text{By the induction hypothesis} \\
\Gamma \vdash \mathsf{case}(e_0, \mathtt{inj}_i\, x.e_i) \Leftarrow A & & \text{By rule } \mathsf{ChkSumElim1}
\end{array}$$

- **Case** $$\dfrac{\Gamma' \vdash e'_0 \Rightarrow A'_1\, \delta'\, A'_2 \quad \Gamma', x_1 : A'_1 \vdash e'_1 \Leftarrow A' \quad \delta' \Rrightarrow + \quad \Gamma', x_2 : A'_2 \vdash e'_2 \Leftarrow A'}{\Gamma' \vdash \underbrace{\mathsf{case}(e'_0, \mathtt{inj}_1\, x_1.e'_1, \mathtt{inj}_2\, x_2.e'_2)}_{e'} \Leftarrow A'}\ \mathsf{ChkSumElim2}$$

$$\begin{array}{rl}
e' \sqsubseteq e & \text{Given} \\
e = \mathsf{case}(e_0, \mathtt{inj}_1\, x_1.e_1, \mathtt{inj}_2\, x_2.e_2) & \text{From definition of } \sqsubseteq \\
e'_0 \sqsubseteq e_0 & '' \\
e'_1 \sqsubseteq e_1 & '' \\
e'_2 \sqsubseteq e_2 & '' \\[1ex]
\Gamma' \vdash e'_0 \Rightarrow A'_1\, \delta'\, A'_2 & \text{Subderivation} \\
\Gamma' \sqsubseteq \Gamma & \text{Given} \\
\Gamma \vdash e_0 \Rightarrow A_1\, \delta\, A_2 & \text{By the induction hypothesis} \\
A'_1\, \delta'\, A'_2 \sqsubseteq A_1\, \delta\, A_2 & '' \\
A'_1 \sqsubseteq A_1 & \text{From definition of } \sqsubseteq \\
A'_2 \sqsubseteq A_2 & '' \\
\delta' \sqsubseteq \delta & ''
\end{array}$$

$$\begin{array}{rl}
\delta' \Rrightarrow + & \text{Subderivation} \\
\delta \Rrightarrow + & \text{By Lemma 31 (Bidirectional sum precision)} \\
A' \sqsubseteq A & \text{Given} \\[1ex]
\Gamma', x_1 : A'_1 \sqsubseteq \Gamma, x_1 : A_1 & \text{By definition of } \sqsubseteq \\
\Gamma', x_1 : A'_1 \vdash e'_1 \Leftarrow A' & \text{Subderivation} \\
\Gamma, x_1 : A_1 \vdash e_1 \Leftarrow A & \text{By the induction hypothesis} \\[1ex]
\Gamma', x_2 : A'_2 \sqsubseteq \Gamma, x_2 : A_2 & \text{By definition of } \sqsubseteq \\
\Gamma', x_2 : A'_2 \vdash e'_2 \Leftarrow A' & \text{Subderivation} \\
\Gamma, x_2 : A_2 \vdash e_2 \Leftarrow A & \text{By the induction hypothesis}
\end{array}$$

$$\Gamma \vdash \mathsf{case}(e_0, \mathtt{inj}_1\, x_1.e_1, \mathtt{inj}_2\, x_2.e_2) \Leftarrow A \qquad \text{By rule } \mathsf{ChkSumElim2}$$

2. By case analysis on the rule concluding $\Gamma' \vdash e' \Rightarrow A'$.

- **Case** $$\dfrac{\Gamma'(x) = A'}{\Gamma' \vdash \underbrace{x}_{e'} \Rightarrow A'}\ \mathsf{SynVar}$$

  Let $A = \Gamma(x)$.

$$
\begin{array}{lll}
& x \sqsubseteq e & \text{Given} \\
& e = x & \text{From definition of } \sqsubseteq \\[6pt]
& \Gamma'(x) = A' & \text{Premise} \\
& \Gamma' \sqsubseteq \Gamma & \text{Given} \\
& \Gamma'(x) \sqsubseteq \Gamma(x) & \text{By definition of } \sqsubseteq \text{ on contexts} \\
☞ & A' \sqsubseteq A & \text{Equivalent} \\
☞ & \Gamma \vdash x \Rightarrow A & \text{By rule SynVar}
\end{array}
$$

- **Case** $\dfrac{\Gamma' \vdash e'_0 \Leftarrow A'}{\Gamma' \vdash \underbrace{(e'_0 :: A')}_{e'} \Rightarrow A'}$ SynAnno

$$
\begin{array}{lll}
& (e'_0 :: A') \sqsubseteq e & \text{Given} \\
& e = (e_0 :: A_0) & \text{From definition of } \sqsubseteq \\
& e'_0 \sqsubseteq e_0 & '' \\
☞ & A' \sqsubseteq A & '' \\[6pt]
& \Gamma' \vdash e'_0 \Leftarrow A' & \text{Subderivation} \\
& \Gamma' \sqsubseteq \Gamma & \text{Given} \\
& \Gamma \vdash e_0 \Leftarrow A & \text{By the induction hypothesis} \\[6pt]
☞ & \Gamma \vdash (e_0 :: A) \Rightarrow A & \text{By rule SynAnno}
\end{array}
$$

- **Case** $\dfrac{\Gamma' \vdash e'_1 \Rightarrow A'_0 \to A' \qquad \Gamma' \vdash e'_2 \Leftarrow A'_0}{\Gamma' \vdash \underbrace{e'_1\, e'_2}_{e'} \Rightarrow A'}$ Syn$\to$Elim

$$
\begin{array}{lll}
& e'_1\, e'_2 \sqsubseteq e & \text{Given} \\
& e = e_1\, e_2 & \text{From definition of } \sqsubseteq \\
& e'_1 \sqsubseteq e_1 & '' \\
& e'_2 \sqsubseteq e_2 & '' \\[6pt]
& \Gamma' \sqsubseteq \Gamma & \text{Given} \\
& \Gamma' \vdash e'_1 \Rightarrow A'_0 \to A' & \text{Subderivation} \\
& \Gamma \vdash e_1 \Rightarrow A_0 \to A & \text{By the induction hypothesis} \\
& A'_0 \to A' \sqsubseteq A_0 \to A & '' \\
& A'_0 \sqsubseteq A_0 & \text{From definition of } \sqsubseteq \\
☞ & A' \sqsubseteq A & '' \\[6pt]
& \Gamma' \vdash e'_2 \Leftarrow A'_0 & \text{Subderivation} \\
& \Gamma \vdash e_2 \Leftarrow A_0 & \text{By the induction hypothesis} \\
☞ & \Gamma \vdash e_1\, e_2 \Rightarrow A & \text{By rule Syn}\to\text{Elim}
\end{array}
$$

□

### D.3 Properties of the Static System

**Lemma 32** (Static looseness)**.**
*If* $+^?_i \leq \delta^S$ *then* $+_i \leq_S \delta^S$.

*Proof.* By case analysis on $+^?_i \leq \delta^S$.

- **Case** $+^?_i \leq +_i$: By definition of static subtyping $+_i \leq_S +_i$.
- **Case** $+^?_i \leq +$: By definition of static subtyping $+_i \leq_S +$. □

**Lemma 33** (Static looseness, II)**.**
*If* $\delta^S \Rrightarrow +^*_i$ *then* $\delta^S = +_i$.

*Proof.* By case analysis on $\delta^S \Rrightarrow +^*_i$.

- **Case** $+_i \Rrightarrow +^*_i$: It is the case that $\delta^S = +_i$. □

The following lemma states that static sums are the most precise and incomparable by the precision relation.

**Lemma 34** (Precision for static sums)**.**
*If* $\delta_1 \sqsubseteq \delta^S_2$ *then* $\delta_1 = \delta^S_2$.

*Proof.* Proceed by case analysis on $\delta^S_2$.

- **Case** $\delta_2^S = +_i$: By the definition of imprecision, $\delta_1 = +_i$ only.
- **Case** $\delta_2^S = +$: By the definition of imprecision, $\delta_1 = +$ only. □

**Lemma 35** (Precision for static types)**.**
*If* $A_1 \sqsubseteq A_2^S$ *then* $A_1 = A_2^S$.

*Proof.* By induction on the structure of $A_2^S$.

- **Case** $A_2^S = \mathsf{Unit}$: By the definition of imprecision, $A_1^S = \mathsf{Unit}$ only.
- **Case** $A_2^S = A_{12}^S\,\delta_2^S\,A_{22}^S$:

| | |
|---|---|
| $A_1 \sqsubseteq A_{12}^S\,\delta_2^S\,A_{22}^S$ | Given |
| $A_1 = A_{11}\,\delta_1\,A_{21}$ | From the definition of $\sqsubseteq$ |
| $A_{11} \sqsubseteq A_{12}^S$ | ″ |
| $A_{21} \sqsubseteq A_{22}^S$ | ″ |
| $\delta_1 \sqsubseteq \delta_2^S$ | ″ |
| $A_{11} = A_{12}^S$ | By the induction hypothesis |
| $A_{21} = A_{22}^S$ | By the induction hypothesis |
| $\delta_1 = \delta_2^S$ | By Lemma 34 (Precision for static sums) |
| $A_1 = A_2^S$ | By definition of $=$ |

- **Case** $A_2^S = A_{12}^S \to A_{22}^S$: Similar to the previous case. □

**Lemma 36** (Equivalence for static subsum)**.**
*1. If* $\delta_1^S \leq_S \delta_2^S$ *then* $\delta_1^S \leq \delta_2^S$.
*2. If* $\delta_1^S \leq \delta_2^S$ *then* $\delta_1^S \leq_S \delta_2^S$.

*Proof.*
1. By case analysis on $\delta_1^S \leq_S \delta_2^S$.
   - **Case** $\delta^S \leq_S \delta^S$: By definition of subtyping, $\delta^S \leq \delta^S$.
   - **Case** $+_i \leq_S +$: By definition of subtyping, $+_i \leq +_i^*$ and $+_i^* \leq +$. By transitivity of subtyping, $+_i \leq +$.
2. By case analysis on $\delta_1^S \leq \delta_2^S$.
   - **Case** $\delta^S \leq \delta^S$: By definition of static subtyping, $\delta^S \leq_S \delta^S$.
   - **Case** $+_i \leq +$: By definition of static subtyping, $+_i \leq_S +$. □

**Lemma 37** (Equivalence for static subtyping)**.**
*1. If* $A_1^S \leq_S A_2^S$ *then* $A_1^S \leq A_2^S$.
*2. If* $A_1^S \leq A_2^S$ *then* $A_1^S \leq_S A_2^S$.

*Proof.*
1. By induction on the structure of the derivation of $A_1^S \leq_S A_2^S$.
   - **Case** $\mathsf{Unit} \leq_S \mathsf{Unit}$: By definition of subtyping, $\mathsf{Unit} \leq \mathsf{Unit}$.
   - **Case** $$\frac{A_{11}^S \leq_S A_{12}^S \qquad A_{21}^S \leq_S A_{22}^S \qquad \delta_1^S \leq_S \delta_2^S}{(A_{11}^S\,\delta_1^S\,A_{21}^S) \leq_S (A_{21}^S\,\delta_2^S\,A_{22}^S)}$$

| | |
|---|---|
| $A_{11}^S \leq_S A_{12}^S$ | Subderivation |
| $A_{21}^S \leq_S A_{22}^S$ | Subderivation |
| $\delta_1^S \leq_S \delta_2^S$ | Subderivation |
| $A_{11}^S \leq A_{12}^S$ | By the induction hypothesis |
| $A_{21}^S \leq A_{22}^S$ | By the induction hypothesis |
| $\delta_1^S \leq \delta_2^S$ | By Lemma 36 (Equivalence for static subsum) |
| $A_{11}^S\,\delta_1^S\,A_{21}^S \leq A_{12}^S\,\delta_2^S\,A_{22}^S$ | By definition of $\leq$ |

   - **Case** $$\frac{A_{12}^S \leq_S A_{11}^S \qquad A_{21}^S \leq_S A_{22}^S}{(A_{11}^S \to A_{21}^S) \leq_S (A_{12}^S \to A_{22}^S)}$$
     Similar to the previous case.
2. By induction on the structure of the derivation of $A_1^S \leq A_2^S$.
   - **Case** $\mathsf{Unit} \leq \mathsf{Unit}$: By definition of subtyping, $\mathsf{Unit} \leq_S \mathsf{Unit}$.

- **Case** $\dfrac{A^{S}_{11} \leq A^{S}_{12} \qquad A^{S}_{21} \leq A^{S}_{22} \qquad \delta^{S}_{1} \leq \delta^{S}_{2}}{(A^{S}_{11}\,\delta^{S}_{1}\,A^{S}_{21}) \leq (A^{S}_{21}\,\delta^{S}_{2}\,A^{S}_{22})}$

| | |
|---|---|
| $A^{S}_{11} \leq A^{S}_{12}$ | Subderivation |
| $A^{S}_{21} \leq A^{S}_{22}$ | Subderivation |
| $\delta^{S}_{1} \leq \delta^{S}_{2}$ | Subderivation |
| $A^{S}_{11} \leq_{S} A^{S}_{12}$ | By the induction hypothesis |
| $A^{S}_{21} \leq_{S} A^{S}_{22}$ | By the induction hypothesis |
| $\delta^{S}_{1} \leq_{S} \delta^{S}_{2}$ | By Lemma 36 (Equivalence for static subsum) |
| $A^{S}_{11}\,\delta^{S}_{1}\,A^{S}_{21} \leq_{S} A^{S}_{12}\,\delta^{S}_{2}\,A^{S}_{22}$ | By definition of $\leq_{S}$ |

- **Case** $\dfrac{A^{S}_{12} \leq A^{S}_{11} \qquad A^{S}_{21} \leq A^{S}_{22}}{(A^{S}_{11} \rightarrow A^{S}_{21}) \leq (A^{S}_{12} \rightarrow A^{S}_{22})}$

  Similar to the previous case. ☐

**Lemma 38** (Directed consistency for static types)**.**
*If* $A^{S}_{1} \rightsquigarrow A^{S}_{2}$ *then* $A^{S}_{1} \leq A^{S}_{2}$.

*Proof.* It is given that $A^{S}_{1} \rightsquigarrow A^{S}_{2}$. By inversion on $\mathsf{DirConsU}$, there exist $A$ and $B$ such that $A \sqsubseteq A^{S}_{1}$ and $A \leq B$ and $B \sqsubseteq A^{S}_{2}$. By Lemma 35 (Precision for static types), $A = A^{S}_{1}$ and $B = A^{S}_{2}$. Therefore, $A \leq B$ is equivalent to $A^{S}_{1} \leq A^{S}_{2}$. ☐

**Theorem 5** (Static soundness and completeness)**.**

1. *Soundness:*
   (a) *If* $\Gamma^{S} \vdash_{S} e^{S} \Leftarrow A^{S}$ *then* $\Gamma^{S} \vdash e^{S} \Leftarrow A^{S}$
   (b) *If* $\Gamma^{S} \vdash_{S} e^{S} \Rightarrow A^{S}$ *then* $\Gamma^{S} \vdash e^{S} \Rightarrow A^{S}$.
2. *Completeness:*
   (a) *If* $\Gamma^{S} \vdash e^{S} \Leftarrow A^{S}$ *then* $\Gamma^{S} \vdash_{S} e^{S} \Leftarrow A^{S}$.
   (b) *If* $\Gamma^{S} \vdash e^{S} \Rightarrow A^{S}$ *then* $\Gamma^{S} \vdash_{S} e^{S} \Rightarrow A^{S}$.

*Proof.*

1. By induction on the structure of the given derivation.
   - **Case** $\mathsf{StVar}$: Apply rule $\mathsf{SynVar}$.
   - **Case** $\dfrac{\Gamma^{S} \vdash_{S} e^{S} \Rightarrow A^{S}_{0} \qquad A^{S}_{0} \leq_{S} A^{S}}{\Gamma^{S} \vdash_{S} e^{S} \Leftarrow A^{S}}\ \mathsf{StSub}$

| | |
|---|---|
| $\Gamma^{S} \vdash_{S} e^{S} \Rightarrow A^{S}_{0}$ | Subderivation |
| $A^{S}_{0} \leq_{S} A^{S}$ | Subderivation |
| $\Gamma^{S} \vdash e^{S} \Rightarrow A^{S}_{0}$ | By the induction hypothesis |
| $A^{S}_{0} \leq A^{S}$ | By Lemma 37 (Equivalence for static subtyping) |
| $A^{S}_{0} \rightsquigarrow A^{S}$ | By Lemma 8 (Subtyping obeys directed consistency) |
| $\Gamma^{S} \vdash e^{S} \Leftarrow A^{S}$ | By rule $\mathsf{ChkCSub}$ |

   - **Case** $\mathsf{StAnno}$: Use the induction hypothesis and apply rule $\mathsf{SynAnno}$.
   - **Case** $\mathsf{StUnitIntro}$: Apply rule $\mathsf{ChkUnitIntro}$.
   - **Case** $\dfrac{\Gamma^{S} \vdash_{S} e^{S}_{i} \Leftarrow A^{S}_{i} \qquad +_{i} \leq \delta^{S}}{\Gamma^{S} \vdash_{S} \mathtt{inj}_{i}\, e^{S}_{i} \Leftarrow (A^{S}_{1}\,\delta^{S}\,A^{S}_{2})}\ \mathsf{StSumIntro}$

| | |
|---|---|
| $\Gamma^{S} \vdash_{S} e^{S}_{i} \Leftarrow A^{S}_{i}$ | Subderivation |
| $+_{i} \leq_{S} \delta^{S}$ | Subderivation |
| $\Gamma^{S} \vdash e^{S}_{i} \Leftarrow A^{S}_{i}$ | By the induction hypothesis |
| $+^{?}_{i} \leq +_{i}$ | By definition of $\leq$ |
| $+_{i} \leq \delta^{S}$ | By Lemma 36 (Equivalence for static subsum) |
| $+^{?}_{i} \leq \delta^{S}$ | By transitivity of $\leq$ |
| $\Gamma^{S} \vdash \mathtt{inj}_{i}\, e^{S}_{i} \Leftarrow (A^{S}_{1}\delta^{S}A^{S}_{2})$ | By rule $\mathsf{ChkSumIntro}$ |

   - **Case** $\mathsf{StSumElim1}$: Use the induction hypothesis, the definition of $\Rrightarrow$ and apply rule $\mathsf{ChkSumElim1}$.
   - **Case** $\mathsf{StSumElim2}$: Use the induction hypothesis, the definition of $\Rrightarrow$ and apply rule $\mathsf{ChkSumElim2}$.
   - **Case** $\mathsf{St{\rightarrow}Intro}$: Use the induction hypothesis and apply rule $\mathsf{Chk{\rightarrow}Intro}$.
   - **Case** $\mathsf{St{\rightarrow}Elim}$: Use the induction hypothesis and apply rule $\mathsf{Syn{\rightarrow}Elim}$.

2. By induction on the structure of the given derivation.
   - **Case** SynVar: Apply rule StVar.
   - **Case** $$\frac{\Gamma^{S} \vdash e^{S} \Rightarrow A_0^{S} \qquad A_0^{S} \rightsquigarrow A^{S}}{\Gamma^{S} \vdash e^{S} \Leftarrow A^{S}}\ \mathsf{ChkCSub}$$

| | |
|---|---|
| $\Gamma^{S} \vdash e^{S} \Rightarrow A_0^{S}$ | Subderivation |
| $A_0^{S} \rightsquigarrow A^{S}$ | Subderivation |
| $A_0^{S} \le A^{S}$ | By Lemma 38 (Directed consistency for static types) |
| $\Gamma^{S} \vdash_{S} e^{S} \Rightarrow A_0^{S}$ | By the induction hypothesis |
| $A_0^{S} \le_{S} A^{S}$ | By Lemma 37 (Equivalence for static subtyping) |
| $\Gamma^{S} \vdash_{S} e^{S} \Leftarrow A^{S}$ | By rule StSub |

   - **Case** SynAnno: Use the induction hypothesis and apply rule StAnno.
   - **Case** ChkUnitIntro: Apply rule StUnitIntro.
   - **Case** $$\frac{\Gamma^{S} \vdash e_i^{S} \Leftarrow A_i^{S} \qquad +_i^{?} \le \delta^{S}}{\Gamma^{S} \vdash \mathtt{inj}_i\, e_i^{S} \Leftarrow (A_1^{S}\, \delta^{S}\, A_2^{S})}\ \mathsf{ChkSumIntro}$$

| | |
|---|---|
| $\Gamma^{S} \vdash e_i^{S} \Leftarrow A_i^{S}$ | Subderivation |
| $+_i^{?} \le \delta^{S}$ | Subderivation |
| $\Gamma^{S} \vdash_{S} e_i^{S} \Leftarrow A_i^{S}$ | By the induction hypothesis |
| $+_i \le_{S} \delta^{S}$ | By Lemma 32 (Static looseness) |
| $\Gamma^{S} \vdash_{S} \mathtt{inj}_i\, e_i^{S} \Leftarrow (A_1^{S}\, \delta^{S}\, A_2^{S})$ | By rule StSumIntro |

   - **Case** $$\frac{\begin{array}{l}\Gamma^{S} \vdash e_0^{S} \Rightarrow (A_1^{S}\, \delta^{S}\, A_2^{S}) \\ \delta^{S} \Rrightarrow +_i^{*} \qquad\qquad \Gamma^{S}, x : A_i^{S} \vdash e_i^{S} \Leftarrow A^{S}\end{array}}{\Gamma^{S} \vdash \mathsf{case}(e_0^{S}, \mathtt{inj}_i\, x.e_i^{S}) \Leftarrow A^{S}}\ \mathsf{ChkSumElim1}$$

| | |
|---|---|
| $\Gamma^{S} \vdash e_0^{S} \Rightarrow (A_1^{S}\, \delta^{S}\, A_2^{S})$ | Subderivation |
| $\Gamma^{S}, x : A_i^{S} \vdash e_i^{S} \Leftarrow A^{S}$ | Subderivation |
| $\delta^{S} \Rrightarrow +_i^{*}$ | Subderivation |
| $\Gamma^{S} \vdash_{S} e_0^{S} \Rightarrow (A_1^{S}\, \delta^{S}\, A_2^{S})$ | By the induction hypothesis |
| $\Gamma^{S}, x : A_i^{S} \vdash_{S} e_i^{S} \Leftarrow A^{S}$ | By the induction hypothesis |
| $\delta^{S} = +_i$ | By Lemma 33 (Static looseness, II) |
| $\Gamma^{S} \vdash_{S} \mathsf{case}(e_0^{S}, \mathtt{inj}_i\, x.e_i^{S}) \Leftarrow A^{S}$ | By rule StSumElim1 |

   - **Case** ChkSumElim2: Use the induction hypothesis, the definition of $\le_{S}$ and apply rule StSumElim2.
   - **Case** Chk$\rightarrow$Intro: Use the induction hypothesis and apply rule St$\rightarrow$Intro.
   - **Case** Syn$\rightarrow$Elim: Use the induction hypothesis and apply rule St$\rightarrow$Elim. □

### D.4 Properties of the Dynamic System

**Lemma 39** (Subtyping for dynamic types)**.**
*If* $A_1^{D} \le A_2^{D}$ *then* $A_2^{D} = A_1^{D}$.

*Proof.* By induction on the structure of $A_1^{D}$.

- **Case** $A_1^{D} = \mathsf{Unit}$: By the definition of subtyping, $A_2^{D} = \mathsf{Unit}$ only.
- **Case** $A_1^{D} = A_{11}^{D} +^{?} A_{21}^{D}$:

| | |
|---|---|
| $A_{11}^{D} +^{?} A_{21}^{D} \le A_2^{D}$ | Given |
| $A_2^{D} = A_{12}^{D} +^{?} A_{22}^{D}$ | From the definition of $\le$ |
| $A_{11}^{D} \le A_{12}^{D}$ | ″ |
| $A_{21}^{D} \le A_{22}^{D}$ | ″ |
| $A_{12}^{D} = A_{11}^{D}$ | By the induction hypothesis |
| $A_{22}^{D} = A_{21}^{D}$ | By the induction hypothesis |
| $A_2^{D} = A_1^{D}$ | By definition of $=$ |

- **Case** $A_1^{D} = A_{11}^{D} \rightarrow A_{21}^{D}$: Similar to the previous case. □

**Lemma 40** (Precision for dynamic types)**.**
*If* $A_1 \sqsubseteq A_2^{D}$ *then* $A_1 = A_2^{D}$.

*Proof.* By induction on the structure of $A_2^D$.

- **Case** $A_2^D = \mathsf{Unit}$: By the definition of imprecision, $A_1 = \mathsf{Unit}$ only.
- **Case** $A_2^D = A_{12}^D +^? A_{22}^D$:

$$\begin{array}{rl} A_1 \sqsubseteq A_2^D & \text{Given} \\ A_1 = A_{11} +^? A_{21} & \text{From the definition of } \sqsubseteq \\ A_{11} \sqsubseteq A_{12}^D & '' \\ A_{21} \sqsubseteq A_{22}^D & '' \end{array}$$

$$\begin{array}{rl} A_{11} = A_{12}^D & \text{By the induction hypothesis} \\ A_{21} = A_{22}^D & \text{By the induction hypothesis} \\ A_1 = A_2^D & \text{By definition of } = \end{array}$$

- **Case** $A_1^D = A_{11}^D \to A_{21}^D$: Similar to the previous case. □

**Lemma 41** (Directed consistency for dynamic types)**.**
*If* $A_1^D \rightsquigarrow A_2^D$ *then* $A_1^D = A_2^D$.

*Proof.* It is given that $A_1^D \rightsquigarrow A_2^D$. By inversion on $\mathsf{DirConsU}$, there exist $A$ and $B$ such that $A \sqsubseteq A_1^D$ and $A \leq B$ and $B \sqsubseteq A_2^D$. By Lemma 40 (Precision for dynamic types), $A = A_1^D$ and $B = A_2^D$. Therefore, $A \leq B$ is equivalent to $A_1^D \leq A_2^D$. By Lemma 39 (Subtyping for dynamic types), $A_1^D = A_2^D$. □

**Theorem 15** (Dynamic soundness and completeness)**.**

1. (a) *If* $\Gamma^D \vdash_D e^D \Leftarrow A^D$ *then* $\Gamma^D \vdash e^D \Leftarrow A^D$.
   (b) *If* $\Gamma^D \vdash_D e^D \Rightarrow A^D$ *then* $\Gamma^D \vdash e^D \Rightarrow A^D$.
2. (a) *If* $\Gamma^D \vdash e^D \Leftarrow A^D$ *then* $\Gamma^D \vdash_D e^D \Leftarrow A^D$.
   (b) *If* $\Gamma^D \vdash e^D \Rightarrow A^D$ *then* $\Gamma^D \vdash_D e^D \Rightarrow A^D$.

*Proof.*

1. By induction on the structure of the given $\vdash_D$-derivation.
   - **Case** $\mathsf{DVar}$: Apply rule $\mathsf{SynVar}$.
   - **Case** $\mathsf{DSub}$: Use the induction hypothesis, reflexivity of directed consistency, and apply rule $\mathsf{ChkCSub}$.
   - **Case** $\mathsf{DUnitIntro}$: Apply rule $\mathsf{ChkUnitIntro}$.
   - **Case** $$\frac{\Gamma^D \vdash_D e_i^D \Leftarrow A_i^D}{\Gamma^D \vdash_D \mathtt{inj}_i\, e_i^D \Leftarrow (A_1^D +^? A_2^D)}\ \mathsf{D+^?Intro}$$

$$\begin{array}{rl} \Gamma^D \vdash_D e_i^D \Leftarrow A_i^D & \text{Subderivation} \\ \Gamma^D \vdash e_i^D \Leftarrow A_i^D & \text{By the induction hypothesis} \\ +_i^? \leq +^? & \text{By definition of } \leq \\ \Gamma^D \vdash \mathtt{inj}_i\, e_i^D \Leftarrow (A_1^D +^? A_2^D) & \text{By rule } \mathsf{ChkSumIntro} \end{array}$$

   - **Case** $\mathsf{D+^?Elim1}$: Use the induction hypothesis, the definition of $\Rrightarrow$ and apply rule $\mathsf{ChkSumElim1}$.
   - **Case** $\mathsf{D+^?Elim2}$: Use the induction hypothesis, the definition of $\Rrightarrow$ and apply rule $\mathsf{ChkSumElim2}$.
   - **Case** $\mathsf{D{\to}Intro}$: Use the induction hypothesis and apply rule $\mathsf{Chk{\to}Intro}$.
   - **Case** $\mathsf{D{\to}Elim}$: Use the induction hypothesis and apply rule $\mathsf{Syn{\to}Elim}$.
2. By induction on the structure of the given $\vdash$-derivation.
   - **Case** $\mathsf{SynVar}$: Apply rule $\mathsf{DVar}$.
   - **Case** $$\frac{\Gamma^D \vdash e^D \Rightarrow A_0^D \qquad A_0^D \rightsquigarrow A^D}{\Gamma^D \vdash e^D \Leftarrow A^D}\ \mathsf{ChkCSub}$$

$$\begin{array}{rl} A_0^D \rightsquigarrow A^D & \text{Subderivation} \\ A_0^D = A^D & \text{By Lemma 41 (Directed consistency for dynamic types)} \\ \Gamma^D \vdash e^D \Rightarrow A_0^D & \text{Subderivation} \\ \Gamma^D \vdash_D e^D \Rightarrow A_0^D & \text{By the induction hypothesis} \\ \Gamma^D \vdash_D e^D \Rightarrow A^D & \text{Equivalent} \\ \Gamma^D \vdash_D e^D \Leftarrow A^D & \text{By rule } \mathsf{DSub} \end{array}$$

   - **Case** $\mathsf{SynAnno}$: Use the induction hypothesis and apply rule $\mathsf{DAnno}$.
   - **Case** $\mathsf{ChkUnitIntro}$: Apply rule $\mathsf{DUnitIntro}$.

- **Case** $\mathsf{ChkSumIntro}$: Use the induction hypothesis, and apply rule $\mathsf{D+^?Intro}$.
- **Case** $\mathsf{ChkSumElim1}$: Use the induction hypothesis, and apply rule $\mathsf{D+^?Elim1}$.
- **Case** $\mathsf{ChkSumElim2}$: Use the induction hypothesis, and apply rule $\mathsf{D+^?Elim2}$.
- **Case** $\mathsf{Chk{\to}Intro}$: Use the induction hypothesis and apply rule $\mathsf{D{\to}Intro}$.
- **Case** $\mathsf{Syn{\to}Elim}$: Use the induction hypothesis and apply rule $\mathsf{D{\to}Elim}$. □

## D.5 Target System

### D.5.1 Subtyping

**Lemma 42** (Subtyping inversion)**.**

1. *If* $\mathsf{T}' \leq \mathsf{Unit}$ *then* $\mathsf{T}' = \mathsf{Unit}$.
2. *If* $\mathsf{Unit} \leq \mathsf{T}$ *then* $\mathsf{T} = \mathsf{Unit}$.
3. *If* $\mathsf{T}' \leq \mathsf{T}_1\, \phi\, \mathsf{T}_2$ *then* $\mathsf{T}' = \mathsf{T}_1'\, \phi'\, \mathsf{T}_2'$ *where* $\mathsf{T}_1' \leq \mathsf{T}_1$ *and* $\mathsf{T}_2' \leq \mathsf{T}_2$ *and* $\phi' \leq \phi$.
4. *If* $\mathsf{T}_1'\, \phi'\, \mathsf{T}_2' \leq \mathsf{T}$ *then* $\mathsf{T} = \mathsf{T}_1\, \phi\, \mathsf{T}_2$ *where* $\mathsf{T}_1' \leq \mathsf{T}_1$ *and* $\mathsf{T}_2' \leq \mathsf{T}_2$ *and* $\phi' \leq \phi$.
5. *If* $\mathsf{T}' \leq \mathsf{T}_1 \to \mathsf{T}_2$ *then* $\mathsf{T}' = \mathsf{T}_1' \to \mathsf{T}_2'$ *where* $\mathsf{T}_1 \leq \mathsf{T}_1'$ *and* $\mathsf{T}_2' \leq \mathsf{T}_2$
6. *If* $\mathsf{T}_1' \to \mathsf{T}_2' \leq \mathsf{T}$ *then* $\mathsf{T} = \mathsf{T}_1 \to \mathsf{T}_2$ *where* $\mathsf{T}_1 \leq \mathsf{T}_1'$ *and* $\mathsf{T}_2' \leq \mathsf{T}_2$.

*Proof.*

1. By case analysis on $\mathsf{T}' \leq \mathsf{Unit}$.
   - **Case** $\mathsf{Unit} \leq \mathsf{Unit}$: Immediate that $\mathsf{T}' = \mathsf{Unit}$.
2. Symmetric to part 1.
3. By case analysis on $\mathsf{T}' \leq \mathsf{T}_1\, \phi\, \mathsf{T}_2$.
   - **Case** $\mathsf{T}_1'\, \phi'\, \mathsf{T}_2' \leq \mathsf{T}_1\, \phi\, \mathsf{T}_2$: Immediate as $\mathsf{T}' = \mathsf{T}_1'\, \phi'\, \mathsf{T}_2'$ and subderivations are $\mathsf{T}_1' \leq \mathsf{T}_1$ and $\mathsf{T}_2' \leq \mathsf{T}_2$ and $\phi' \leq \phi$.
4. Symmetric to part 3.
5. By case analysis on $\mathsf{T}' \leq \mathsf{T}_1 \to \mathsf{T}_2$.
   - **Case** $\mathsf{T}_1' \to \mathsf{T}_2' \leq \mathsf{T}_1 \to \mathsf{T}_2$: Immediate as $\mathsf{T}' = \mathsf{T}_1' \to \mathsf{T}_2'$ and subderivations are $\mathsf{T}_1 \leq \mathsf{T}_1'$ and $\mathsf{T}_2' \leq \mathsf{T}_2$.
6. Symmetric to part 5. □

**Lemma 43** (Reflexivity of subtyping)**.**
*For all types* $\mathsf{T}$, *it is the case that* $\mathsf{T} \leq \mathsf{T}$.

*Proof.* By induction on the structure of $\mathsf{T}$.

- **Case** $\mathsf{T} = \mathsf{Unit}$: By the definition of subtyping, $\mathsf{T} \leq \mathsf{T}$.
- **Case** $\mathsf{T} = \mathsf{T}_1\, \phi\, \mathsf{T}_2$: By the induction hypothesis, $\mathsf{T}_1 \leq \mathsf{T}_1$ and $\mathsf{T}_2 \leq \mathsf{T}_2$. By the reflexivity of subsum, $\phi \leq \phi$. Thus, by the definition of subtyping, $\mathsf{T} \leq \mathsf{T}$.
- **Case** $\mathsf{T} = \mathsf{T}_1 \to \mathsf{T}_2$: By the induction hypothesis, $\mathsf{T}_1 \leq \mathsf{T}_1$ and $\mathsf{T}_2 \leq \mathsf{T}_2$. Thus, by the definition of subtyping, $\mathsf{T} \leq \mathsf{T}$. □

**Lemma 44** (Transitivity of subtyping)**.**
*If* $\mathsf{T}_1 \leq \mathsf{T}_2$ *and* $\mathsf{T}_2 \leq \mathsf{T}_3$ *then* $\mathsf{T}_1 \leq \mathsf{T}_2$.

*Proof.* By induction on the structure of $\mathsf{T}_2$.

- **Case** $\mathsf{T}_2 = \mathsf{Unit}$:

| | |
|---|---|
| $\mathsf{T}_1 \leq \mathsf{Unit}$ | Given |
| $\mathsf{Unit} \leq \mathsf{T}_3$ | Given |
| $\mathsf{T}_1 = \mathsf{Unit}$ | By Lemma 42 (Subtyping inversion) |
| $\mathsf{T}_3 = \mathsf{Unit}$ | By Lemma 42 (Subtyping inversion) |
| $\mathsf{Unit} \leq \mathsf{Unit}$ | By Lemma 43 (Reflexivity of subtyping) |
| $\mathsf{T}_1 \leq \mathsf{T}_3$ | Equivalent |

- **Case** $\mathsf{T}_2 = \mathsf{T}_{12}\, \phi_2\, \mathsf{T}_{22}$:

| | |
|---|---|
| $\mathsf{T}_1 \leq \mathsf{T}_{12}\, \phi_2\, \mathsf{T}_{22}$ | Given |
| $\mathsf{T}_1 = \mathsf{T}_{11}\, \phi_1\, \mathsf{T}_{21}$ | By Lemma 42 (Subtyping inversion) |
| $\mathsf{T}_{11} \leq \mathsf{T}_{12}$ | ″ |
| $\mathsf{T}_{21} \leq \mathsf{T}_{22}$ | ″ |
| $\phi_1 \leq \phi_2$ | ″ |
| | |
| $\mathsf{T}_{12}\, \phi_2\, \mathsf{T}_{22} \leq \mathsf{T}_3$ | Given |
| $\mathsf{T}_3 = \mathsf{T}_{13}\, \phi_3\, \mathsf{T}_{23}$ | By Lemma 42 (Subtyping inversion) |
| $\mathsf{T}_{12} \leq \mathsf{T}_{13}$ | ″ |
| $\mathsf{T}_{22} \leq \mathsf{T}_{23}$ | ″ |
| $\phi_2 \leq \phi_3$ | ″ |

$$
\begin{array}{rl}
T_{11} \leq T_{13} & \text{By the induction hypothesis} \\
T_{21} \leq T_{23} & \text{By the induction hypothesis} \\
\phi_1 \leq \phi_3 & \text{By the transitivity of } \leq \\
T_{11}\,\phi_1\,T_{21} \leq T_{13}\,\phi_3\,T_{23} & \text{By the definition of } \leq \\
T_1 \leq T_3 & \text{Equivalent}
\end{array}
$$

- **Case** $T_2 = T_{12} \to T_{22}$:

$$
\begin{array}{rl}
T_1 \leq T_{12} \to T_{22} & \text{Given} \\
T_1 = T_{11} \to T_{21} & \text{By Lemma 42 (Subtyping inversion)} \\
T_{12} \leq T_{11} & '' \\
T_{21} \leq T_{22} & '' \\
\\
T_{12} \to T_{22} \leq T_3 & \text{Given} \\
T_3 = T_{13} \to T_{23} & \text{By Lemma 42 (Subtyping inversion)} \\
T_{13} \leq T_{12} & '' \\
T_{22} \leq T_{23} & '' \\
\\
T_{13} \leq T_{11} & \text{By the induction hypothesis} \\
T_{21} \leq T_{23} & \text{By the induction hypothesis} \\
T_{11} \to T_{21} \leq T_{13} \to T_{23} & \text{By the definition of } \leq \\
T_1 \leq T_3 & \text{Equivalent}
\end{array}
$$

□

**Corollary 45** (Subtyping inversion)**.**

1. *If* $T_1'\,\phi'\,T_2' \leq T_1\,\phi\,T_2$ *then* $T_1' \leq T_1$ *and* $T_2' \leq T_2$ *and* $\phi' \leq \phi$.
2. *If* $T_1' \to T_2' \leq T_1 \to T_2$ *then* $T_1 \leq T_1'$ *and* $T_2' \leq T_2$.

*Proof.*

1. Let $T' = T_1'\,\phi'\,T_2'$. We are given $T' \leq T_1\,\phi\,T_2$. Therefore, by Lemma 42 (Subtyping inversion), $T_1' \leq T_1$ and $T_2' \leq T_2$ and $\phi' \leq \phi$.
2. Let $T' = T_1' \to T_2'$. We are given $T' \leq T_1 \to T_2$. Therefore, by Lemma 42 (Subtyping inversion), $T_1 \leq T_1'$ and $T_2' \leq T_2$. □

### D.5.2 Values

**Lemma 46** (Value inversion)**.**

1. *If* $\cdot \vdash W : T$ *and* $T \leq (T_1 + T_2)$ *then* $W = \mathtt{inj}_i\, W_i$ *and* $\cdot \vdash W_i : T_i$.
   *Moreover, if* $T \leq (T_1 +_k T_2)$ *then* $i = k$.
2. *If* $\cdot \vdash W : T$ *and* $T \leq (T_1 \to T_2)$ *then* $W = \lambda x.\, M$ *and* $\cdot, x : T_1 \vdash M : T_2$.

*Proof.*

1. By induction on the structure of the derivation of $\cdot \vdash W : T$.
   - **Case** TVar: Impossible because context $\Theta = \cdot$ is empty.
   - **Case** $\dfrac{\cdot \vdash W : T' \qquad T' \leq T}{\cdot \vdash W : T}\ \mathsf{TSub}$

     $$
     \begin{array}{ll}
     T' \leq T & \text{Subderivation} \\
     T \leq T_1 + T_2 & \text{Given} \\
     T' \leq T_1 + T_2 & \text{By Lemma 44 (Transitivity of subtyping).}
     \end{array}
     $$

     Immediate from the induction hypothesis.
   - **Cases** TCast, TMatchfail: Impossible because the subject term is not a value.
   - **Case** TUnitIntro: Impossible because $T = \mathsf{Unit}$ cannot be a subtype of $T_1 + T_2$.
   - **Case** $\dfrac{\cdot \vdash W_i : T_i'}{\cdot \vdash \underbrace{\mathtt{inj}_i\, W_i}_{W} : \underbrace{(T_1' +_i T_2')}_{T}}\ \mathsf{T{+}_i Intro}$

     By the definition of values $W$, we know that $W_i$ is a value and $W = \mathtt{inj}_i\, W_i$.

     $$
     \begin{array}{lrl}
     & T_1' +_i T_2' \leq T_1 + T_2 & \text{Given} \\
     & T_i' \leq T_i & \text{By Corollary 45} \\
     & \cdot \vdash W_i : T_i' & \text{Subderivation} \\
     ☞ & \cdot \vdash W_i : T_i & \text{By rule TSub} \\
     \\
     & T_1' +_i T_2' \leq T_1 +_k T_2 & \text{Suppose} \\
     & +_i \leq +_k & \text{By Corollary 45} \\
     ☞ & i = k & \text{From definition of } \leq
     \end{array}
     $$

- **Cases** T$+_i$Elim, T+Elim: Impossible because the subject term is not a value.
- **Case** T$\rightarrow$Intro: Impossible because $\mathsf{T} = \mathsf{T}_1' \rightarrow \mathsf{T}_2'$ cannot be a subtype of $\mathsf{T}_1 + \mathsf{T}_2$.
- **Case** T$\rightarrow$Elim: Impossible because the subject term is not a value.

2. By induction on the structure of the derivation of $\cdot \vdash W : \mathsf{T}$.

- **Case** TVar: Impossible because context $\Theta = \cdot$ is empty.
- **Case** $\dfrac{\cdot \vdash W : \mathsf{T}' \qquad \mathsf{T}' \leq \mathsf{T}}{\cdot \vdash W : \mathsf{T}}$ TSub

  $\mathsf{T}' \leq \mathsf{T}$ Subderivation
  $\mathsf{T} \leq \mathsf{T}_1 \rightarrow \mathsf{T}_2$ Given
  $\mathsf{T}' \leq \mathsf{T}_1 \rightarrow \mathsf{T}_2$ By Lemma 44 (Transitivity of subtyping).

  Immediate from the induction hypothesis.
- **Cases** TCast, TMatchfail: Impossible because the subject term is not a value.
- **Case** TUnitIntro: Impossible because $\mathsf{T} = \mathsf{Unit}$ cannot be a subtype of $\mathsf{T}_1 \rightarrow \mathsf{T}_2$.
- **Case** T$+_i$Intro: Impossible because $\mathsf{T} = \mathsf{T}_1' +_i^? \mathsf{T}_2'$ cannot be a subtype of $\mathsf{T}_1 \rightarrow \mathsf{T}_2$.
- **Cases** T$+_i$Elim, T+Elim: Impossible because the subject term is not a value.
- **Case** $\dfrac{\cdot, x : \mathsf{T}_1' \vdash M : \mathsf{T}_2'}{\cdot \vdash \underbrace{\lambda x.\, M}_{W} : \underbrace{(\mathsf{T}_1' \rightarrow \mathsf{T}_2')}_{\mathsf{T}}}$ T$\rightarrow$Intro

  By the definition of values $W$, we know that $W = \lambda x.\, M$.

  | | | |
  |---|---|---|
  | | $\mathsf{T}_1' \rightarrow \mathsf{T}_2' \leq \mathsf{T}_1 \rightarrow \mathsf{T}_2$ | Given |
  | | $\mathsf{T}_1 \leq \mathsf{T}_1'$ | By Corollary 45 |
  | | $\mathsf{T}_2' \leq \mathsf{T}_2$ | " |
  | | $\cdot, x : \mathsf{T}_1' \vdash M : \mathsf{T}_2'$ | Subderivation |
  | | $\cdot, x : \mathsf{T}_1 \vdash M : \mathsf{T}_2'$ | By Lemma 50 (Context Strengthening) |
  | ☞ | $\cdot, x : \mathsf{T}_1 \vdash M : \mathsf{T}_2$ | By rule TSub |

- **Case** T$\rightarrow$Elim: Impossible because the subject term is not a value. ☐

**Corollary 47** (Target value inversion for $+_i$).
*If* $\cdot \vdash W : (\mathsf{T}_1 +_i \mathsf{T}_2)$ *then* $W = \mathtt{inj}_i\, W_i$ *and* $\cdot \vdash W_i : \mathsf{T}_i$.

*Proof.* Let $\mathsf{T} = \mathsf{T}_1 +_i \mathsf{T}_2$.

| | | |
|---|---|---|
| | $\mathsf{T}_1 \leq \mathsf{T}_1$ | By Lemma 43 (Reflexivity of subtyping) |
| | $\mathsf{T}_2 \leq \mathsf{T}_2$ | By Lemma 43 (Reflexivity of subtyping) |
| | $+_i \leq +$ | By definition of $\leq$ |
| | $\mathsf{T} \leq \mathsf{T}_1 + \mathsf{T}_2$ | By definition of $\leq$ |
| | $\cdot \vdash W : \mathsf{T}$ | Given |
| | $W = \mathtt{inj}_k\, W_k$ | By Lemma 46 (Value inversion) |
| | $\cdot \vdash W_k : \mathsf{T}_k$ | " |
| | $(\mathsf{T} \leq \mathsf{T}_1 +_i \mathsf{T}_2)$ implies $(k = i)$ | " |
| | $\mathsf{T} \leq \mathsf{T}$ | By Lemma 43 (Reflexivity of subtyping) |
| | $i = k$ | Implication |
| ☞ | $W = \mathtt{inj}_i\, W_i$ | Equivalent |
| ☞ | $\cdot \vdash W_i : \mathsf{T}_i$ | Equivalent |

☐

**Corollary 48** (Target value inversion for $+$).
*If* $\cdot \vdash W : (\mathsf{T}_1 + \mathsf{T}_2)$ *then* $W = \mathtt{inj}_i\, W_i$ *and* $\cdot \vdash W_i : \mathsf{T}_i$.

*Proof.* By Lemma 46 (Value inversion) with $\mathsf{T} = \mathsf{T}_1 + \mathsf{T}_2$, using Lemma 43 (Reflexivity of subtyping). ☐

**Corollary 49.**
*If* $\cdot \vdash W : (\mathsf{T}_1 \rightarrow \mathsf{T}_2)$ *then* $W = \lambda x.\, M_0$ *and* $\cdot, x : \mathsf{T}_1 \vdash M_0 : \mathsf{T}_2$.

*Proof.* By Lemma 46 (Value inversion) with $\mathsf{T} = \mathsf{T}_1 \rightarrow \mathsf{T}_2$, using Lemma 43 (Reflexivity of subtyping). ☐

### D.5.3 Typing and Evaluation Contexts

**Lemma 50** (Context Strengthening)**.**
*If* $\Theta, y : T' \vdash M : T_0$ *and* $T \leq T'$ *then* $\Theta, y : T \vdash M : T_0$.

*Proof.* By induction on the structure of the derivation of $\Theta, y : T' \vdash M : T_0$.

- **Case** $\dfrac{(\Theta, y : T')(M) = T_0}{\Theta, y : T' \vdash M : T_0}$ TVar

  Either $M = y$, or $M \neq y$.
  In the first case:

| | |
|---|---|
| $(\Theta, y : T')(M) = T_0$ | Premise |
| $T' = T_0$ | By definition |
| $\Theta, y : T \vdash y : T$ | By rule TVar |
| $T \leq T'$ | Given |
| $\Theta, y : T \vdash y : T'$ | By rule TSub |
| $\Theta, y : T \vdash M : T_0$ | By above equalities |

  In the second case:

  $\Theta, y : T \vdash M : T_0$ By rule TVar

- **Case** TSub: Use the induction hypothesis and apply rule TSub.
- **Case** TCast: Use the induction hypothesis and apply rule TCast.
- **Case** TMatchfail: Immediate from rule TMatchfail.
- **Case** TUnitIntro: Immediate from rule TUnitIntro.
- **Case** T+$_i$Intro: Use the induction hypothesis and apply rule T+$_i$Intro.
- **Case** T+$_i$Elim: Use the induction hypothesis and apply rule T+$_i$Elim.
- **Case** T+Elim: Use the induction hypothesis and apply rule T+Elim.
- **Case** T$\rightarrow$Intro: Use the induction hypothesis and apply rule T$\rightarrow$Intro.
- **Case** T$\rightarrow$Elim: Use the induction hypothesis and apply rule T$\rightarrow$Elim. □

**Lemma 51** (Substitution)**.**
*If* $\Theta, x : T' \vdash M : T$ *and* $\cdot \vdash W : T'$ *then* $\Theta \vdash [W/x]M : T$.

*Proof.* By induction on the structure of the derivation of $\Theta, x : T' \vdash M : T$.

- **Case** TVar: Use the definition of substitution, well-formedness of $\Theta$, and rule TVar.
- **Case** TSub: Use the induction hypothesis and apply rule TSub.
- **Case** TCast: Use the definition of substitution, the induction hypothesis and apply rule TCast.
- **Case** TMatchfail: Use the definition of substitution and apply rule TMatchfail.
- **Case** TUnitIntro: Use the definition of substitution and apply rule TUnitIntro.
- **Case** T+$_i$Intro: Use the definition of substitution, the induction hypothesis and apply rule T+$_i$Intro.
- **Case** T+$_i$Elim: Use the definition of substitution, the induction hypothesis and apply rule T+$_i$Elim.
- **Case** T+Elim: Use the definition of substitution, the induction hypothesis and apply rule T+Elim.
- **Case** T$\rightarrow$Intro: Use the definition of substitution, the induction hypothesis and apply rule T$\rightarrow$Intro.
- **Case** T$\rightarrow$Elim: Use the definition of substitution, the induction hypothesis and apply rule T$\rightarrow$Elim. □

**Lemma 52** (Evaluation context typing)**.**
*If* $\Theta \vdash \mathcal{E}[M_0] : T$ *then there exists* $T_0$ *such that* $\Theta \vdash M_0 : T_0$.

*Proof.* By induction on the structure of the derivation of $\Theta \vdash \mathcal{E}[M_0] : T$.

- **Case** TVar: Immediate as $\mathcal{E}[M_0] = M_0$, so $T_0 = T$.
- **Case** TSub: Immediate from the induction hypothesis.
- **Case** TCast: Immediate from the induction hypothesis.
- **Case** TMatchfail: Immediate as $\mathcal{E}[M_0] = M_0$, so $T_0 = T$.
- **Case** TUnitIntro: Immediate as $\mathcal{E}[M_0] = M_0$, so $T_0 = T$.
- **Case** T+$_i$Intro: Immediate from the induction hypothesis.
- **Case** T+$_i$Elim: Immediate from the induction hypothesis.
- **Case** T+Elim: Immediate from the induction hypothesis.
- **Case** T$\rightarrow$Intro: Immediate as $\mathcal{E}[M_0] = M_0$, so $T_0 = T$.

- **Case** T→Elim: Proceed by case analysis on $\mathcal{E}$. Each case is immediate from the induction hypothesis. □

**Lemma 53** (Evaluation context replacement)**.**
*If* $\Theta \vdash \mathcal{E}[M_0] : T$ *and* $\Theta \vdash M_0 : T_0$ *and* $\Theta \vdash M_0' : T_0$ *then* $\Theta \vdash \mathcal{E}[M_0'] : T$.

*Proof.* By induction on the structure of the derivation of $\Theta \vdash \mathcal{E}[M_0] : T$.

- **Case** TVar: Immediate as $\mathcal{E}[M_0] = M_0$, so $T_0 = T$ and $\mathcal{E}[M_0'] = M_0'$.
- **Case** TSub: Use the induction hypothesis and apply rule TSub.
- **Case** TCast: Use the induction hypothesis and apply rule TCast.
- **Case** TMatchfail: Immediate as $\mathcal{E}[M_0] = M_0$, so $T_0 = T$ and $\mathcal{E}[M_0'] = M_0'$.
- **Case** TUnitIntro: Immediate as $\mathcal{E}[M_0] = M_0$, so $T_0 = T$ and $\mathcal{E}[M_0'] = M_0'$.
- **Case** T$+_i$Intro: Use the induction hypothesis and apply rule T$+_i$Intro.
- **Case** T$+_i$Elim: Use the induction hypothesis and apply rule T$+_i$Elim.
- **Case** T+Elim: Use the induction hypothesis and apply rule T+Elim.
- **Case** T→Intro: Immediate as $\mathcal{E}[M_0] = M_0$, so $T_0 = T$ and $\mathcal{E}[M_0'] = M_0'$.
- **Case** T→Elim: Proceed by case analysis on $\mathcal{E}$. For each case, use the induction hypothesis and apply rule T→Elim. □

### D.5.4 Type Safety

**Lemma 54** (Type preservation under reduction)**.**
*If* $\cdot \vdash M : T$ *and* $M \mapsto_R M'$ *then* $\cdot \vdash M' : T$.

*Proof.* By induction on the structure of the derivation of $\cdot \vdash M : T$.

- **Case** TVar: Impossible because the context $\Theta = \cdot$ is empty.
- **Case** TSub: Use the induction hypothesis and apply rule TSub.
- **Case**
$$\frac{\cdot \vdash M_0 : (T_1\ \phi'\ T_2)}{\cdot \vdash \underbrace{\langle \phi \Leftarrow \phi' \rangle M_0}_{M} : \underbrace{(T_1\ \phi\ T_2)}_{T}}\ \mathsf{TCast}$$
Proceed by case analysis on $M \mapsto_R M'$.
  - **Case**
$$\frac{\phi' \leq \phi}{\langle \phi \Leftarrow \phi' \rangle \underbrace{W}_{M_0} \mapsto_R \underbrace{W}_{M'}}\ \mathsf{ReduceUpcast}$$

| | |
|---|---|
| $T_1 \leq T_1$ | By Lemma 43 (Reflexivity of subtyping) |
| $T_2 \leq T_2$ | By Lemma 43 (Reflexivity of subtyping) |
| $\phi' \leq \phi$ | Given |
| $(T_1\ \phi'\ T_2) \leq (T_1\ \phi\ T_2)$ | Definition of $\leq$ |
| $\cdot \vdash W : (T_1\ \phi'\ T_2)$ | Subderivation |
| $\cdot \vdash W : (T_1\ \phi\ T_2)$ | By rule TSub |

  - **Case**
$$\frac{}{\langle +_i \Leftarrow + \rangle \underbrace{(\mathtt{inj}_i\ W)}_{M_0} \mapsto_R \underbrace{\mathtt{inj}_i\ W}_{M'}}\ \mathsf{ReduceCastSuccess}$$

| | |
|---|---|
| $\cdot \vdash \mathtt{inj}_i\ W : (T_1 + T_2)$ | Subderivation |
| $\cdot \vdash W : T_i$ | By Corollary 48 |
| $\cdot \vdash \mathtt{inj}_i\ W : (T_1 +_i T_2)$ | By rule T$+_i$Intro |

  - **Case**
$$\frac{\phi' \in \{+_i, +\} \qquad i \neq k}{\langle +_k \Leftarrow \phi' \rangle \underbrace{(\mathtt{inj}_i\ W)}_{M_0} \mapsto_R \underbrace{\mathtt{matchfail}}_{M'}}\ \mathsf{ReduceCastFailure}$$

| | |
|---|---|
| $\cdot \vdash \mathtt{matchfail} : T$ | By rule TMatchfail |

- **Case** TMatchfail: Impossible because $\mathtt{matchfail} \not\mapsto_R M'$ for any $M'$.
- **Case** TUnitIntro: Impossible because $() \not\mapsto_R M'$ for any $M'$.
- **Case** T$+_i$Intro: Impossible because $M = \mathtt{inj}_i\ M_0 \not\mapsto_R M'$ for any $M'$.

- **Case** $\dfrac{\cdot \vdash M_0 : T_1 +_i T_2 \qquad \cdot, x : T_i \vdash M_i : T}{\cdot \vdash \underbrace{\mathsf{case}(M_0, \mathsf{inj}_i\, x.M_i)}_{M} : T}$ T$+_i$Elim

  Proceed by case analysis on $M \mapsto_R M'$.

  - **Case**

    $$\dfrac{}{\mathsf{case}(\underbrace{\mathsf{inj}_i\, W}_{M_0}, \mathsf{inj}_i\, x.M_i) \mapsto_R \underbrace{[W/x]M_i}_{M'}}\ \mathsf{ReduceCase1}$$

    | | |
    |---|---|
    | $\cdot \vdash \mathsf{inj}_i\, W : T_1 +_i T_2$ | Subderivation |
    | $\cdot \vdash W : T_i$ | By Corollary 47 |
    | $\cdot, x : T_i \vdash M_i : T$ | Subderivation |
    | $\cdot \vdash [W/x]M_i : T$ | By Lemma 51 (Substitution) |

- **Case** T+Elim: Similar to the T$+_i$Elim case. Apply Corollary 48 instead of Corollary 47 when considering the ReduceCase2 case.
- **Case** T$\rightarrow$Intro: Impossible because $M = \lambda x.\, M_0 \not\mapsto_R M'$ for any $M'$.
- **Case** $\dfrac{\cdot \vdash M_1 : T' \rightarrow T \qquad \cdot \vdash M_2 : T'}{\cdot \vdash \underbrace{M_1\, M_2}_{M} : T}$ T$\rightarrow$Elim

  Proceed by case analysis on $M \mapsto_R M'$.

  - **Case**

    $$\dfrac{}{\underbrace{(\lambda x.\, M_0)}_{M_1}\ \underbrace{W}_{M_2} \mapsto_R \underbrace{[W/x]M_0}_{M'}}\ \mathsf{Reduce}\beta$$

    | | |
    |---|---|
    | $\cdot \vdash W : T'$ | Subderivation |
    | $\cdot \vdash \lambda x.\, M_0 : T' \rightarrow T$ | Subderivation |
    | $\cdot, x : T' \vdash M_0 : T$ | By Corollary 49 |
    | $\cdot \vdash [W/x]M_0 : T$ | By Lemma 51 (Substitution) |

$\square$

**Theorem 6** (Type preservation)**.**
*If $\cdot \vdash M : T$ and $M \mapsto M'$ then $\cdot \vdash M' : T$.*

*Proof.* By case analysis on $M \mapsto M'$.

- **Case** $\dfrac{M_0 \mapsto_R M_0'}{\mathcal{E}[M_0] \mapsto \mathcal{E}[M_0']}$ StepContext

  | | |
  |---|---|
  | $\cdot \vdash \mathcal{E}[M_0] : T$ | Given |
  | $\cdot \vdash M_0 : T_0$ | By Lemma 52 (Evaluation context typing) |
  | $M_0 \mapsto_R M_0'$ | Subderivation |
  | $\cdot \vdash M_0' : T_0$ | By Lemma 54 (Type preservation under reduction) |
  | $\cdot \vdash \mathcal{E}[M_0'] : T$ | By Lemma 53 (Evaluation context replacement) |

- **Case** $\dfrac{\mathcal{E} \neq [\,]}{\mathcal{E}[\mathtt{matchfail}] \mapsto \mathtt{matchfail}}$ StepMatchfail

  Immediate by TMatchfail. $\square$

**Theorem 7** (Progress)**.**
*If $\cdot \vdash M : T$ then either (a) $M$ is a value, or (b) there exists $M'$ such that $M \mapsto M'$, or (c) $M = \mathtt{matchfail}$.*

*Proof.* By induction on the structure of the derivation of $\cdot \vdash M : T$.

- **Case** TVar: Impossible, because the context $\Theta$ is empty.
- **Case** TSub: Immediate by the induction hypothesis.
- **Case** TCast:
  We have $M = \langle \phi \Leftarrow \phi' \rangle M_0$ and $T = T_1\, \phi\, T_2$ where $\cdot \vdash M_0 : (T_1\, \phi'\, T_2)$.
  By the induction hypothesis, either $M_0$ is a value or there exists $M_0'$ such that $M_0 \mapsto M_0'$.
  In the first case, we need to consider all possible assignments to $\phi'$ and $\phi$.
  Suppose $\phi' \leq \phi$, then $M \mapsto M_0$.
  Suppose $\phi' = +_i$ and $\phi = +_k$ where $i \neq k$, then $M_0 = \mathsf{inj}_i\, W$ by Corollary 47, so $M \mapsto \mathtt{matchfail}$.

Suppose $\phi' = +$ and $\phi = +_i$, then $M_0 = \mathtt{inj}_k\, W$ by Corollary 48. Proceed by cases analysis on $i$, if $i = k$ then $M \mapsto M_0$, otherwise $M \mapsto \mathtt{matchfail}$.
In the second case, $\langle \phi \Leftarrow \phi' \rangle M_0 \mapsto \langle \phi \Leftarrow \phi' \rangle M_0'$.

- **Case** TUnitIntro: We have $M = ()$, a value, which is alternative (a).
- **Case** TMatchfail: We have $M = \mathtt{matchfail}$, which is alternative (c).
- **Case** $\mathsf{T}{+_i}\mathsf{Intro}$:
  We have $M = \mathtt{inj}_i\, M_0$ and $T = T_1 +_i^? T_2$ where $\cdot \vdash M_0 : T_i$.
  By the induction hypothesis, either $M_0$ is a value or there exists $M_0'$ such that $M_0 \mapsto M_0'$.
  In the first case, $\mathtt{inj}_i\, M_0 = M$ is a value.
  In the second case, $\mathtt{inj}_i\, M_0 \mapsto \mathtt{inj}_i\, M_0'$.
- **Case** $\mathsf{T}{+_i}\mathsf{Elim}$:
  We have $M = \mathtt{case}(M_0, \mathtt{inj}_i\, x.M_i)$ where $\cdot \vdash M_0 : T_1 +_i T_2$ and $\cdot, x : T_i \vdash M_i : T$.
  By the induction hypothesis, either $M_0$ is a value or there exists $M_0'$ such that $M_0 \mapsto M_0'$.
  In the first case, $M_0 = \mathtt{inj}_i\, W$ by Corollary 47, so $\mathtt{case}(M_0, \mathtt{inj}_i\, x.M_i) \mapsto [W/x]M_i$.
  In the second case, $\mathtt{case}(M_0, \mathtt{inj}_i\, x.M_i) \mapsto \mathtt{case}(M_0', \mathtt{inj}_i\, x.M_i)$.
- **Case** T+Elim: Similar to the $\mathsf{T}{+_i}\mathsf{Elim}$ case, using Corollary 48 instead of Corollary 47.
- **Case** $\mathsf{T}{\rightarrow}\mathsf{Intro}$: We have $M = \lambda x.\, M_0$, a value.
- **Case** $\mathsf{T}{\rightarrow}\mathsf{Elim}$:
  We have $M = M_1\, M_2$ where $\cdot \vdash M_1 : T_1 \rightarrow T_2$ and $\cdot \vdash M_2 : T_1$.
  By the induction hypothesis, either $M_1$ is a value or there exists $M_1'$ such that $M_1 \mapsto M_1'$.
  In the first case, $M_1 = \lambda x.\, M_0$ by Corollary 49.
  By the induction hypothesis, either $M_2$ is a value or there exists $M_2'$ such that $M_2 \mapsto M_2'$.
  In the first subcase, $M_2$ is a value, so $(\lambda x.\, M_0)\, M_2 \mapsto [M_2/x]M_0$.
  In the second subcase, $(\lambda x.\, M_0)\, M_2 \mapsto (\lambda x.\, M_0)\, M_2'$.
  In the second case, $M_1\, M_2 \mapsto M_1'\, M_2$. ☐

**Theorem 8** (matchfail-freeness)**.**
*If $M$ is cast-free and* matchfail*-free and $M \mapsto M'$ then $M'$ is cast-free and* matchfail*-free.*

*Proof.* By induction on the derivation of $M \mapsto M'$.
By the assumption that $M$ is matchfail-free, rule StepMatchfail is impossible. Therefore, the derivation is by StepContext with subderivation $M_0 \mapsto_R M_0'$, where $M = \mathcal{E}[M_0]$ and $M' = \mathcal{E}[M_0']$.

- **Cases** ReduceUpcast, ReduceCastSuccess, ReduceCastFailure: In these cases, $M_0$ contains a cast, contradicting the assumption that $M = \mathcal{E}[M_0]$ is cast-free. Hence, these cases are impossible.
- **Case** ReduceCase1:
  We have $M_0 = \mathtt{case}(\mathtt{inj}_i\, W, \mathtt{inj}_i\, x.M_i)$ and $M_0' = [W/x]M_i$.
  Since $M_0$ is cast- and matchfail-free, its subterms $W$ and $M_i$ are cast- and matchfail-free.
  Therefore, $[W/x]M_i$ is cast- and matchfail-free.
- **Cases** ReduceCase2, Reduce$\beta$: Similar to the ReduceCase1 case. ☐

### D.5.5 Precision

**Lemma 55** (Precision on values)**.**
*If $W' \preccurlyeq M$ then $M = W$ for some value $M$.*

*Proof.* By induction on the structure of the derivation of $W' \preccurlyeq M$.

- **Case** $() \preccurlyeq M$: From definition of $\preccurlyeq$, it is immediate that $M = ()$, a value.
- **Case** $x \preccurlyeq M$: From definition of $\preccurlyeq$, it is immediate that $M = x$, a value.
- **Case** $\lambda x.\, M_0' \preccurlyeq M$: From definition of $\preccurlyeq$, it is immediate that $M = \lambda x.\, M_0$, a value.
- **Case** $\mathtt{inj}_i\, W_0' \preccurlyeq M$: From definition of $\preccurlyeq$, $M = \mathtt{inj}_i\, M_0$ and $W_0' \preccurlyeq M_0$. By the induction hypothesis, $M_0 = W_0$ for some value $W_0$. Therefore, $M = \mathtt{inj}_i\, W_0$, a value. ☐

**Lemma 56** (Substitution preserves precision)**.**
*If $M' \preccurlyeq M$ and $W' \preccurlyeq W$ then $[W'/x]M' \preccurlyeq [W/x]M$.*

*Proof.* By induction on the structure of the derivation of $M' \preccurlyeq M$. All cases are immediate by the induction hypothesis, the definition of substitution, and the definition of $\preccurlyeq$. ☐

**Lemma 57** (Precision inversion on evaluation contexts)**.**
*If $\mathcal{E}'[M_0'] \preccurlyeq M$ then there exists $\mathcal{E}$ and $M_0$ such that $M = \mathcal{E}[M_0]$ and $M_0' \preccurlyeq M_0$.*

*Proof.* Proceed by induction on the structure of $\mathcal{E}'$.

- **Case** $\mathcal{E}' = []$: Choose $\mathcal{E} = []$ and $M_0 = M$ then $M_0' \preccurlyeq M_0$ is given.

- **Case** $\mathcal{E}' = \mathtt{inj}_i\, \mathcal{E}'_0$:

$$\begin{array}{lrl}
 & \mathcal{E}'[M'_0] \preccurlyeq M & \text{Given} \\
 & \mathtt{inj}_i\, \mathcal{E}'_0[M'_0] \preccurlyeq M & \text{By above equations} \\
 & M = \mathtt{inj}_i\, M_i & \text{From the definition of } \preccurlyeq \\
 & \mathcal{E}'_0[M'_0] \preccurlyeq M_i & '' \\
 & M_i = \mathcal{E}_0[M_0] & \text{By the induction hypothesis} \\
\text{☞} & M'_0 \preccurlyeq M_0 & '' \\
\text{☞} & M = \mathtt{inj}_i\, \mathcal{E}_0[M_0] & \text{By above equations}
\end{array}$$

- **Case** $\mathcal{E}' = \mathtt{case}(\mathcal{E}'_0, \mathtt{inj}_i\, x.M'_i)$: Similar to the $\mathcal{E}' = \mathtt{inj}_i\, \mathcal{E}'_0$ case, hence omitted.
- **Case** $\mathcal{E}' = \mathtt{case}(\mathcal{E}'_0, \mathtt{inj}_1\, x_1.M'_1, \mathtt{inj}_2\, x_2.M'_2)$: Similar to the $\mathcal{E}' = \mathtt{inj}_i\, \mathcal{E}'_0$ case, hence omitted.
- **Case** $\mathcal{E}' = \langle \phi'_2 \Leftarrow \phi'_1 \rangle \mathcal{E}'_0$:
  By inversion on $\langle \phi'_2 \Leftarrow \phi'_1 \rangle \mathcal{E}'_0[M'_0] \preccurlyeq M$, either $M = \langle \phi_2 \Leftarrow \phi_1 \rangle M_1$ or $M \neq \langle \phi_2 \Leftarrow \phi_1 \rangle M_1$.
  In the former case:

$$\begin{array}{lrl}
 & \mathcal{E}'_0[M'_0] \preccurlyeq M_1 & \text{From the definition of } \preccurlyeq \\
 & M_1 = \mathcal{E}_0[M_0] & \text{By the induction hypothesis} \\
\text{☞} & M'_0 \preccurlyeq M_0 & '' \\
\text{☞} & M = \langle \phi_2 \Leftarrow \phi_1 \rangle \mathcal{E}_0[M_0] & \text{By above equations}
\end{array}$$

  In the latter case:

$$\begin{array}{lrl}
 & \mathcal{E}'_0[M'_0] \preccurlyeq M & \text{From the definition of } \preccurlyeq \\
\text{☞} & M = \mathcal{E}[M_0] & \text{By the induction hypothesis} \\
\text{☞} & M'_0 \preccurlyeq M_0 & ''
\end{array}$$

- **Case** $\mathcal{E}' = \mathcal{E}'_0\, M'_2$:

$$\begin{array}{lrl}
 & \mathcal{E}'[M'_0] \preccurlyeq M & \text{Given} \\
 & \mathcal{E}'_0[M'_0]\, M'_2 \preccurlyeq M & \text{By above equations} \\
 & M = M_1\, M_2 & \text{From the definition of } \preccurlyeq \\
 & \mathcal{E}'_0[M'_0] \preccurlyeq M_1 & '' \\
 & M_1 = \mathcal{E}_0[M_0] & \text{By the induction hypothesis} \\
\text{☞} & M'_0 \preccurlyeq M_0 & '' \\
\text{☞} & M = \mathcal{E}_0[M_0]\, M_2 & \text{By above equations}
\end{array}$$

- **Case** $\mathcal{E}' = W_1\, \mathcal{E}'_0$:

$$\begin{array}{lrl}
 & \mathcal{E}'[M'_0] \preccurlyeq M & \text{Given} \\
 & W'_1\, \mathcal{E}'_0[M'_0] \preccurlyeq M & \text{By above equations} \\
 & M = M_1\, M_2 & \text{From the definition of } \preccurlyeq \\
 & \mathcal{E}'_0[M'_0] \preccurlyeq M_2 & '' \\
 & M_2 = \mathcal{E}_0[M_0] & \text{By the induction hypothesis} \\
\text{☞} & M'_0 \preccurlyeq M_0 & '' \\
\text{☞} & M = W_1\, \mathcal{E}_0[M_0] & \text{By above equations}
\end{array}$$

□

**Lemma 58** (Evaluation contexts preserve precision)**.**
*If $\mathcal{E}'[M'_0] \preccurlyeq \mathcal{E}[M_0]$ and $M'_0 \preccurlyeq M_0$ and $M'_1 \preccurlyeq M_1$ then $\mathcal{E}'[M'_1] \preccurlyeq \mathcal{E}[M_1]$.*

*Proof.* By induction on the derivation of $\mathcal{E}'[M'_0] \preccurlyeq \mathcal{E}[M_0]$. All cases are straightforward, using the induction hypothesis and the definition of $\preccurlyeq$. □

**Lemma 59** (Reduction preserves precision)**.**
*If $\cdot \vdash M'_1 : T'_1$ and $\cdot \vdash M_1 : T_1$ and $M'_1 \preccurlyeq M_1$ and $M'_1 \longmapsto_R M'_2$ then either*
*(a) $M_1$ is a value and $M'_2 \preccurlyeq M_1$, or*
*(b) there exists $M_2$ such that $M_1 \longmapsto_R M_2$ and $M'_2 \preccurlyeq M_2$.*

*Proof.* Proceed by case analysis on $M'_1 \longmapsto_R M_1$.

- **Case**

$$\dfrac{\phi'_1 \leq \phi'_2}{\underbrace{\langle \phi'_2 \Leftarrow \phi'_1 \rangle W'}_{M'_1} \longmapsto_R \underbrace{W'}_{M'_2}}\ \mathsf{ReduceUpcast}$$

  Proceed by case analysis on $M'_1 \preccurlyeq M_1$.

- **Case** $\dfrac{W' \preccurlyeq M \qquad \langle \phi_2' \Leftarrow \phi_1' \rangle \preccurlyeq \langle \phi_2 \Leftarrow \phi_1 \rangle}{\langle \phi_2' \Leftarrow \phi_1' \rangle W' \preccurlyeq \underbrace{\langle \phi_2 \Leftarrow \phi_1 \rangle M}_{M_1}}$

  By Lemma 55 (Precision on values), $M = W$ as $W' \preccurlyeq M$. Since $\phi_1' \leq \phi_2'$, it is the case that $\langle \phi_2' \Leftarrow \phi_1' \rangle = \mathsf{sc}'$.
  Proceed by cases on the rule deriving $\mathsf{sc}' \preccurlyeq \langle \phi_2 \Leftarrow \phi_1 \rangle$.

  - **Case** Cast$\preccurlyeq$Refl: In this case, $\langle \phi_2 \Leftarrow \phi_1 \rangle = \mathsf{sc}'$. Since $\phi_1 \leq \phi_2$ by rule ReduceUpcast it follows that $M_1 \mapsto_R W$, and we already have $M_2' \preccurlyeq M_2$.
  - **Cases** CastM$\preccurlyeq$B, CastB$\preccurlyeq$S, CastM$\preccurlyeq$S: These rules do not have a safe cast on the left, so these cases are impossible.
  - **Case** Rule deriving $\langle +_i \Leftarrow +_i \rangle \preccurlyeq \langle +_i \Leftarrow + \rangle$:
    In this case, $\mathsf{sc}' = \langle +_i \Leftarrow +_i \rangle$ and $\langle \phi_2 \Leftarrow \phi_1 \rangle = \langle +_i \Leftarrow + \rangle$.

| | |
|---|---|
| $\cdot \vdash \langle +_i \Leftarrow +_i \rangle W' : T_1'$ | Given |
| $\cdot \vdash W' : T_{11}' +_i T_{21}'$ | By inversion on rule TCast |
| $W' = \mathtt{inj}_i\, W_0'$ | By Corollary 47 |
| $W = \mathtt{inj}_i\, W_0$ | By inversion on $(\mathtt{inj}_i\, W_0') \preccurlyeq W$ |
| $M = \mathtt{inj}_i\, W_0$ | By equality |
| $M_1 \mapsto_R (\mathtt{inj}_i\, W_0)$ | By ReduceCastSuccess |
| $M_2' \preccurlyeq M_2$ | By equality |

  - **Cases** Remaining rules:
    In the remaining rules, $\langle \phi_2 \Leftarrow \phi_1 \rangle = \mathsf{sc}$. Thus, $\phi_1 \leq \phi_2$.
    By rule ReduceUpcast it follows that $M_1 \mapsto_R \mathtt{inj}_i\, W_0$ and it was already given that $M_2' \preccurlyeq M_2$.

- **Case** $\dfrac{W' \preccurlyeq M_1}{\langle \phi_2' \Leftarrow \phi_1' \rangle W' \preccurlyeq M_1}$

| | | |
|---|---|---|
| | $W' \preccurlyeq M_1$ | Subderivation |
| ☞ | $M_1 = W$ | By Lemma 55 (Precision on values) |
| ☞ | $M_2' \preccurlyeq M_1$ | By above equations |

• **Case**

$$\dfrac{}{\underbrace{\langle +_i \Leftarrow + \rangle \mathtt{inj}_i\, W'}_{M_1'} \mapsto_R \underbrace{\mathtt{inj}_i\, W'}_{M_2'}}\ \textsf{ReduceCastSuccess}$$

Proceed by case analysis on $M_1' \preccurlyeq M_1$.

- **Case** $\dfrac{\mathtt{inj}_i\, W' \preccurlyeq M \qquad \langle +_i \Leftarrow + \rangle \preccurlyeq \langle \phi_2 \Leftarrow \phi_1 \rangle}{\langle +_i \Leftarrow + \rangle \mathtt{inj}_i\, W' \preccurlyeq \underbrace{\langle \phi_2 \Leftarrow \phi_1 \rangle M}_{M_1}}$

  Inversion on $\mathtt{inj}_i\, W' \preccurlyeq M$ gives $M = \mathtt{inj}_i\, M_0$ and $W' \preccurlyeq M_0$.
  By Lemma 55 (Precision on values), $M_0 = W$.
  Since $\langle +_i \Leftarrow + \rangle$ is a backward cast $\mathtt{bc}'$, to derive $\mathtt{bc}' \preccurlyeq \langle \phi_2 \Leftarrow \phi_1 \rangle$, we either used Cast$\preccurlyeq$Refl or CastB$\preccurlyeq$S.
  In the former case, we have $\langle \phi_2 \Leftarrow \phi_1 \rangle = \mathtt{bc}'$. By rule ReduceCastSuccess we have $M_1 \mapsto_R M$, and we already have $M_2' \preccurlyeq M_2$.
  In the latter case, we have $\langle \phi_2 \Leftarrow \phi_1 \rangle = \mathsf{sc}$. By definition of being a safe cast, $\phi_1 \leq \phi_2$. Therefore, by rule ReduceUpcast we have $M_1 \mapsto_R M$, and we already have $M_2' \preccurlyeq M_2$.

- **Case** $\dfrac{\mathtt{inj}_i\, W' \preccurlyeq M_1}{\langle +_i \Leftarrow + \rangle \mathtt{inj}_i\, W' \preccurlyeq M_1}$

| | | |
|---|---|---|
| | $\mathtt{inj}_i\, W' \preccurlyeq M_1$ | Subderivation |
| ☞ | $M_1 = W$ | By Lemma 55 (Precision on values) |
| ☞ | $M_2' \preccurlyeq M_1$ | By above equations |

• **Case**

$$\dfrac{\phi' \in \{+_i, +\} \qquad i \neq k}{\underbrace{\langle +_k \Leftarrow \phi' \rangle \mathtt{inj}_i\, W'}_{M_1'} \mapsto_R \underbrace{\mathtt{matchfail}}_{M_2'}}\ \textsf{ReduceCastFailure}$$

Proceed by case analysis on $M_1' \preccurlyeq M_1$.

- **Case** $\dfrac{\mathsf{inj}_i\, W' \preccurlyeq M \qquad \langle +_k \Leftarrow \phi' \rangle \preccurlyeq \langle \phi_2 \Leftarrow \phi_1 \rangle}{\langle +_k \Leftarrow \phi' \rangle \mathsf{inj}_i\, W' \preccurlyeq \underbrace{\langle \phi_2 \Leftarrow \phi_1 \rangle M}_{M_1}}$

  Since $\cdot \vdash M_1 : T_1$ and $M_1$ is not a value nor is it $\mathtt{matchfail}$, by Theorem 7 there exists $M_2$ such that $M_1 \mapsto M_2$.
  By definition, $M_2' = \mathtt{matchfail} \preccurlyeq M_2$.

- **Case** $\dfrac{\mathsf{inj}_i\, W' \preccurlyeq M_1}{\langle +_k \Leftarrow \phi' \rangle \mathsf{inj}_i\, W' \preccurlyeq M_1}$

| | | |
|---|---|---|
| | $\mathsf{inj}_i\, W' \preccurlyeq M_1$ | Subderivation |
| ☞ | $M_1 = W$ | By Lemma 55 (Precision on values) |
| ☞ | $M_2' \preccurlyeq M_1$ | By definition of $\preccurlyeq$ |

- **Case**

$$\dfrac{}{\underbrace{\mathsf{case}(\mathsf{inj}_i\, W', \mathsf{inj}_i\, x.M_i')}_{M_1'} \mapsto_R \underbrace{[W'/x]M_i'}_{M_2'}}\ \mathsf{ReduceCase1}$$

Proceed by inversion on $\mathsf{case}(\mathsf{inj}_i\, W', \mathsf{inj}_i\, x.M_i') \preccurlyeq M_1$.
In the first case, $M_1 = \mathsf{case}(M, \mathsf{inj}_i\, x.M_i)$:

| | | |
|---|---|---|
| | $\mathsf{inj}_i\, W' \preccurlyeq M$ | From definition of $\preccurlyeq$ |
| | $M_i' \preccurlyeq M_i$ | ″ |
| | $M = \mathsf{inj}_i\, M_0$ | From definition of $\preccurlyeq$ |
| | $W' \preccurlyeq M_0$ | ″ |
| | $M_0 = W$ | By Lemma 55 (Precision on values) |
| | $W' \preccurlyeq W$ | By above equations |
| | $M_1 = \mathsf{case}(\mathsf{inj}_i\, W, \mathsf{inj}_i\, x.M_i)$ | By above equations |
| ☞ | $M_1 \mapsto_R \underbrace{[W/x]M_i}_{M_2}$ | By rule ReduceCase1 |
| ☞ | $[W'/x]M_i' \preccurlyeq [W/x]M_i$ | By Lemma 56 (Substitution preserves precision) |

In the second case, $M_1 = \mathsf{case}(M, \mathsf{inj}_1\, x_1.M_{11}, \mathsf{inj}_2\, x_2.M_{21})$:

| | | |
|---|---|---|
| | $\mathsf{inj}_i\, W' \preccurlyeq M$ | From definition of $\preccurlyeq$ |
| | $M_i' \preccurlyeq M_{i1}$ | ″ |
| | $M = \mathsf{inj}_i\, M_0$ | From definition of $\preccurlyeq$ |
| | $W' \preccurlyeq M_0$ | ″ |
| | $M_0 = W$ | By Lemma 55 (Precision on values) |
| | $W' \preccurlyeq W$ | By above equations |
| | $M_1 = \mathsf{case}(\mathsf{inj}_i\, W, \mathsf{inj}_1\, x_1.M_{11}, \mathsf{inj}_2\, x_2.M_{21})$ | By above equations |
| ☞ | $M_1 \mapsto_R \underbrace{[W/x_i]M_{i1}}_{M_2}$ | By rule ReduceCase2 |
| ☞ | $[W'/x]M_i' \preccurlyeq [W/x_i]M_{i1}$ | By Lemma 56 (Substitution preserves precision) |

- **Case**

$$\dfrac{}{\underbrace{\mathsf{case}(\mathsf{inj}_i\, W', \mathsf{inj}_1\, x_1.M_{11}', \mathsf{inj}_2\, x_2.M_{21}')}_{M_1'} \mapsto_R \underbrace{[W'/x_i]M_{i1}'}_{M_2'}}\ \mathsf{ReduceCase2}$$

| | |
|---|---|
| $M_1' \preccurlyeq M_1$ | Given |
| $M_1 = \mathsf{case}(M, \mathsf{inj}_1\, x_1.M_{11}, \mathsf{inj}_2\, x_2.M_{21})$ | From definition of $\preccurlyeq$ |
| $\mathsf{inj}_i\, W' \preccurlyeq M$ | ″ |
| $M_{11}' \preccurlyeq M_{11}$ | ″ |
| $M_{21}' \preccurlyeq M_{21}$ | ″ |
| $M = \mathsf{inj}_i\, M_0$ | From definition of $\preccurlyeq$ |
| $W' \preccurlyeq M_0$ | ″ |
| $M_0 = W$ | By Lemma 55 (Precision on values) |
| $W' \preccurlyeq W$ | By above equations |

$$M_1 = \mathsf{case}(\mathtt{inj}_i\, W, \mathtt{inj}_1\, x_1.M_{11}, \mathtt{inj}_2\, x_2.M_{21}) \quad \text{By above equations}$$

☞ $M_1 \mapsto_R \underbrace{[W/x_i]M_{i1}}_{M_2}$ By rule ReduceCase2

☞ $[W'/x_i]M'_{i1} \preccurlyeq [W/x_i]M_{i1}$ By Lemma 56 (Substitution preserves precision)

- **Case**

$$\frac{}{\underbrace{(\lambda x.\, M'_0)W'}_{M'_1} \mapsto_R \underbrace{[W'/x]M'_0}_{M'_2}}\ \mathsf{Reduce}\beta$$

| | |
|---|---|
| $(\lambda x.\, M'_0)W' \preccurlyeq M_1$ | Given |
| $M_1 = M_{11}\, M_{21}$ | From definition of $\preccurlyeq$ |
| $\lambda x.\, M'_0 \preccurlyeq M_{11}$ | " |
| $W' \preccurlyeq M_{21}$ | " |
| $M_{11} = \lambda x.\, M_0$ | From definition of $\preccurlyeq$ |
| $M'_0 \preccurlyeq M_0$ | " |
| $M_{21} = W$ | By Lemma 55 (Precision on values) |
| $W' \preccurlyeq W$ | By above equations |
| $M_1 = (\lambda x.\, M_0)W$ | By above equations |
| ☞ $(\lambda x.\, M_0)W \mapsto_R \underbrace{[W/x]M_0}_{M_2}$ | By rule $\mathsf{Reduce}\beta$ |
| ☞ $[W'/x]M_0 \preccurlyeq [W/x]M$ | By Lemma 56 (Substitution preserves precision) |

□

**Theorem 12** (Stepping preserves precision).
*If $\cdot \vdash M'_1 : T'_1$ and $\cdot \vdash M_1 : T_1$ and $M'_1 \preccurlyeq M_1$ and $M'_1 \mapsto M'_2$ then either*
*(a) $M_1$ is a value and $M'_2 \preccurlyeq M_1$, or*
*(b) there exists $M_2$ such that $M_1 \mapsto M_2$ and $M'_2 \preccurlyeq M_2$, or*
*(c) $M_1 = \mathtt{matchfail}$ and $M'_2 \preccurlyeq M_1$.*

*Proof.* Proceed by case analysis on $M'_1 \mapsto M'_2$.

- **Case**

$$\frac{M'_{01} \mapsto_R M'_{02}}{\underbrace{\mathcal{E}'[M'_{01}]}_{M'_1} \mapsto \underbrace{\mathcal{E}'[M'_{02}]}_{M'_2}}\ \mathsf{StepContext}$$

| | |
|---|---|
| $\mathcal{E}'[M'_{01}] \preccurlyeq M_1$ | Given |
| $M_1 = \mathcal{E}[M_{01}]$ | By Lemma 57 (Precision inversion on evaluation contexts) |
| $M'_{01} \preccurlyeq M_{01}$ | " |
| $\cdot \vdash \mathcal{E}'[M'_{01}] : T'_1$ | Given |
| $\cdot \vdash \mathcal{E}[M_{01}] : T_1$ | Given |
| $\cdot \vdash M'_{01} : T'_{01}$ | By Lemma 52 (Evaluation context typing) |
| $\cdot \vdash M_{01} : T_{01}$ | By Lemma 52 (Evaluation context typing) |
| $M'_{01} \mapsto_R M'_{02}$ | Given |

Proceed by case analysis on the result of applying Lemma 59 (Reduction preserves precision).
In the first case, $M_{01}$ is a value and $M'_{02} \preccurlyeq M_{01}$. Since $M'_{01} \preccurlyeq M_{01}$ and $\mathcal{E}'[M'_{01}] \preccurlyeq \mathcal{E}[M_{01}]$, by Lemma 58 (Evaluation contexts preserve precision) it follows that $\mathcal{E}'[M'_{02}] \preccurlyeq \mathcal{E}[M_{01}]$. This is alternative (a).
In the second case, $M_{01} \mapsto_R M_{02}$ and $M'_{02} \preccurlyeq M_{02}$. Therefore, by rule StepContext it follows $M_1 \mapsto \mathcal{E}[M_{02}]$. Since $M'_{01} \preccurlyeq M_{01}$ and $\mathcal{E}'[M'_{01}] \preccurlyeq \mathcal{E}[M_{01}]$, by Lemma 58 (Evaluation contexts preserve precision) it follows that $\mathcal{E}'[M'_{02}] \preccurlyeq \mathcal{E}[M_{02}]$. This is alternative (b).

- **Case**

$$\frac{\mathcal{E}' \neq [\,]}{\underbrace{\mathcal{E}'[\mathtt{matchfail}]}_{M'_1} \mapsto \underbrace{\mathtt{matchfail}}_{M'_2}}\ \mathsf{StepMatchfail}$$

Since $\cdot \vdash M_1 : T_1$, by Theorem 7 it follows that either $M_1$ is a value, or there exists $M_2$ such that $M_1 \mapsto M_2$, or $M_1 = \mathtt{matchfail}$.
In the first case, $M'_2 = \mathtt{matchfail} \preccurlyeq M_1$ by definition of $\preccurlyeq$, which is alternative (a).
In the second case, $M'_2 = \mathtt{matchfail} \preccurlyeq M_2$ by definition of $\preccurlyeq$, which is alternative (b).
In the first case, $M'_2 = \mathtt{matchfail} \preccurlyeq \mathtt{matchfail} = M_1$ by definition of $\preccurlyeq$, which is alternative (c). □

**Theorem 13** ($\preccurlyeq$ respects convergence).
*If $M' \preccurlyeq M$ where $\cdot \vdash M' : T'$ and $\cdot \vdash M : T$*
*and $M'$ converges then $M$ also converges.*

*Proof.* It is given that $M'$ converges. By Definition 1, there exists a value $W'$ such that $M' \mapsto^* W'$. Proceed by induction on the number of steps in $M' \mapsto^* W'$.

If $M' = W'$ then $W' \preccurlyeq M$. By Lemma 55 (Precision on values), $M = W$ for some value $W$. Therefore, $M$ converges as well.

Otherwise, $M'$ takes at least one step, that is, $M' \mapsto M'_0 \mapsto^* W'$. Then $M'_0$ must also converge, with $M'_0 \mapsto^* W'$ in fewer steps than $M' \mapsto^* W'$. Since $M' \mapsto M'_0$, proceed by case analysis on the result of applying Theorem 12.

- In the first case (a), $M$ is a value, so $M$ converges.
- In the second case (b), there exists $M_0$ such that $M \mapsto M_0$ and $M'_0 \preccurlyeq M_0$.
  By Theorem 6, $\cdot \vdash M'_0 : T'$ and similarly $\cdot \vdash M_0 : T$.
  By the induction hypothesis, $M_0$ converges. Since $M_0$ converges, $M$ must also converge to the same value.
- In the third case (c), $M = \texttt{matchfail}$ and $M'_0 \preccurlyeq M$.
  By inversion on $M'_0 \preccurlyeq \texttt{matchfail}$ it follows that $M'_0 = \texttt{matchfail}$. But we know that $M'_0$ converges, a contradiction. Hence, this case is impossible. □

### D.6 Translation

#### D.6.1 Soundness

**Theorem 16** (Sum Translation soundness)**.**
*Given $\delta'$ and $\delta$, there exists $\mathcal{C}$ such that $\delta' \Rightarrow \delta \hookrightarrow \mathcal{C}$.*
*Moreover, if $\Theta \vdash M : (T_1\,|\delta'|\,T_2)$ then $\Theta \vdash \mathcal{C}[M] : (T_1\,|\delta|\,T_2)$.*

*Proof.* Proceed by case analysis on whether $|\delta'| \leq |\delta|$.

- **Case** $|\delta'| \leq |\delta|$:

| | |
|---|---|
| $T_1 \leq T_1$ | By Lemma 43 (Reflexivity of subtyping) |
| $T_2 \leq T_2$ | By Lemma 43 (Reflexivity of subtyping) |
| $\|\delta'\| \leq \|\delta\|$ | Given |
| $(T_1\,\|\delta'\|\,T_2) \leq (T_1\,\|\delta\|\,T_2)$ | By definition of $\leq$ |
| $\delta' \Rightarrow \delta \hookrightarrow [\,]$ | By rule CoeSub |
| $\Theta \vdash M : (T_1\,\|\delta'\|\,T_2)$ | Suppose |
| $\Theta \vdash M : (T_1\,\|\delta\|\,T_2)$ | By rule TSub |
| $\mathcal{C}[M] = M$ | By definition |
| $\Theta \vdash \mathcal{C}[M] : (T_1\,\|\delta\|\,T_2)$ | By above equations |

- **Case** $|\delta'| \not\leq |\delta|$:

| | |
|---|---|
| $\|\delta'\| \not\leq \|\delta\|$ | Given |
| $\|\delta'\| \Rightarrow \|\delta\| \hookrightarrow \langle \|\delta\| \Leftarrow \|\delta'\| \rangle [\,]$ | By rule CoeCast |
| $\Theta \vdash M : (T_1\,\|\delta'\|\,T_2)$ | Suppose |
| $\Theta \vdash \langle \|\delta\| \Leftarrow \|\delta'\| \rangle M : (T_1\,\|\delta\|\,T_2)$ | By rule TCast |
| $\mathcal{C}[M] = \langle \|\delta\| \Leftarrow \|\delta'\| \rangle M$ | By definition |
| $\Theta \vdash \mathcal{C}[M] : (T_1\,\|\delta\|\,T_2)$ | By above equations |

□

**Theorem 17** (Type translation soundness)**.**
*If $A' \simeq A$ then there exists $\mathcal{C}$ such that $A' \Rightarrow A \hookrightarrow \mathcal{C}$.*
*Moreover, if $\Theta \vdash M : |A'|$ then $\Theta \vdash \mathcal{C}[M] : |A|$.*

*Proof.* By induction on the structure of the derivation of $A' \simeq A$.

- **Case** $\mathsf{Unit} \simeq \mathsf{Unit}$:

| | |
|---|---|
| $\mathsf{Unit} \Rightarrow \mathsf{Unit} \hookrightarrow [\,]$ | By rule CoeUnit |
| $\Theta \vdash M : \|\mathsf{Unit}\|$ | Suppose |
| $\Theta \vdash \mathcal{C}[M] : \|\mathsf{Unit}\|$ | By definition of $\mathcal{C}$ |

- **Case** $$\dfrac{A'_1 \simeq A_1 \qquad A'_2 \simeq A_2}{\underbrace{(A'_1\,\delta'\,A'_2)}_{A'} \simeq \underbrace{(A_1\,\delta\,A_2)}_{A}}$$

  Proceed by case analysis on the definition of $\delta'$.
  In the first case, suppose $\delta' \in \{+^{?}_{1}, +_1\}$.

$$\begin{array}{rl}
|A'_1| \leq |A'_1| & \text{By Lemma 43 (Reflexivity of subtyping)} \\
|A'_2| \leq |A'_2| & \text{By Lemma 43 (Reflexivity of subtyping)} \\
|\delta'| \leq +_1 & \text{By definition of } \leq \\
|A'_1|\,|\delta'|\,|A'_2| \leq |A'_1| +_1 |A'_2| & \text{By definition of } \leq
\end{array}$$

$$\begin{array}{rl}
\Theta \vdash M : |(A'_1\, \delta'\, A'_2)| & \text{Suppose} \\
\Theta \vdash M : (|A'_1|\,|\delta'|\,|A'_2|) & \text{By definition of type translation} \\
\Theta \vdash M : (|A'_1| +_1 |A'_2|) & \text{By rule TSub}
\end{array}$$

$$\begin{array}{rl}
|A_1| \leq |A_1| & \text{By Lemma 43 (Reflexivity of subtyping)} \\
|A_2| \leq |A_2| & \text{By Lemma 43 (Reflexivity of subtyping)} \\
+_1 \leq |\delta'| & \text{By definition of } \leq \\
|A_1| +_1 |A_2| \leq |A_1|\,|\delta'|\,|A_2| & \text{By definition of } \leq
\end{array}$$

$$\begin{array}{ll}
\Theta, x_1 : |A'_1| \vdash x_1 : |A'_1| & \text{By rule TVar} \\
\quad A'_1 \simeq A_1 & \text{Subderivation} \\
\quad A'_1 \Rightarrow A_1 \hookrightarrow \mathcal{C}_1 & \text{By the induction hypothesis} \\
\Theta, x_1 : |A'_1| \vdash \mathcal{C}_1[x_1] : |A_1| & '' \\
\Theta, x_1 : |A'_1| \vdash \mathbf{inj}_1\, \mathcal{C}_1[x_1] : (|A_1| +_1 |A_2|) & \text{By rule T+}_i\text{Intro} \\
\Theta, x_1 : |A'_1| \vdash \mathbf{inj}_1\, \mathcal{C}_1[x_1] : (|A_1|\,|\delta'|\,|A_2|) & \text{By rule TSub}
\end{array}$$

$$\begin{array}{ll}
\Theta \vdash \mathsf{case}(M, \mathbf{inj}_1\, x_1.\mathbf{inj}_1\, \mathcal{C}_1[x_1]) : (|A_1|\,|\delta'|\,|A_2|) & \text{By rule T+}_i\text{Elim} \\
\delta' \Rightarrow \delta \hookrightarrow \mathcal{C}_3 & \text{By Theorem 16} \\
\Theta \vdash \mathcal{C}_3[\mathsf{case}(M, \mathbf{inj}_1\, x_1.\mathbf{inj}_1\, \mathcal{C}_1[x_1])] : (|A_1|\,|\delta|\,|A_2|) & '' \\
\Theta \vdash \underbrace{\mathcal{C}_3[\mathsf{case}(M, \mathbf{inj}_1\, x_1.\mathbf{inj}_1\, \mathcal{C}_1[x_1])] : |(A_1\, \delta\, A_2)|}_{\mathcal{C}[M]} & \text{By definition of type translation}
\end{array}$$

$$(A'_1\, \delta'\, A'_2) \Rightarrow (A_1\, \delta\, A_2) \hookrightarrow \underbrace{\mathcal{C}_3[\mathsf{case}([\,], \mathbf{inj}_1\, x_1.\mathbf{inj}_1\, \mathcal{C}_1[x_1])]}_{\mathcal{C}} \qquad \text{By rule CoeCase1L}$$

In the second case, suppose $\delta' \in \{+^?_2, +_2\}$. Symmetric to the previous case, hence omitted.
In the last case, suppose $\delta' \in \{+^?, +^*_1, +^*_2, +\}$.

$$\begin{array}{rl}
|A'_1| \leq |A'_1| & \text{By Lemma 43 (Reflexivity of subtyping)} \\
|A'_2| \leq |A'_2| & \text{By Lemma 43 (Reflexivity of subtyping)} \\
|\delta'| \leq + & \text{By definition of } \leq \\
|A'_1|\,|\delta'|\,|A'_2| \leq |A'_1| + |A'_2| & \text{By definition of } \leq
\end{array}$$

$$\begin{array}{rl}
\Theta \vdash M : |(A'_1\, \delta'\, A'_2)| & \text{Suppose} \\
\Theta \vdash M : (|A'_1|\,|\delta'|\,|A'_2|) & \text{By definition of type translation} \\
\Theta \vdash M : (|A'_1| + |A'_2|) & \text{By rule TSub}
\end{array}$$

$$\begin{array}{ll}
\Theta, x_1 : |A'_1| \vdash x_1 : |A'_1| & \text{By rule TVar} \\
\quad A'_1 \simeq A_1 & \text{Subderivation} \\
\quad A'_1 \Rightarrow A_1 \hookrightarrow \mathcal{C}_1 & \text{By the induction hypothesis} \\
\Theta, x_1 : |A'_1| \vdash \mathcal{C}_1[x_1] : |A_1| & '' \\
\Theta, x_1 : |A'_1| \vdash \mathbf{inj}_1\, \mathcal{C}_1[x_1] : (|A_1| +_1 |A_2|) & \text{By rule T+}_i\text{Intro} \\
\quad +^?_1 \Rightarrow \delta' \hookrightarrow \mathcal{C}'_1 & \text{By Theorem 16} \\
\Theta, x_1 : |A'_1| \vdash \mathcal{C}'_1[\mathbf{inj}_1\, \mathcal{C}_1[x_1]] : (|A_1|\,|\delta'|\,|A_2|) & ''
\end{array}$$

$$\begin{array}{ll}
\Theta, x_2 : |A'_2| \vdash x_2 : |A'_2| & \text{By rule TVar} \\
\quad A'_2 \simeq A_2 & \text{Subderivation} \\
\quad A'_2 \Rightarrow A_2 \hookrightarrow \mathcal{C}_2 & \text{By the induction hypothesis} \\
\Theta, x_2 : |A'_2| \vdash \mathcal{C}_2[x_2] : |A_2| & '' \\
\Theta, x_2 : |A'_2| \vdash \mathbf{inj}_2\, \mathcal{C}_2[x_2] : (|A_1| +_2 |A_2|) & \text{By rule T+}_i\text{Intro} \\
\quad +^?_2 \Rightarrow \delta' \hookrightarrow \mathcal{C}'_2 & \text{By Theorem 16} \\
\Theta, x_2 : |A'_2| \vdash \mathcal{C}'_2[\mathbf{inj}_2\, \mathcal{C}_2[x_2]] : (|A_1|\,|\delta'|\,|A_2|) & ''
\end{array}$$

$$\begin{array}{ll}
\Theta \vdash \mathsf{case}(M, \mathbf{inj}_1\, x_1.\mathcal{C}'_1[\mathbf{inj}_1\, \mathcal{C}_1[x_1]], \mathbf{inj}_2\, x_2.\mathcal{C}'_2[\mathbf{inj}_2\, \mathcal{C}_2[x_2]]) : (|A_1|\,|\delta'|\,|A_2|) & \text{By rule T+Elim} \\
\delta' \Rightarrow \delta \hookrightarrow \mathcal{C}_3 & \text{By Theorem 16} \\
\Theta \vdash \mathcal{C}_3[\mathsf{case}(M, \mathbf{inj}_1\, x_1.\mathcal{C}'_1[\mathbf{inj}_1\, \mathcal{C}_1[x_1]], \mathbf{inj}_2\, x_2.\mathcal{C}'_2[\mathbf{inj}_2\, \mathcal{C}_2[x_2]])] : (|A_1|\,|\delta|\,|A_2|) & '' \\
\Theta \vdash \underbrace{\mathcal{C}_3[\mathsf{case}(M, \mathbf{inj}_1\, x_1.\mathcal{C}'_1[\mathbf{inj}_1\, \mathcal{C}_1[x_1]], \mathbf{inj}_2\, x_2.\mathcal{C}'_2[\mathbf{inj}_2\, \mathcal{C}_2[x_2]])] : |A_1\, \delta\, A_2)|}_{\mathcal{C}[M]} & \text{By definition}
\end{array}$$

$$(A_1' \,\delta'\, A_2') \Rrightarrow (A_1 \,\delta\, A_2) \hookrightarrow \underbrace{\mathcal{C}_3[\mathsf{case}([\,], \mathsf{inj}_1\, x_1.\mathcal{C}_1'[\mathsf{inj}_1\, \mathcal{C}_1[x_1]], \mathsf{inj}_2\, x_2.\mathcal{C}_2'[\mathsf{inj}_2\, \mathcal{C}_2[x_2]])]}_{\mathcal{C}} \quad \text{By rule CoeCase2}$$

- **Case** $\dfrac{A_1' \simeq A_1 \qquad A_2' \simeq A_2}{\underbrace{(A_1' \to A_2')}_{A'} \simeq \underbrace{(A_1 \to A_2)}_{A}}$

| | |
|---|---|
| $\Theta, x : \lvert A_1 \rvert \vdash x : \lvert A_1 \rvert$ | By rule TVar |
| $A_1' \simeq A_1$ | Subderivation |
| $A_1 \simeq A_1'$ | By Lemma 12 (Symmetry of Structural Equivalence) |
| $A_1 \Rrightarrow A_1' \hookrightarrow \mathcal{C}_1$ | By the induction hypothesis |
| $\Theta, x : \lvert A_1 \rvert \vdash \mathcal{C}_1[x] : \lvert A_1' \rvert$ | " |
| $\Theta \vdash M : \lvert (A_1' \to A_2') \rvert$ | Suppose |
| $\Theta \vdash M : (\lvert A_1' \rvert \to \lvert A_2' \rvert)$ | By definition of type translation |
| $\Theta, x : \lvert A_1 \rvert \vdash M\, \mathcal{C}_1[x_1] : \lvert A_2' \rvert$ | By rule T→Elim |
| $A_2' \simeq A_2$ | Subderivation |
| $A_2' \Rrightarrow A_2 \hookrightarrow \mathcal{C}_2$ | By the induction hypothesis |
| $\Theta, x : \lvert A_1 \rvert \vdash \mathcal{C}_2[M\, \mathcal{C}_1[x_1]] : \lvert A_2 \rvert$ | " |
| $\Theta \vdash \lambda x. \mathcal{C}_2[M\, \mathcal{C}_1[x_1]] : (\lvert A_1 \rvert \to \lvert A_2 \rvert)$ | By rule T→Intro |
| $\Theta \vdash \underbrace{\lambda x. \mathcal{C}_2[M\, \mathcal{C}_1[x_1]]}_{\mathcal{C}[M]} : \lvert (A_1 \to A_2) \rvert$ | By definition of type translation |

$$(A_1' \to A_2') \Rrightarrow (A_1 \to A_2) \hookrightarrow \underbrace{\lambda x. \mathcal{C}_2[[\,]\, \mathcal{C}_1[x]]}_{\mathcal{C}} \quad \text{By rule Coe}\to$$

□

**Theorem 9** (Translation soundness).
*If* $\Gamma \vdash e : A$ *then there exists* $M$ *such that* $\Gamma \vdash e : A \hookrightarrow M$ *and* $\lvert\Gamma\rvert \vdash M : \lvert A \rvert$.

*Proof.* By induction on the structure of the derivation of $\Gamma \vdash e : A$.

- **Case** SVar: Apply rules STVar and TVar.
- **Case** $\dfrac{\Gamma \vdash e : A' \qquad A' \rightsquigarrow A}{\Gamma \vdash e : A}$ SCSub

| | |
|---|---|
| $\Gamma \vdash e : A'$ | Subderivation |
| $\Gamma \vdash e : A' \hookrightarrow M'$ | By the induction hypothesis |
| $\lvert\Gamma\rvert \vdash M' : \lvert A' \rvert$ | " |
| $A' \rightsquigarrow A$ | Given |
| $A' \simeq A$ | By Lemma 17 (Directed consistency obeys Structural Equivalence) |
| $A' \Rrightarrow A \hookrightarrow \mathcal{C}$ | By Theorem 17 |
| $\lvert\Gamma\rvert \vdash \mathcal{C}[M'] : \lvert A \rvert$ | " |
| $\Gamma \vdash e : A \hookrightarrow \mathcal{C}[M']$ | By rule STCSub |

- **Case** SAnno: Use the induction hypothesis and apply rule STAnno.
- **Case** SUnitIntro: Apply rules STUnitIntro and TUnitIntro.
- **Case** SSumIntro: Use the induction hypothesis and apply rules STSumIntro and T+$_i$Intro.
- **Case** SSumElim1: Use the induction hypothesis and apply rules STSumElim1 and T+$_i$Elim.
- **Case** SSumElim2: Use the induction hypothesis and apply rules STSumElim2 and T+Elim.
- **Case** S→Intro: Use the induction hypothesis and apply rules ST→Intro and T→Intro.
- **Case** S→Elim: Use the induction hypothesis and apply rules ST→Elim and T→Elim. □

### D.6.2 Precision

Theorem 11 depends on Lemma 60 (Cast insertion preserves precision), which uses a modified version of the translation that always inserts casts, even safe ones. In effect, the modified translation does not have rule CoeSub and always uses rule CoeCast (Figure 12). It also inserts safe casts $\mathcal{C}_1'$ and $\mathcal{C}_2'$, similar to CoeCase2, in rules *CoeCase1L and *CoeCase1R. See Figure 21.

**Lemma 60** (Cast insertion preserves precision).
*If* $\delta_1' \Rrightarrow \delta_2' \hookrightarrow \mathcal{C}'$ *and* $\delta_1 \Rrightarrow \delta_2 \hookrightarrow \mathcal{C}$
*and* $\delta_1' \sqsubseteq \delta_1$ *and* $\delta_2' \sqsubseteq \delta_2$ *and* $M' \preccurlyeq M$
*then* $\mathcal{C}'[M'] \preccurlyeq \mathcal{C}[M]$.

$\boxed{\delta' \Rightarrow \delta \hookrightarrow \mathcal{C}}$ Coercion $\mathcal{C}$ coerces sum $|\delta'|$ to sum $|\delta|$

$$\frac{\cancel{|\delta'| \not\leq |\delta|}}{\delta' \Rightarrow \delta \hookrightarrow \langle |\delta| \Leftarrow |\delta'| \rangle [\,]}\ {*}\mathsf{CoeCast}$$

$\boxed{A' \Rightarrow A \hookrightarrow \mathcal{C}}$ Coercion $\mathcal{C}$ coerces target type $|A'|$ to $|A|$

$$\frac{}{\mathsf{Unit} \Rightarrow \mathsf{Unit} \hookrightarrow [\,]}\ \mathsf{CoeUnit} \qquad \frac{A_1 \Rightarrow A'_1 \hookrightarrow \mathcal{C}_1 \qquad A'_2 \Rightarrow A_2 \hookrightarrow \mathcal{C}_2}{(A'_1 \to A'_2) \Rightarrow (A_1 \to A_2) \hookrightarrow \lambda x.\, \mathcal{C}_2\big[[\,]\ \mathcal{C}_1[x]\big]}\ \mathsf{Coe}{\to}$$

$$\frac{\begin{matrix}\delta' \in \{+^?_1, +_1\} & +^?_1 \Rightarrow \delta' \hookrightarrow \mathcal{C}'_1 \\ A'_1 \Rightarrow A_1 \hookrightarrow \mathcal{C}_1 & \delta' \Rightarrow \delta \hookrightarrow \mathcal{C}_3\end{matrix}}{\begin{matrix}(A'_1\, \delta'\, A'_2) \Rightarrow (A_1\, \delta\, A_2) \\ \hookrightarrow \mathcal{C}_3\big[\mathsf{case}([\,], \mathsf{inj}_1\, x_1.\mathcal{C}'_1[\mathsf{inj}_1\, \mathcal{C}_1[x_1]])\big]\end{matrix}}\ {*}\mathsf{CoeCase1L} \qquad \frac{\begin{matrix}\delta' \in \{+^?_2, +_2\} & +^?_2 \Rightarrow \delta' \hookrightarrow \mathcal{C}'_2 \\ A'_2 \Rightarrow A_2 \hookrightarrow \mathcal{C}_2 & \delta' \Rightarrow \delta \hookrightarrow \mathcal{C}_3\end{matrix}}{\begin{matrix}(A'_1\, \delta'\, A'_2) \Rightarrow (A_1\, \delta\, A_2) \\ \hookrightarrow \mathcal{C}_3\big[\mathsf{case}([\,], \mathsf{inj}_2\, x_2.\mathcal{C}'_2[\mathsf{inj}_2\, \mathcal{C}_2[x_2]])\big]\end{matrix}}\ {*}\mathsf{CoeCase1R}$$

$$\frac{\begin{matrix} & +^?_1 \Rightarrow \delta' \hookrightarrow \mathcal{C}'_1 & +^?_2 \Rightarrow \delta' \hookrightarrow \mathcal{C}'_2 & \\ \delta' \in \{+^?, +^*_1, +^*_2, +\} & A'_1 \Rightarrow A_1 \hookrightarrow \mathcal{C}_1 & A'_2 \Rightarrow A_2 \hookrightarrow \mathcal{C}_2 & \delta' \Rightarrow \delta \hookrightarrow \mathcal{C}_3\end{matrix}}{(A'_1\, \delta'\, A'_2) \Rightarrow (A_1\, \delta\, A_2)\ \hookrightarrow\ \mathcal{C}_3\big[\mathsf{case}([\,], \mathsf{inj}_1\, x_1.\mathcal{C}'_1[\mathsf{inj}_1\, \mathcal{C}_1[x_1]], \mathsf{inj}_2\, x_2.\mathcal{C}'_2[\mathsf{inj}_2\, \mathcal{C}_2[x_2]])\big]}\ \mathsf{CoeCase2}$$

**Figure 21.** Part of the type-directed translation, modified to insert safe casts; differences highlighted

*Proof.* Note the following reasons for arriving at the result.

(a) If the translated sums are equal, that is, $|\delta'_1| = |\delta_1|$ and $|\delta'_2| = |\delta_2|$, we have $\mathcal{C}' = \mathcal{C}$. (Casts are unique; in this context, this is immediate because we are using a translation that generates casts even if they are safe, so there is only one rule, ${*}\mathsf{CoeCast}$, that derives the judgment.) Then the result follows from $M' \preccurlyeq M$ and the definition of $\preccurlyeq$.

(b) If $\mathcal{C}' = \langle \delta'_2 \Leftarrow |\delta'_1| \rangle [\,]$ and $\mathcal{C} = \langle |\delta_2| \Leftarrow |\delta_1| \rangle [\,]$ and $\langle \delta'_2 \Leftarrow \delta'_1 \rangle \preccurlyeq \langle \delta_2 \Leftarrow \delta_1 \rangle$ then $\mathcal{C}'[M'] \preccurlyeq \mathcal{C}[M]$ by definition of $\preccurlyeq$ as $M' \preccurlyeq M$.

Proceed by case analysis on $\delta'_1 \sqsubseteq \delta_1$ based on the reflexive, transitive closure of precision on sums.

- **Cases** $+_i \sqsubseteq +_i$, $+_i \sqsubseteq +^?_i$, $+_i \sqsubseteq +^*_i$, $+^?_i \sqsubseteq +^?_i$, $+^*_i \sqsubseteq +^*_i$: In these cases, $|\delta'_1| = |\delta_1| = +_i$.
  Proceed by case analysis on $\delta'_2 \sqsubseteq \delta_2$.
  - **Cases** $+_i \sqsubseteq +_i$, $+_i \sqsubseteq +^?_i$, $+_i \sqsubseteq +^*_i$, $+^?_i \sqsubseteq +^?_i$, $+^*_i \sqsubseteq +^*_i$:
    Here, $|\delta'_2| = |\delta_2| = +_i$.
    The translated sums are equal: go to (a) above.
  - **Cases** $+_i \sqsubseteq +^?$, $+^?_i \sqsubseteq +^?$, $+^*_i \sqsubseteq +^?$:
    Here, $|\delta'_2| = +_i$ and $|\delta_2| = +$.
    We have $\langle +_i \Leftarrow +_i \rangle \preccurlyeq \langle + \Leftarrow +_i \rangle$. Go to (b).
  - **Cases** $+ \sqsubseteq +$, $+ \sqsubseteq +^?$, $+^? \sqsubseteq +^?$:
    Here, $|\delta'_2| = |\delta_2| = +$. Go to (a) above.
  - **Cases** $+_k \sqsubseteq +_k$, $+_k \sqsubseteq +^?_k$, $+_k \sqsubseteq +^*_k$, $+^?_k \sqsubseteq +^?_k$, $+^*_k \sqsubseteq +^*_k$:
    Here, $|\delta'_2| = |\delta_2| = +_k$. Go to (a).
  - **Cases** $+_k \sqsubseteq +^?$, $+^?_k \sqsubseteq +^?$, $+^*_k \sqsubseteq +^?$:
    Here, $|\delta'_2| = +_k$ and $|\delta_2| = +$.
    We have $\langle +_k \Leftarrow +_i \rangle \preccurlyeq \langle + \Leftarrow +_i \rangle$. Go to (b).
- **Cases** $+_i \sqsubseteq +^?$, $+^?_i \sqsubseteq +^?$, $+^*_i \sqsubseteq +^?$: In these cases, $|\delta'_1| = +_i$ and $|\delta_1| = +$. Proceed by case analysis on $\delta'_2 \sqsubseteq \delta_2$.
  - **Cases** $+_i \sqsubseteq +_i$, $+_i \sqsubseteq +^?_i$, $+_i \sqsubseteq +^*_i$, $+^?_i \sqsubseteq +^?_i$, $+^*_i \sqsubseteq +^*_i$: We have $\langle +_i \Leftarrow +_i \rangle \preccurlyeq \langle +_i \Leftarrow + \rangle$. Go to (b).
  - **Cases** $+_i \sqsubseteq +^?$, $+^?_i \sqsubseteq +^?$, $+^*_i \sqsubseteq +^?$: We have $\langle +_i \Leftarrow +_i \rangle \preccurlyeq \langle + \Leftarrow + \rangle$. Go to (b).
  - **Cases** $+ \sqsubseteq +$, $+ \sqsubseteq +^?$, $+^? \sqsubseteq +^?$: We have $\langle + \Leftarrow +_i \rangle \preccurlyeq \langle + \Leftarrow + \rangle$. Go to (b).
  - **Cases** $+_k \sqsubseteq +_k$, $+_k \sqsubseteq +^?_k$, $+_k \sqsubseteq +^*_k$, $+^?_k \sqsubseteq +^?_k$, $+^*_k \sqsubseteq +^*_k$: We have $\langle +_k \Leftarrow +_i \rangle \preccurlyeq \langle +_k \Leftarrow + \rangle$. Go to (b).
  - **Cases** $+_k \sqsubseteq +^?$, $+^?_k \sqsubseteq +^?$, $+^*_k \sqsubseteq +^?$: We have $\langle +_k \Leftarrow +_i \rangle \preccurlyeq \langle + \Leftarrow + \rangle$. Go to (b).
- **Cases** $+ \sqsubseteq +$, $+ \sqsubseteq +^?$, $+^? \sqsubseteq +^?$: In these cases, $|\delta'_1| = |\delta_1| = +$. Proceed by case analysis on $\delta'_2 \sqsubseteq \delta_2$.
  - **Cases** $+_i \sqsubseteq +_i$, $+_i \sqsubseteq +^?_i$, $+_i \sqsubseteq +^*_i$, $+^?_i \sqsubseteq +^?_i$, $+^*_i \sqsubseteq +^*_i$: Here, $|\delta'_2| = |\delta_2| = +_i$. Go to (a).
  - **Cases** $+_i \sqsubseteq +^?$, $+^?_i \sqsubseteq +^?$, $+^*_i \sqsubseteq +^?$: We have $\langle +_i \Leftarrow + \rangle \preccurlyeq \langle + \Leftarrow + \rangle$. Go to (b).
  - **Cases** $+ \sqsubseteq +$, $+ \sqsubseteq +^?$, $+^? \sqsubseteq +^?$: Here, $|\delta'_2| = |\delta_2| = +$. Go to (a).
  - **Cases** $+_k \sqsubseteq +_k$, $+_k \sqsubseteq +^?_k$, $+_k \sqsubseteq +^*_k$, $+^?_k \sqsubseteq +^?_k$, $+^*_k \sqsubseteq +^*_k$: Here, $|\delta'_2| = |\delta_2| = +_k$. Go to (a).
  - **Cases** $+_k \sqsubseteq +^?$, $+^?_k \sqsubseteq +^?$, $+^*_k \sqsubseteq +^?$: We have $\langle +_k \Leftarrow + \rangle \preccurlyeq \langle + \Leftarrow + \rangle$. Go to (b). $\square$

**Lemma 61** (Coercion preserves precision)**.**
*If* $A_1' \Rightarrow A_2' \hookrightarrow \mathcal{C}'$ *and* $A_1 \Rightarrow A_2 \hookrightarrow \mathcal{C}$
*and* $A_1' \sqsubseteq A_1$ *and* $A_2' \sqsubseteq A_2$ *and* $M' \preccurlyeq M$
*then* $\mathcal{C}'[M'] \preccurlyeq \mathcal{C}[M]$.

*Proof.* By induction on the structure of the derivation of $A_1' \Rightarrow A_2' \hookrightarrow \mathcal{C}'$.

- **Case**
$$\dfrac{}{\mathsf{Unit} \Rightarrow \mathsf{Unit} \hookrightarrow []}\ \mathsf{CoeUnit}$$

| | |
|---|---|
| $\mathsf{Unit} \sqsubseteq A_1$ | Given |
| $\mathsf{Unit} \sqsubseteq A_2$ | Given |
| $A_1 = \mathsf{Unit}$ | By Lemma 4 (Precision inversion) |
| $A_2 = \mathsf{Unit}$ | By Lemma 4 (Precision inversion) |
| $\mathsf{Unit} \Rightarrow \mathsf{Unit} \hookrightarrow \mathcal{C}$ | Given |
| $\mathcal{C} = []$ | By inversion on $\mathsf{CoeUnit}$ |
| $M' \preccurlyeq M$ | Given |
| $\mathcal{C}'[M'] \preccurlyeq \mathcal{C}[M]$ | By definition of $\mathcal{C}'$ and $\mathcal{C}$ |

- **Case**
$$\dfrac{A_{12}' \Rightarrow A_{11}' \hookrightarrow \mathcal{C}_1' \qquad A_{21}' \Rightarrow A_{22}' \hookrightarrow \mathcal{C}_2'}{(A_{11}' \to A_{21}') \Rightarrow (A_{12}' \to A_{22}') \hookrightarrow \lambda x.\, \mathcal{C}_2'\big[[]\ \mathcal{C}_1'[x]\big]}\ \mathsf{Coe}{\to}$$

| | |
|---|---|
| $A_{11}' \to A_{21}' \sqsubseteq A_1$ | Given |
| $A_1 = A_{11} \to A_{21}$ | By Lemma 4 (Precision inversion) |
| $A_{11}' \sqsubseteq A_{11}$ | $''$ |
| $A_{21}' \sqsubseteq A_{21}$ | $''$ |
| $A_{12}' \to A_{22}' \sqsubseteq A_2$ | Given |
| $A_2 = A_{12} \to A_{22}$ | By Lemma 4 (Precision inversion) |
| $A_{12}' \sqsubseteq A_{12}$ | $''$ |
| $A_{22}' \sqsubseteq A_{22}$ | $''$ |

| | |
|---|---|
| $(A_{11} \to A_{21}) \Rightarrow (A_{12} \to A_{22}) \hookrightarrow \mathcal{C}$ | Given |
| $A_{12} \Rightarrow A_{11} \hookrightarrow \mathcal{C}_1$ | By inversion on $\mathsf{Coe}{\to}$ |
| $A_{21} \Rightarrow A_{22} \hookrightarrow \mathcal{C}_2$ | $''$ |
| $\mathcal{C} = \lambda x.\, \mathcal{C}_2\big[[]\ \mathcal{C}_1[x]\big]$ | $''$ |
| $x \preccurlyeq x$ | By definition of $\preccurlyeq$ |
| $A_{12}' \Rightarrow A_{11}' \hookrightarrow \mathcal{C}_1'$ | Subderivation |
| $\mathcal{C}_1'[x] \preccurlyeq \mathcal{C}_1[x]$ | By the induction hypothesis |
| $M' \preccurlyeq M$ | Given |
| $M'\, \mathcal{C}_1'[x] \preccurlyeq M\, \mathcal{C}_1[x]$ | By definition of $\preccurlyeq$ |
| $A_{21}' \Rightarrow A_{22}' \hookrightarrow \mathcal{C}_2'$ | Subderivation |
| $\mathcal{C}_2'\big[M'\, \mathcal{C}_1'[x]\big] \preccurlyeq \mathcal{C}_2\big[M\, \mathcal{C}_1[x]\big]$ | By the induction hypothesis |
| $\lambda x.\, \mathcal{C}_2'\big[M'\, \mathcal{C}_1'[x]\big] \preccurlyeq \lambda x.\, \mathcal{C}_2\big[M\, \mathcal{C}_1[x]\big]$ | By definition of $\preccurlyeq$ |

- **Case**
$$\dfrac{\begin{array}{ll}\delta_1' \in \{+_1^?, +_1\} & +_1^? \Rightarrow \delta_1' \hookrightarrow \mathcal{C}_{11}' \\ A_{11}' \Rightarrow A_{12}' \hookrightarrow \mathcal{C}_1' & \delta_1' \Rightarrow \delta_2' \hookrightarrow \mathcal{C}_3'\end{array}}{\begin{array}{l}(A_{11}'\ \delta_1'\ A_{21}') \Rightarrow (A_{12}'\ \delta_2'\ A_{22}') \\ \qquad \hookrightarrow \mathcal{C}_3'\big[\mathsf{case}([], \mathtt{inj}_1\ x_1.\mathcal{C}_{11}'[\mathtt{inj}_1\ \mathcal{C}_1'[x_1]])\big]\end{array}}\ \mathsf{CoeCase1L}$$

$$\begin{array}{rl}
A'_{12}\,\delta'_2\,A'_{22} \sqsubseteq A_2 & \text{Given} \\
A_2 = A_{12}\,\delta_2\,A_{22} & \text{By Lemma 4 (Precision inversion)} \\
A'_{12} \sqsubseteq A_{12} & '' \\
A'_{22} \sqsubseteq A_{22} & '' \\
\delta'_2 \sqsubseteq \delta_2 & ''
\end{array}$$

$$\begin{array}{rl}
A'_{11}\,\delta'_1\,A'_{21} \sqsubseteq A_1 & \text{Given} \\
A_1 = A_{11}\,\delta_1\,A_{21} & \text{By Lemma 4 (Precision inversion)} \\
A'_{11} \sqsubseteq A_{11} & '' \\
A'_{21} \sqsubseteq A_{21} & '' \\
\delta'_1 \sqsubseteq \delta_1 & ''
\end{array}$$

Since $\delta'_1 \in \{+^?_1, +_1\}$ and $\delta'_1 \sqsubseteq \delta_1$, by definition of $\sqsubseteq$ it follows that $\delta_1 \in \{+^?_1, +_1, +^*_1, +^?\}$ as well.
Consider the case when $\delta_1 \in \{+^?_1, +_1\}$.

$$\begin{array}{rl}
(A_{11}\,\delta_1\,A_{21}) \Rrightarrow (A_{12}\,\delta_2\,A_{22}) \hookrightarrow \mathcal{C} & \text{Given} \\
A_{11} \Rrightarrow A_{12} \hookrightarrow \mathcal{C}_1 & \text{By inversion on } \mathsf{CoeCase1L} \\
+^?_1 \Rrightarrow \delta_1 \hookrightarrow \mathcal{C}_{11} & '' \\
\delta_1 \Rrightarrow \delta_2 \hookrightarrow \mathcal{C}_3 & '' \\
\mathcal{C} = \mathcal{C}_3\big[\mathtt{case}([\,], \mathtt{inj}_1\,x_1.\mathcal{C}_{11}[\mathtt{inj}_1\,\mathcal{C}_1[x_1]])\big] & ''
\end{array}$$

$$\begin{array}{rl}
x_1 \preccurlyeq x_1 & \text{By definition of } \preccurlyeq \\
A'_{11} \Rrightarrow A'_{12} \hookrightarrow \mathcal{C}'_1 & \text{Subderivation} \\
\mathcal{C}'_1[x_1] \preccurlyeq \mathcal{C}_1[x_1] & \text{By the induction hypothesis} \\
\mathtt{inj}_1\,\mathcal{C}'_1[x_1] \preccurlyeq \mathtt{inj}_1\,\mathcal{C}_1[x_1] & \text{By definition of } \preccurlyeq \\
+^?_1 \Rrightarrow \delta'_1 \hookrightarrow \mathcal{C}'_{11} & \text{Subderivation} \\
+^?_1 \sqsubseteq +^?_1 & \text{By definition of } \sqsubseteq \\
\underbrace{\mathcal{C}'_{11}[\mathtt{inj}_1\,\mathcal{C}'_1[x_1]]}_{M'_1} \preccurlyeq \underbrace{\mathcal{C}_{11}[\mathtt{inj}_1\,\mathcal{C}_1[x_1]]}_{M_1} & \text{By Lemma 60 (Cast insertion preserves precision)} \\
M' \preccurlyeq M & \text{Given} \\
\underbrace{\mathtt{case}(M', \mathtt{inj}_1\,x_1.M'_1)}_{M'_0} \preccurlyeq \underbrace{\mathtt{case}(M, \mathtt{inj}_1\,x_1.M_1)}_{M_0} & \text{By definition of } \preccurlyeq \\
\delta'_1 \Rrightarrow \delta'_2 \hookrightarrow \mathcal{C}'_3 & \text{Subderivation} \\
\mathcal{C}'_3[M'_0] \preccurlyeq \mathcal{C}_3[M_0] & \text{By Lemma 60 (Cast insertion preserves precision)}
\end{array}$$

Consider the case when $\delta_1 \in \{+^*_1, +^?\}$.

$$\begin{array}{rl}
(A_{11}\,\delta_1\,A_{21}) \Rrightarrow (A_{12}\,\delta_2\,A_{22}) \hookrightarrow \mathcal{C} & \text{Given} \\
A_{11} \Rrightarrow A_{12} \hookrightarrow \mathcal{C}_1 & \text{By inversion on } \mathsf{CoeCase2} \\
A_{21} \Rrightarrow A_{22} \hookrightarrow \mathcal{C}_2 & '' \\
\delta_1 \Rrightarrow \delta_2 \hookrightarrow \mathcal{C}_3 & '' \\
+^?_1 \Rrightarrow \delta_1 \hookrightarrow \mathcal{C}_{11} & '' \\
+^?_2 \Rrightarrow \delta_1 \hookrightarrow \mathcal{C}_{21} & ''
\end{array}$$

$$\mathcal{C} = \mathcal{C}_3\big[\mathtt{case}([\,], \mathtt{inj}_1\,x_1.\mathcal{C}_{11}[\mathtt{inj}_1\,\mathcal{C}_1[x_1]], \mathtt{inj}_2\,x_2.\mathcal{C}_{21}[\mathtt{inj}_2\,\mathcal{C}_2[x_2]])\big] \quad ''$$

$$\begin{array}{rl}
x_1 \preccurlyeq x_1 & \text{By definition of } \preccurlyeq \\
A'_{11} \Rrightarrow A'_{12} \hookrightarrow \mathcal{C}'_1 & \text{Subderivation} \\
\mathcal{C}'_1[x_1] \preccurlyeq \mathcal{C}_1[x_1] & \text{By the induction hypothesis} \\
\mathtt{inj}_1\,\mathcal{C}'_1[x_1] \preccurlyeq \mathtt{inj}_1\,\mathcal{C}_1[x_1] & \text{By definition of } \preccurlyeq \\
+^?_1 \Rrightarrow \delta'_1 \hookrightarrow \mathcal{C}'_{11} & \text{Subderivation} \\
+^?_1 \sqsubseteq +^?_1 & \text{By definition of } \sqsubseteq \\
\underbrace{\mathcal{C}'_{11}[\mathtt{inj}_1\,\mathcal{C}'_1[x_1]]}_{M'_1} \preccurlyeq \underbrace{\mathcal{C}_{11}[\mathtt{inj}_1\,\mathcal{C}_1[x_1]]}_{M_1} & \text{By Lemma 60 (Cast insertion preserves precision)}
\end{array}$$

$$\begin{array}{rl}
M' \preccurlyeq M & \text{Given} \\
\underbrace{\mathtt{case}(M', \mathtt{inj}_1\,x_1.M'_1)}_{M'_0} \preccurlyeq \underbrace{\mathtt{case}(M, \mathtt{inj}_1\,x_1.M_1, \mathtt{inj}_2\,x_2.\mathcal{C}_{21}[\mathtt{inj}_2\,\mathcal{C}_2[x_2]])}_{M_0} & \text{By definition of } \preccurlyeq \\
\delta'_1 \Rrightarrow \delta'_2 \hookrightarrow \mathcal{C}'_3 & \text{Subderivation} \\
\mathcal{C}'_3[M'_0] \preccurlyeq \mathcal{C}_3[M_0] & \text{By Lemma 60 (Cast insertion preserves precision)}
\end{array}$$

- **Case** CoeCase1R: Symmetric to the CoeCase1L case.

- **Case**

$$\frac{\delta_1' \in \{+^?, +_1^*, +_2^*, +\} \quad \begin{array}{c} +_1^? \Rightarrow \delta_1' \hookrightarrow \mathcal{C}_{11}' \\ A_{11}' \Rightarrow A_{12}' \hookrightarrow \mathcal{C}_1' \end{array} \quad \begin{array}{c} +_2^? \Rightarrow \delta_1' \hookrightarrow \mathcal{C}_{21}' \\ A_{21}' \Rightarrow A_{22}' \hookrightarrow \mathcal{C}_2' \end{array} \quad \delta_1' \Rightarrow \delta_2' \hookrightarrow \mathcal{C}_3'}{(A_{11}'\,\delta_1'\,A_{21}') \Rightarrow (A_{12}'\,\delta_2'\,A_{22}') \;\hookrightarrow\; \mathcal{C}_3'\big[\mathsf{case}([\,], \mathsf{inj}_1\,x_1.\mathcal{C}_{11}'[\mathsf{inj}_1\,\mathcal{C}_1'[x_1]], \mathsf{inj}_2\,x_2.\mathcal{C}_{21}'[\mathsf{inj}_2\,\mathcal{C}_2'[x_2]])\big]}\ \mathsf{CoeCase2}$$

| | |
|---|---|
| $A_{12}'\,\delta_2'\,A_{22}' \sqsubseteq A_2$ | Given |
| $A_2 = A_{12}\,\delta_2\,A_{22}$ | By Lemma 4 (Precision inversion) |
| $A_{12}' \sqsubseteq A_{12}$ | " |
| $A_{22}' \sqsubseteq A_{22}$ | " |
| $\delta_2' \sqsubseteq \delta_2$ | " |

| | |
|---|---|
| $A_{11}'\,\delta_1'\,A_{21}' \sqsubseteq A_1$ | Given |
| $A_1 = A_{11}\,\delta_1\,A_{21}$ | By Lemma 4 (Precision inversion) |
| $A_{11}' \sqsubseteq A_{11}$ | " |
| $A_{21}' \sqsubseteq A_{21}$ | " |
| $\delta_1' \sqsubseteq \delta_1$ | " |

Since $\delta_1' \in \{+^?, +_1^*, +_2^*, +\}$ and $\delta_1' \sqsubseteq \delta_1$, by definition of $\sqsubseteq$ it follows that $\delta_1 \in \{+^?, +_1^*, +_2^*, +\}$ as well.

| | |
|---|---|
| $(A_{11}\,\delta_1\,A_{21}) \Rightarrow (A_{12}\,\delta_2\,A_{22}) \hookrightarrow \mathcal{C}$ | Given |
| $A_{11} \Rightarrow A_{12} \hookrightarrow \mathcal{C}_1$ | By inversion on CoeCase2 |
| $A_{21} \Rightarrow A_{22} \hookrightarrow \mathcal{C}_2$ | " |
| $\delta_1 \Rightarrow \delta_2 \hookrightarrow \mathcal{C}_3$ | " |
| $+_1^? \Rightarrow \delta_1 \hookrightarrow \mathcal{C}_{11}$ | " |
| $+_2^? \Rightarrow \delta_1 \hookrightarrow \mathcal{C}_{21}$ | " |

$\mathcal{C} = \mathcal{C}_3\big[\mathsf{case}([\,], \mathsf{inj}_1\,x_1.\mathcal{C}_{11}[\mathsf{inj}_1\,\mathcal{C}_1[x_1]], \mathsf{inj}_2\,x_2.\mathcal{C}_{21}[\mathsf{inj}_2\,\mathcal{C}_2[x_2]])\big]$ "

| | |
|---|---|
| $x_1 \preccurlyeq x_1$ | By definition of $\preccurlyeq$ |
| $A_{11}' \Rightarrow A_{12}' \hookrightarrow \mathcal{C}_1'$ | Subderivation |
| $\mathcal{C}_1'[x_1] \preccurlyeq \mathcal{C}_1[x_1]$ | By the induction hypothesis |
| $\mathsf{inj}_1\,\mathcal{C}_1'[x_1] \preccurlyeq \mathsf{inj}_1\,\mathcal{C}_1[x_1]$ | By definition of $\preccurlyeq$ |
| $+_1^? \Rightarrow \delta_1' \hookrightarrow \mathcal{C}_{11}'$ | Subderivation |
| $+_1^? \sqsubseteq +_1^?$ | By definition of $\sqsubseteq$ |
| $\underbrace{\mathcal{C}_{11}'[\mathsf{inj}_1\,\mathcal{C}_1'[x_1]]}_{M_1'} \preccurlyeq \underbrace{\mathcal{C}_{11}[\mathsf{inj}_1\,\mathcal{C}_1[x_1]]}_{M_1}$ | By Lemma 60 (Cast insertion preserves precision) |

| | |
|---|---|
| $x_2 \preccurlyeq x_2$ | By definition of $\preccurlyeq$ |
| $A_{21}' \Rightarrow A_{22}' \hookrightarrow \mathcal{C}_2'$ | Subderivation |
| $\mathcal{C}_2'[x_2] \preccurlyeq \mathcal{C}_2[x_2]$ | By the induction hypothesis |
| $\mathsf{inj}_2\,\mathcal{C}_2'[x_2] \preccurlyeq \mathsf{inj}_2\,\mathcal{C}_2[x_2]$ | By definition of $\preccurlyeq$ |
| $+_2^? \Rightarrow \delta_1' \hookrightarrow \mathcal{C}_{21}'$ | Subderivation |
| $+_2^? \sqsubseteq +_2^?$ | By definition of $\sqsubseteq$ |
| $\underbrace{\mathcal{C}_{21}'[\mathsf{inj}_2\,\mathcal{C}_2'[x_2]]}_{M_2'} \preccurlyeq \underbrace{\mathcal{C}_{21}[\mathsf{inj}_2\,\mathcal{C}_2[x_2]]}_{M_2}$ | By Lemma 60 (Cast insertion preserves precision) |

| | |
|---|---|
| $M' \preccurlyeq M$ | Given |
| $\underbrace{\mathsf{case}(M', \mathsf{inj}_1\,x_1.M_1', \mathsf{inj}_2\,x_2.M_2')}_{M_0'} \preccurlyeq \underbrace{\mathsf{case}(M, \mathsf{inj}_1\,x_1.M_1, \mathsf{inj}_2\,x_2.M_2)}_{M_0}$ | By definition of $\preccurlyeq$ |

| | |
|---|---|
| $\delta_1' \Rightarrow \delta_2' \hookrightarrow \mathcal{C}_3'$ | Subderivation |
| $\mathcal{C}_3'[M_0'] \preccurlyeq \mathcal{C}_3[M_0]$ | By Lemma 60 (Cast insertion preserves precision) |

∎

**Theorem 11** (Translation preserves precision)**.**
*Suppose* $\Gamma' \sqsubseteq \Gamma$ *and* $e' \sqsubseteq e$.

1. *If* $\Gamma' \vdash e' \Leftarrow A'$ *and* $\Gamma \vdash e \Leftarrow A$ *and* $A' \sqsubseteq A$ *then* $\Gamma' \vdash e' : A' \hookrightarrow M'$ *and* $\Gamma \vdash e : A \hookrightarrow M$ *where* $M' \preccurlyeq M$.
2. *If* $\Gamma' \vdash e' \Rightarrow A'$ *and* $\Gamma \vdash e \Rightarrow A$ *then* $\Gamma' \vdash e' : A' \hookrightarrow M'$ *and* $\Gamma \vdash e : A \hookrightarrow M$ *where* $A' \sqsubseteq A$ *and* $M' \preccurlyeq M$.

*Proof.* By induction on the structure of the derivation of $\Gamma' \vdash e' \Leftarrow A'$ (part 1) or $\Gamma' \vdash e' \Rightarrow A'$ (part 2).

- **Case** $\dfrac{\Gamma'(x) = A'}{\Gamma' \vdash x \Rightarrow A'}$ SynVar

| | | |
|---|---|---|
| | $x \sqsubseteq e$ | Given |
| | $e = x$ | From definition of $\sqsubseteq$ |
| | $\Gamma \vdash x \Leftarrow A$ | Given |
| | $\Gamma(x) = A$ | By inversion on SynVar |
| | $\Gamma'(x) = A'$ | Premise |
| ☞ | $\Gamma' \vdash x : A' \hookrightarrow x$ | By rule STVar |
| ☞ | $\Gamma \vdash x : A \hookrightarrow x$ | By rule STVar |
| ☞ | $x \preccurlyeq x$ | By definition of $\preccurlyeq$ |

- **Case** $\dfrac{\Gamma' \vdash e' \Rightarrow A'_0 \qquad A'_0 \rightsquigarrow A'}{\Gamma' \vdash e' \Leftarrow A'}$ ChkCSub

By inversion on $\Gamma \vdash e \Leftarrow A$, rule ChkCSub was applied.

| | | |
|---|---|---|
| | $\Gamma \vdash e \Rightarrow A_0$ | By inversion on ChkCSub |
| | $A_0 \rightsquigarrow A$ | ″ |
| | $\Gamma' \vdash e' \Rightarrow A'_0$ | Subderivation |
| | $\Gamma' \vdash e' : A'_0 \hookrightarrow M'_0$ | By the induction hypothesis |
| | $\Gamma \vdash e : A_0 \hookrightarrow M_0$ | ″ |
| | $A'_0 \sqsubseteq A_0$ | ″ |
| | $M'_0 \preccurlyeq M_0$ | ″ |
| | $A'_0 \rightsquigarrow A'$ | Subderivation |
| | $A'_0 \simeq A'$ | By Lemma 17 (Directed consistency obeys Structural Equivalence) |
| | $A_0 \simeq A$ | By Lemma 17 (Directed consistency obeys Structural Equivalence) |
| | $A'_0 \Rightarrow A' \hookrightarrow \mathcal{C}'$ | By Theorem 17 |
| | $A_0 \Rightarrow A \hookrightarrow \mathcal{C}$ | By Theorem 17 |
| | $A' \sqsubseteq A$ | Given |
| ☞ | $\mathcal{C}'[M'_0] \preccurlyeq \mathcal{C}[M_0]$ | By Lemma 61 (Coercion preserves precision) |
| ☞ | $\Gamma' \vdash e' : A' \hookrightarrow \mathcal{C}'[M'_0]$ | By rule STCSub |
| ☞ | $\Gamma \vdash e : A \hookrightarrow \mathcal{C}[M_0]$ | By rule STCSub |

- **Case** $\dfrac{\Gamma' \vdash e'_0 \Leftarrow A'}{\Gamma' \vdash (e'_0 :: A') \Rightarrow A'}$ SynAnno

| | | |
|---|---|---|
| | $(e'_0 :: A') \sqsubseteq e$ | Given |
| | $e = (e_0 :: A)$ | From definition of $\sqsubseteq$ |
| | $e'_0 \sqsubseteq e_0$ | ″ |
| ☞ | $A' \sqsubseteq A$ | ″ |
| | $\Gamma \vdash (e_0 :: A) \Rightarrow A$ | Given |
| | $\Gamma \vdash e_0 \Leftarrow A$ | By inversion on rule SynAnno |
| | $\Gamma' \sqsubseteq \Gamma$ | Given |
| | $\Gamma' \vdash e'_0 \Leftarrow A'$ | Subderivation |
| | $\Gamma' \vdash e'_0 : A' \hookrightarrow M'$ | By the induction hypothesis |
| | $\Gamma \vdash e_0 : A \hookrightarrow M$ | ″ |
| ☞ | $M' \preccurlyeq M$ | ″ |
| ☞ | $\Gamma' \vdash (e'_0 :: A') : A' \hookrightarrow M'$ | By rule STAnno |
| ☞ | $\Gamma \vdash (e_0 :: A) : A \hookrightarrow M$ | By rule STAnno |

- **Case** $\dfrac{}{\Gamma' \vdash () \Leftarrow \mathsf{Unit}}$ ChkUnitIntro

| | |
|---|---|
| $() \sqsubseteq e$ | Given |
| $e = ()$ | From definition of $\sqsubseteq$ |
| $\Gamma \vdash () \Leftarrow A$ | Given |
| $A = \mathsf{Unit}$ | By inversion on ChkUnitIntro |
| ☞ $\Gamma' \vdash () : \mathsf{Unit} \hookrightarrow ()$ | By rule STUnitIntro |
| ☞ $\Gamma \vdash () : \mathsf{Unit} \hookrightarrow ()$ | By rule STUnitIntro |
| ☞ $() \preccurlyeq ()$ | By definition of $\preccurlyeq$ |

- **Case** $\dfrac{\Gamma' \vdash e_0' \Leftarrow A_i' \qquad +_i^? \leq \delta'}{\Gamma' \vdash (\mathbf{inj}_i\, e_0') \Leftarrow (A_1'\, \delta'\, A_2')}$ ChkSumIntro

| | |
|---|---|
| $\mathbf{inj}_i\, e_0' \sqsubseteq e$ | Given |
| $e = \mathbf{inj}_i\, e_0$ | From definition of $\sqsubseteq$ |
| $e_0' \sqsubseteq e_0$ | $''$ |
| $\Gamma \vdash (\mathbf{inj}_i\, e_0) \Leftarrow A$ | Given |
| $\Gamma \vdash e_0 \Leftarrow A_i$ | By inversion on ChkSumIntro |
| $A = A_1\, \delta\, A_2$ | $''$ |
| $+_i^? \leq \delta$ | $''$ |
| $A_1'\, \delta'\, A_2' \sqsubseteq A_1\, \delta\, A_2$ | Given |
| $A_1' \sqsubseteq A_1$ | From definition of $\sqsubseteq$ |
| $A_2' \sqsubseteq A_2$ | $''$ |
| $+_i^? \sqsubseteq +_i^?$ | By definition of $\sqsubseteq$ |
| $A_1' +_i^? A_2' \sqsubseteq A_1 +_i^? A_2$ | By definition of $\sqsubseteq$ |
| $\Gamma' \vdash e_0' \Leftarrow A_i'$ | Subderivation |
| $\Gamma' \vdash e_0' : A_i' \hookrightarrow M_0'$ | By the induction hypothesis |
| $\Gamma \vdash e_0 : A_i \hookrightarrow M_0$ | $''$ |
| $M_0' \preccurlyeq M_0$ | $''$ |
| $\Gamma' \vdash (\mathbf{inj}_i\, e_0') : (A_1' +_i^? A_2') \hookrightarrow (\mathbf{inj}_i\, M_0')$ | By rule STSumIntro |
| $\Gamma \vdash (\mathbf{inj}_i\, e_0) : (A_1 +_i^? A_2) \hookrightarrow (\mathbf{inj}_i\, M_0)$ | By rule STSumIntro |
| $\mathbf{inj}_i\, M_0' \preccurlyeq \mathbf{inj}_i\, M_0$ | By definition of $\preccurlyeq$ |

| | |
|---|---|
| $A_1' \leq A_1'$ | By Lemma 2 (Reflexivity of subtyping) |
| $A_1 \leq A_1$ | By Lemma 2 (Reflexivity of subtyping) |
| $A_2' \leq A_2'$ | By Lemma 2 (Reflexivity of subtyping) |
| $A_2 \leq A_2$ | By Lemma 2 (Reflexivity of subtyping) |
| $A_1' +_i^? A_2' \leq A_1'\, \delta'\, A_2'$ | By definition of $\leq$ |
| $A_1 +_i^? A_2 \leq A_1\, \delta\, A_2$ | By definition of $\leq$ |
| $A_1' +_i^? A_2' \rightsquigarrow A_1'\, \delta'\, A_2'$ | By Lemma 8 (Subtyping obeys directed consistency) |
| $A_1 +_i^? A_2 \rightsquigarrow A_1\, \delta\, A_2$ | By Lemma 8 (Subtyping obeys directed consistency) |

| | |
|---|---|
| $A_1' +_i^? A_2' \simeq A_1'\, \delta'\, A_2'$ | By Lemma 17 (Directed consistency obeys Structural Equivalence) |
| $A_1 +_i^? A_2 \simeq A_1\, \delta\, A_2$ | By Lemma 17 (Directed consistency obeys Structural Equivalence) |
| $A_1' +_i^? A_2' \Rightarrow A_1'\, \delta'\, A_2' \hookrightarrow \mathcal{C}'$ | By Theorem 17 |
| $A_1 +_i^? A_2 \Rightarrow A_1\, \delta\, A_2 \hookrightarrow \mathcal{C}$ | By Theorem 17 |
| ☞ $\Gamma' \vdash (\mathbf{inj}_i\, e_0') : (A_1'\, \delta'\, A_2') \hookrightarrow \mathcal{C}'[\mathbf{inj}_i\, M_0']$ | By rule STCSub |
| ☞ $\Gamma \vdash (\mathbf{inj}_i\, e_0) : (A_1\, \delta\, A_2) \hookrightarrow \mathcal{C}[\mathbf{inj}_i\, M_0]$ | By rule STCSub |
| ☞ $\mathcal{C}'[\mathbf{inj}_i\, M_0'] \preccurlyeq \mathcal{C}[\mathbf{inj}_i\, M_0]$ | By Lemma 61 (Coercion preserves precision) |

- **Case** $\dfrac{\begin{array}{l}\Gamma' \vdash e_0' \Rightarrow (A_1'\, \delta'\, A_2') \\ \delta' \Rrightarrow +_i^* \qquad\qquad \Gamma', x : A_i' \vdash e_i' \Leftarrow A'\end{array}}{\Gamma' \vdash \mathsf{case}(e_0', \mathbf{inj}_i\, x.e_i') \Leftarrow A'}$ ChkSumElim1

$$
\begin{array}{rl}
\mathsf{case}(e_0', \mathsf{inj}_i\, x.e_i') \sqsubseteq e & \text{Given} \\
e = \mathsf{case}(e_0, \mathsf{inj}_i\, x.e_i) & \text{From definition of } \sqsubseteq \\
e_0' \sqsubseteq e_0 & '' \\
e_i' \sqsubseteq e_i & '' \\
\\
\Gamma \vdash \mathsf{case}(e_0, \mathsf{inj}_i\, x.e_i) \Leftarrow A & \text{Given} \\
\Gamma \vdash e_0 \Rightarrow (A_1\, \delta\, A_2) & \text{By inversion on } \mathsf{ChkSumElim1} \\
\Gamma, x : A_i \vdash e_i \Leftarrow A & '' \\
\delta \Rrightarrow +_i^* & '' \\
\\
\Gamma' \sqsubseteq \Gamma & \text{Given} \\
\Gamma' \vdash e_0' \Rightarrow (A_1'\, \delta'\, A_2') & \text{Subderivation} \\
\Gamma' \vdash e_0' : (A_1'\, \delta'\, A_2') \hookrightarrow M_0' & \text{By the induction hypothesis} \\
\Gamma \vdash e_0 : (A_1\, \delta\, A_2) \hookrightarrow M_0 & '' \\
A_1'\, \delta'\, A_2' \sqsubseteq A_1\, \delta\, A_2 & '' \\
M_0' \preccurlyeq M_0 & '' \\
\\
A_1' \sqsubseteq A_1 & \text{From definition of } \sqsubseteq \\
A_2' \sqsubseteq A_2 & '' \\
+_i^* \sqsubseteq +_i^* & \text{By definition of } \sqsubseteq \\
A_1' +_i^* A_2' \sqsubseteq A_1 +_i^* A_2 & \text{By definition of } \sqsubseteq \\
\\
\delta' \Rrightarrow +_i^* & \text{Subderivation} \\
\delta' \leq +_i^* & \text{By Lemma 24 } (\Rrightarrow \text{ implies subsum}) \\
\delta \leq +_i^* & \text{By Lemma 24 } (\Rrightarrow \text{ implies subsum}) \\
A_1' \leq A_1' & \text{By Lemma 2 (Reflexivity of subtyping)} \\
A_1 \leq A_1 & \text{By Lemma 2 (Reflexivity of subtyping)} \\
A_2' \leq A_2' & \text{By Lemma 2 (Reflexivity of subtyping)} \\
A_2 \leq A_2 & \text{By Lemma 2 (Reflexivity of subtyping)} \\
A_1'\, \delta'\, A_2' \leq A_1' +_i^* A_2' & \text{By definition of } \leq \\
A_1\, \delta\, A_2 \leq A_1 +_i^* A_2 & \text{By definition of } \leq \\
A_1'\, \delta'\, A_2' \rightsquigarrow A_1' +_i^* A_2' & \text{By Lemma 8 (Subtyping obeys directed consistency)} \\
A_1\, \delta\, A_2 \rightsquigarrow A_1 +_i^* A_2 & \text{By Lemma 8 (Subtyping obeys directed consistency)} \\
\\
A_1'\, \delta'\, A_2' \simeq A_1' +_i^* A_2' & \text{By Lemma 17 (Directed consistency obeys Structural Equivalence)} \\
A_1\, \delta\, A_2 \simeq A_1 +_i^* A_2 & \text{By Lemma 17 (Directed consistency obeys Structural Equivalence)} \\
A_1'\, \delta'\, A_2' \Rightarrow A_1' +_i^* A_2' \hookrightarrow \mathcal{C}' & \text{By Theorem 17} \\
A_1\, \delta\, A_2 \Rightarrow A_1 +_i^* A_2 \hookrightarrow \mathcal{C} & \text{By Theorem 17} \\
\Gamma' \vdash e_0' : (A_1' +_i^* A_2') \hookrightarrow \mathcal{C}'[M_0'] & \text{By rule } \mathsf{STCSub} \\
\Gamma \vdash e_0 : (A_1 +_i^* A_2) \hookrightarrow \mathcal{C}[M_0] & \text{By rule } \mathsf{STCSub} \\
\mathcal{C}'[M_0'] \preccurlyeq \mathcal{C}[M_0] & \text{By Lemma 61 (Coercion preserves precision)} \\
\\
A' \sqsubseteq A & \text{Given} \\
\Gamma', x : A_i' \sqsubseteq \Gamma, x : A_i & \text{By definition of } \sqsubseteq \\
\Gamma', x : A_i' \vdash e_i' \Leftarrow A' & \text{Subderivation} \\
\Gamma', x : A_i' \vdash e_i' : A' \hookrightarrow M_i' & \text{By the induction hypothesis} \\
\Gamma, x : A_i \vdash e_i : A \hookrightarrow M_i & '' \\
M_i' \preccurlyeq M_i & ''
\end{array}
$$

$$
\begin{array}{lll}
☞ & \Gamma' \vdash e' : A' \hookrightarrow \underbrace{\mathsf{case}(\mathcal{C}'[M_0'], \mathsf{inj}_i\, x.M_i')}_{M'} & \text{By rule } \mathsf{STSumElim1} \\
☞ & \Gamma \vdash e : A \hookrightarrow \underbrace{\mathsf{case}(\mathcal{C}[M_0], \mathsf{inj}_i\, x.M_i)}_{M} & \text{By rule } \mathsf{STSumElim1} \\
☞ & M' \preccurlyeq M & \text{By definition of } \preccurlyeq
\end{array}
$$

- **Case** ChkSumElim2: Similar to the ChkSumElim1 case, hence omitted.

- **Case** $\dfrac{\Gamma', x : A_1' \vdash e_0' \Leftarrow A_2'}{\Gamma' \vdash (\lambda x.\, e_0') \Leftarrow (A_1' \to A_2')}\ \mathsf{Chk{\to}Intro}$

$$
\begin{array}{ll}
\lambda x.\, e_0' \sqsubseteq e & \text{Given} \\
e = \lambda x.\, e_0 & \text{From definition of } \sqsubseteq \\
e_0' \sqsubseteq e_0 & '' \\
\\
\Gamma \vdash (\lambda x.\, e_0) \Leftarrow A & \text{Given} \\
\Gamma, x : A_1 \vdash e_0 \Leftarrow A_2 & \text{By inversion on Chk}{\rightarrow}\text{Intro} \\
A = A_1 \rightarrow A_2 & ''
\end{array}
$$

$$
\begin{array}{ll}
A_1' \rightarrow A_2' \sqsubseteq A_1 \rightarrow A_2 & \text{Given} \\
A_1' \sqsubseteq A_1 & \text{From definition of } \sqsubseteq \\
A_2' \sqsubseteq A_2 & '' \\
\\
\Gamma' \sqsubseteq \Gamma & \text{Given} \\
\Gamma', x : A_1' \sqsubseteq \Gamma, x : A_1 & \text{By definition of } \sqsubseteq \\
\Gamma', x : A_1' \vdash e_0' \Leftarrow A_2' & \text{Subderivation} \\
\Gamma', x : A_1' \vdash e_0' : A_2' \hookrightarrow M_0' & \text{By the induction hypothesis} \\
\Gamma, x : A_1 \vdash e_0 : A_2 \hookrightarrow M_0 & '' \\
M_0' \preccurlyeq M_0 & ''
\end{array}
$$

$$
\begin{array}{lll}
☞ & \Gamma' \vdash (\lambda x.\, e_0') : (A_1' \rightarrow A_2') \hookrightarrow (\lambda x.\, M_0') & \text{By rule ST}{\rightarrow}\text{Intro} \\
☞ & \Gamma \vdash (\lambda x.\, e_0) : (A_1 \rightarrow A_2) \hookrightarrow (\lambda x.\, M_0) & \text{By rule ST}{\rightarrow}\text{Intro} \\
☞ & \lambda x.\, M_0' \preccurlyeq \lambda x.\, M_0 & \text{By definition of } \preccurlyeq
\end{array}
$$

- **Case** $\dfrac{\Gamma' \vdash e_1' \Rightarrow (A_0' \rightarrow A') \qquad \Gamma' \vdash e_2' \Leftarrow A_0'}{\Gamma' \vdash (e_1'\, e_2') \Rightarrow A'}$ Syn$\rightarrow$Elim

$$
\begin{array}{ll}
e_1'\, e_2' \sqsubseteq e & \text{Given} \\
e = e_1\, e_2 & \text{From definition of } \sqsubseteq \\
e_1' \sqsubseteq e_1 & '' \\
e_2' \sqsubseteq e_2 & '' \\
\\
\Gamma \vdash (e_1\, e_2) \Leftarrow A & \text{Given} \\
\Gamma \vdash e_1 \Rightarrow (A_0 \rightarrow A) & \text{By inversion on Syn}{\rightarrow}\text{Elim} \\
\Gamma \vdash e_2 \Leftarrow A_0 & '' \\
\\
\Gamma' \sqsubseteq \Gamma & \text{Given} \\
\Gamma' \vdash e_1' \Rightarrow (A_0' \rightarrow A') & \text{Subderivation} \\
\Gamma' \vdash e_1' : (A_0' \rightarrow A') \hookrightarrow M_1' & \text{By the induction hypothesis} \\
\Gamma \vdash e_1 : (A_0 \rightarrow A) \hookrightarrow M_1 & '' \\
A_0' \rightarrow A' \sqsubseteq A_0 \rightarrow A & '' \\
M_1' \preccurlyeq M_1 & ''
\end{array}
$$

$$
\begin{array}{lll}
☞ & A' \sqsubseteq A & \text{From definition of } \sqsubseteq \\
& A_0' \sqsubseteq A_0 & '' \\
& \Gamma' \vdash e_2' \Leftarrow A_0' & \text{Subderivation} \\
& \Gamma' \vdash e_2' : A_0' \hookrightarrow M_2' & \text{By the induction hypothesis} \\
& \Gamma \vdash e_2 : A_0 \hookrightarrow M_2 & '' \\
& M_2' \preccurlyeq M_2 & '' \\
\\
☞ & \Gamma' \vdash (e_1'\, e_2') : A' \hookrightarrow (M_1'\, M_2') & \text{By rule ST}{\rightarrow}\text{Intro} \\
☞ & \Gamma \vdash (e_1\, e_2) : A \hookrightarrow (M_1\, M_2) & \text{By rule ST}{\rightarrow}\text{Intro} \\
☞ & M_1'\, M_2' \preccurlyeq M_1\, M_2 & \text{By definition of } \preccurlyeq
\end{array}
$$

□

### D.7 Static programs don't go wrong

We write $\Gamma|_V$ for $\Gamma$ restricted to the set of variables $V$.

**Theorem 18** (Static programs don't go wrong)**.**
*If $\Gamma \vdash e \Leftarrow A$ by a static derivation then $\Gamma|_{FV(e)} \vdash e : A \hookrightarrow M$ and, for all $M'$ such that $M \mapsto^* M'$, it is the case that $M'$ `free`.*

*Proof.* Apply Theorem 19 and Theorem 10 to show $M$ `free`.
The result follows by induction on the number of steps in $M \mapsto^* M'$, using Theorem 8. □

#### D.7.1 Static derivations

**Definition 2.** *We say that a derivation of $\Gamma \vdash e \Leftarrow A$ or $\Gamma \vdash e \Rightarrow A$ is a* static derivation *if, for all subderivations deriving checking or synthesis judgments, the types checked or synthesized are static.*

*Note.* If a derivation is static, then all of its subderivations must be static.

**Lemma 62** (Context thinning)**.**
*If* $y \notin FV(e)$ *then:*

1. *If* $\Gamma, y : A' \vdash e \Leftarrow A$ *then* $\Gamma \vdash e \Leftarrow A$.
2. *If* $\Gamma, y : A' \vdash e \Rightarrow A$ *then* $\Gamma \vdash e \Rightarrow A$.

*Proof.* By induction on the structure of the given derivation.

- **Case** $\dfrac{(\Gamma, y : A')(x) = A}{\Gamma, y : A' \vdash x \Rightarrow A}$ SynVar

$$\begin{array}{rl} y \neq x & \text{Since } y \notin FV(x) \\ (\Gamma, y : A')(x) = A & \text{Premise} \\ \Gamma(x) = A & \text{By definition} \\ \Gamma \vdash x \Rightarrow A & \text{By rule SynVar} \end{array}$$

- **Case** ChkCSub: Use the induction hypothesis and apply rule ChkCSub.
- **Case** SynAnno: Use the induction hypothesis, and apply rule SynAnno.
- **Case** ChkUnitIntro: Apply rule ChkUnitIntro.
- **Case** ChkSumIntro: Use the induction hypothesis, the definition of $FV(-)$, and apply rule ChkSumIntro.
- **Case** ChkSumElim1: Use the induction hypothesis, the definition of $FV(-)$, and apply rule ChkSumElim1.
- **Case** ChkSumElim2: Use the induction hypothesis, the definition of $FV(-)$, and apply rule ChkSumElim2.
- **Case** Chk$\rightarrow$Intro: Use the induction hypothesis, the definition of $FV(-)$, and apply rule Chk$\rightarrow$Intro.
- **Case** Syn$\rightarrow$Elim: Use the induction hypothesis, the definition of $FV(-)$, and apply rule Syn$\rightarrow$Elim. □

**Corollary 63** (Context support)**.**

1. *If* $\Gamma \vdash e \Leftarrow A$ *then* $\Gamma|_{FV(e)} \vdash e \Leftarrow A$.
2. *If* $\Gamma \vdash e \Rightarrow A$ *then* $\Gamma|_{FV(e)} \vdash e \Rightarrow A$.

*Proof.* By induction on $\left|\mathsf{dom}(\Gamma) \setminus FV(e)\right|$.
If $\mathsf{dom}(\Gamma) = FV(e)$, then $\Gamma = \Gamma|_{FV(e)}$ so we already have the result.
Otherwise, use the induction hypothesis, and apply Lemma 62 (Context thinning). □

**Theorem 19** (Static subformula)**.**

1. *If* $\Gamma \vdash e \Leftarrow A$ *by a* static derivation *then* $\Gamma^S \vdash e^S \Leftarrow A^S$ *where* $\Gamma^S = \Gamma|_{FV(e)}$, $e^S = e$, *and* $A^S = A$.
2. *If* $\Gamma \vdash e \Rightarrow A$ *by a* static derivation *then* $\Gamma^S \vdash e^S \Rightarrow A^S$ *where* $\Gamma^S = \Gamma|_{FV(e)}$, $e^S = e$, *and* $A^S = A$.

*Proof.* By induction on the height of the given derivation.
Since $\Gamma \vdash e \Leftarrow A$ and $\Gamma \vdash e \Rightarrow A$ by static derivations, all occurrences of types in checking and synthesizing positions are static, including $A$. Therefore, $A^S = A$ already holds.
Applying Corollary 63 individually to $\Gamma \vdash e \Leftarrow A^S$ and $\Gamma \vdash e \Rightarrow A^S$ produces the derivations $\Gamma|_{FV(e)} \vdash e \Leftarrow A^S$ and $\Gamma|_{FV(e)} \vdash e \Rightarrow A^S$ respectively.
Note that $\Gamma|_{FV(e)} \vdash e \Leftarrow A^S$ and $\Gamma|_{FV(e)} \vdash e \Rightarrow A^S$ are also static derivations.
All cases are then immediate by the induction hypothesis and applying the relevant rule. □

### D.7.2 Static translations are free of casts and match failures

*Notation.* We write $M$ `free` to denote that the target term $M$ contains no casts or `matchfail`s.

**Lemma 64** (Subsums don't need casts)**.**

1. *If* $+^?_i \leq \delta^S$ *and* $+^?_i \Rightarrow \delta^S \hookrightarrow \mathcal{C}$ *then* $\mathcal{C} = []$.
2. *If* $+_i \leq +^*_i$ *and* $+_i \Rightarrow +^*_i \hookrightarrow \mathcal{C}$ *then* $\mathcal{C} = []$.

*Proof.*

1. From definition of subtyping, it is either the case that $\delta^S = +_i$ or $\delta^S = +$. In both cases, by definition of subtyping, $|+^?_i| = +_i \leq |\delta^S|$. By inversion on $+^?_i \Rightarrow \delta^S \hookrightarrow \mathcal{C}$, either rule CoeSub or CoeCast was applied. If rule CoeCast was applied then $+_i \not\leq |\delta^S|$, a contradiction. If rule CoeSub was applied, then indeed $\mathcal{C} = []$.
2. By definition of subtyping, $|+_i| = +_i \leq +_i = |+^*_i|$. By inversion on $+_i \Rightarrow +^*_i \hookrightarrow \mathcal{C}$, either rule CoeSub or CoeCast was applied. If rule CoeCast was applied then $+_i \not\leq +_i$, a contradiction. If rule CoeSub was applied, then indeed $\mathcal{C} = []$.

□

**Lemma 65** (Gradual sums in static don't need casts)**.**

1. *If* $A^S_{11} +^?_i A^S_{21} \le A^S_{12}\, \delta^S\, A^S_{22}$ *and* $A^S_{11} +^?_i A^S_{21} \Rightarrow A^S_{12}\, \delta^S\, A^S_{22} \hookrightarrow \mathcal{C}$ *and* $M$ free *then* $\mathcal{C}[M]$ free.
2. *If* $A^S_{11} +_i A^S_{21} \le A^S_{12} +^*_i A^S_{22}$ *and* $A^S_{11} +_i A^S_{21} \Rightarrow A^S_{12} +^*_i A^S_{22} \hookrightarrow \mathcal{C}$ *and* $M$ free *then* $\mathcal{C}[M]$ free.

*Proof.*

1.

| | |
|---|---|
| $A^S_{11} +^?_i A^S_{21} \Rightarrow A^S_{12}\, \delta^S\, A^S_{22} \hookrightarrow \mathcal{C}$ | Given |
| $A^S_{i1} \Rightarrow A^S_{i2} \hookrightarrow \mathcal{C}_i$ | By inversion on CoeCase1L or CoeCase1R |
| $+^?_i \Rightarrow \delta^S \hookrightarrow \mathcal{C}_3$ | ″ |
| $\mathcal{C} = \mathcal{C}_3\big[\mathsf{case}([\,], \mathsf{inj}_i\, x_i.\mathsf{inj}_i\, \mathcal{C}_i[x_i])\big]$ | ″ |
| $A^S_{11} +^?_i A^S_{21} \le A^S_{12}\, \delta^S\, A^S_{22}$ | Given |
| $A^S_{i1} \le A^S_{i2}$ | By Lemma 1 (Subtyping inversion) |
| $+^?_i \le \delta^S$ | ″ |
| $\mathcal{C}_3 = [\,]$ | By Lemma 64 (Subsums don't need casts) |
| $M$ free | Suppose |
| $x_i$ free | By definition of free |
| $\mathcal{C}_i[x_i]$ free | By Lemma 67 (Static subtypes don't need casts) |
| $\mathsf{inj}_i\, \mathcal{C}_i[x_i]$ free | By definition of free |
| $\mathsf{case}(M, \mathsf{inj}_i\, x_i.\mathsf{inj}_i\, \mathcal{C}_i[x_i])$ free | By definition of free |
| $\mathcal{C}_3\big[\mathsf{case}(M, \mathsf{inj}_i\, x_i.\mathsf{inj}_i\, \mathcal{C}_i[x_i])\big]$ free | By definition of $\mathcal{C}_3$ |

2. Similar to the proof for the previous statement, hence omitted. □

**Lemma 66** (Static sums don't need casts)**.**
*If* $\delta^S_0 \le \delta^S$ *and* $\delta^S_0 \Rightarrow \delta^S \hookrightarrow \mathcal{C}$ *then* $\mathcal{C} = [\,]$.

*Proof.* By definition of sum translation, $|\delta^S_0| = \delta^S_0$ and $|\delta^S| = \delta^S$. Therefore, $|\delta^S_0| \le |\delta^S|$. By inversion on $\delta^S_0 \Rightarrow \delta^S \hookrightarrow \mathcal{C}$, either rule CoeSub or CoeCast was applied. If rule CoeCast was applied then $|\delta^S_0| \not\le |\delta^S|$, a contradiction. If rule CoeSub was applied, then indeed $\mathcal{C} = [\,]$. □

**Lemma 67** (Static subtypes don't need casts)**.**
*If* $A^S_0 \le A^S$ *and* $A^S_0 \Rightarrow A^S \hookrightarrow \mathcal{C}$ *then* $\mathcal{C}[M]$ free *for any* $M$ free.

*Proof.* By induction on the structure of the derivation of $A^S_0 \Rightarrow A^S \hookrightarrow \mathcal{C}$.

- **Case** CoeUnit: Immediate by the definition of $\mathcal{C} = [\,]$.
- **Case**
$$\frac{A^S_{12} \Rightarrow A^S_{11} \hookrightarrow \mathcal{C}_1 \qquad A^S_{21} \Rightarrow A^S_{22} \hookrightarrow \mathcal{C}_2}{(A^S_{11} \to A^S_{21}) \Rightarrow (A^S_{12} \to A^S_{22}) \hookrightarrow \lambda x.\, \mathcal{C}_2\big[[\,]\, \mathcal{C}_1[x]\big]}\ \mathsf{Coe}{\to}$$

| | |
|---|---|
| $A^S_{11} \to A^S_{21} \le A^S_{12} \to A^S_{22}$ | Given |
| $A^S_{12} \le A^S_{11}$ | By Lemma 1 (Subtyping inversion) |
| $A^S_{21} \le A^S_{22}$ | ″ |
| $x$ free | By the definition of free |
| $A^S_{12} \Rightarrow A^S_{11} \hookrightarrow \mathcal{C}_1$ | Subderivation |
| $\mathcal{C}_1[x]$ free | By the induction hypothesis |
| $M$ free | Suppose |
| $M\, \mathcal{C}_1[x]$ free | By the definition of free |
| $A^S_{21} \Rightarrow A^S_{22} \hookrightarrow \mathcal{C}_2$ | Subderivation |
| $\mathcal{C}_2\big[M\, \mathcal{C}_1[x]\big]$ free | By the induction hypothesis |
| $\lambda x.\, \mathcal{C}_2\big[M\, \mathcal{C}_1[x]\big]$ free | By the definition of free |

- **Case**
$$\frac{A^S_{11} \Rightarrow A^S_{12} \hookrightarrow \mathcal{C}_1 \qquad +_1 \Rightarrow \delta^S \hookrightarrow \mathcal{C}_3}{(A^S_{11} +_1 A^S_{21}) \Rightarrow (A^S_{12}\, \delta^S\, A^S_{22}) \hookrightarrow \mathcal{C}_3\big[\mathsf{case}([\,], \mathsf{inj}_1\, x_1.\mathsf{inj}_1\, \mathcal{C}_1[x_1])\big]}\ \mathsf{CoeCase1L}$$

$$
\begin{array}{ll}
A^S_{11} +_1 A^S_{21} \leq A^S_{12}\,\delta^S\,A^S_{22} & \text{Given} \\
A^S_{11} \leq A^S_{12} & \text{By Lemma 1 (Subtyping inversion)} \\
A^S_{21} \leq A^S_{22} & '' \\
+_1 \leq \delta^S & '' \\
\\
+_1 \Rightarrow \delta^S \hookrightarrow \mathcal{C}_3 & \text{Subderivation} \\
\mathcal{C}_3 = [\,] & \text{By Lemma 66 (Static sums don't need casts)} \\
\\
x_1\ \mathsf{free} & \text{By the definition of } \mathsf{free} \\
A^S_{11} \Rightarrow A^S_{12} \hookrightarrow \mathcal{C}_1 & \text{Subderivation} \\
\mathcal{C}_1[x_1]\ \mathsf{free} & \text{By the induction hypothesis} \\
\mathsf{inj}_1\,\mathcal{C}_1[x_1]\ \mathsf{free} & \text{By the definition of } \mathsf{free} \\
\\
M\ \mathsf{free} & \text{Suppose} \\
\mathsf{case}(M, \mathsf{inj}_1\,x_1.\mathcal{C}_1[x_1])\ \mathsf{free} & \text{By the definition of } \mathsf{free} \\
\mathcal{C}_3\big[\mathsf{case}(M, \mathsf{inj}_1\,x_1.\mathcal{C}_1[x_1])\big]\ \mathsf{free} & \text{By the definition of } \mathcal{C}_3
\end{array}
$$

- **Case** $\mathsf{CoeCase1R}$: Symmetric to the $\mathsf{CoeCase1L}$ case, hence omitted.
- **Case**

$$
\dfrac{\begin{array}{ccc} +^?_1 \Rightarrow + \hookrightarrow \mathcal{C}'_1 & +^?_2 \Rightarrow + \hookrightarrow \mathcal{C}'_2 & \\ A^S_{11} \Rightarrow A^S_{12} \hookrightarrow \mathcal{C}_1 & A^S_{21} \Rightarrow A^S_{22} \hookrightarrow \mathcal{C}_2 & + \Rightarrow \delta^S \hookrightarrow \mathcal{C}_3 \end{array}}{(A^S_{11} + A^S_{21}) \Rightarrow (A^S_{12}\,\delta^S\,A^S_{22}) \ \hookrightarrow\ \mathcal{C}_3\big[\mathsf{case}([\,], \mathsf{inj}_1\,x_1.\mathcal{C}'_1[\mathsf{inj}_1\,\mathcal{C}_1[x_1]], \mathsf{inj}_2\,x_2.\mathcal{C}'_2[\mathsf{inj}_2\,\mathcal{C}_2[x_2]])\big]}\ \mathsf{CoeCase2}
$$

$$
\begin{array}{ll}
A^S_{11} + A^S_{21} \leq A^S_{12}\,\delta^S\,A^S_{22} & \text{Given} \\
A^S_{11} \leq A^S_{12} & \text{By Lemma 1 (Subtyping inversion)} \\
A^S_{21} \leq A^S_{22} & '' \\
+ \leq \delta^S & '' \\
\\
+^?_1 \Rightarrow + \hookrightarrow \mathcal{C}'_1 & \text{Subderivation} \\
+^?_2 \Rightarrow + \hookrightarrow \mathcal{C}'_2 & \text{Subderivation} \\
+ \Rightarrow \delta^S \hookrightarrow \mathcal{C}_3 & \text{Subderivation} \\
\mathcal{C}'_1 = [\,] & \text{By inversion on } \mathsf{CoeSub} \\
\mathcal{C}'_2 = [\,] & \text{By inversion on } \mathsf{CoeSub} \\
\mathcal{C}_3 = [\,] & \text{By Lemma 66 (Static sums don't need casts)} \\
\\
x_1\ \mathsf{free} & \text{By the definition of } \mathsf{free} \\
A^S_{11} \Rightarrow A^S_{12} \hookrightarrow \mathcal{C}_1 & \text{Subderivation} \\
\mathcal{C}_1[x_1]\ \mathsf{free} & \text{By the induction hypothesis} \\
\mathsf{inj}_1\,\mathcal{C}_1[x_1]\ \mathsf{free} & \text{By the definition of } \mathsf{free} \\
\mathcal{C}'_1[\mathsf{inj}_1\,\mathcal{C}_1[x_1]]\ \mathsf{free} & \text{By the definition of } \mathcal{C}'_1 \\
\\
x_2\ \mathsf{free} & \text{By the definition of } \mathsf{free} \\
A^S_{21} \Rightarrow A^S_{22} \hookrightarrow \mathcal{C}_2 & \text{Subderivation} \\
\mathcal{C}_2[x_2]\ \mathsf{free} & \text{By the induction hypothesis} \\
\mathsf{inj}_2\,\mathcal{C}_2[x_2]\ \mathsf{free} & \text{By the definition of } \mathsf{free} \\
\mathcal{C}'_2[\mathsf{inj}_2\,\mathcal{C}_2[x_2]]\ \mathsf{free} & \text{By the definition of } \mathcal{C}'_2
\end{array}
$$

$$
\begin{array}{ll}
M\ \mathsf{free} & \text{Suppose} \\
\mathsf{case}(M, \mathsf{inj}_1\,x_1.\mathcal{C}'_1[\mathsf{inj}_1\,\mathcal{C}_1[x_1]], \mathsf{inj}_2\,x_2.\mathcal{C}'_2[\mathsf{inj}_2\,\mathcal{C}_2[x_2]])\ \mathsf{free} & \text{By the definition of } \mathsf{free} \\
\mathcal{C}_3\big[\mathsf{case}(M, \mathsf{inj}_1\,x_1.\mathcal{C}'_1[\mathsf{inj}_1\,\mathcal{C}_1[x_1]], \mathsf{inj}_2\,x_2.\mathcal{C}'_2[\mathsf{inj}_2\,\mathcal{C}_2[x_2]])\big]\ \mathsf{free} & \text{By definition of } \mathcal{C}_3
\end{array}
$$

□

**Theorem 10** (Static derivations don't have match failures)**.**
*If* $\Gamma^S \vdash e^S \Leftarrow A^S$ *or* $\Gamma^S \vdash e^S \Rightarrow A^S$
*then there exists* $M$ *such that* $\Gamma^S \vdash e^S : A^S \hookrightarrow M$
*and* $M$ *is free of casts and* $\mathtt{matchfail}$*.*

*Proof.* By induction on the structure of the given derivation.

- **Case** $\mathsf{SynVar}$: Apply rule $\mathsf{STVar}$. $M = x$ is free of casts and $\mathtt{matchfail}$.
- **Case** $\dfrac{\Gamma^S \vdash e^S \Rightarrow A^S_0 \qquad A^S_0 \rightsquigarrow A^S}{\Gamma^S \vdash e^S \Leftarrow A^S}\ \mathsf{ChkCSub}$

| | | |
|---|---|---|
| | $\Gamma^S \vdash e^S \Rightarrow A_0^S$ | Subderivation |
| | $\Gamma^S \vdash e^S : A_0^S \hookrightarrow M'$ | By the induction hypothesis |
| | $M'$ free | $''$ |
| | $A_0^S \rightsquigarrow A^S$ | Subderivation |
| | $A_0^S \leq A^S$ | By Lemma 38 (Directed consistency for static types) |
| | $A_0^S \simeq A^S$ | By Lemma 15 (Subtyping obeys Structural Equivalence) |
| | $A_0^S \Rightarrow A^S \hookrightarrow \mathcal{C}$ | By Theorem 17 |
| ☞ | $\Gamma^S \vdash e^S : A^S \hookrightarrow \mathcal{C}[M']$ | By rule STCSub |
| ☞ | $\mathcal{C}[M']$ free | By Lemma 67 (Static subtypes don't need casts) |

- **Case** SynAnno: Use the induction hypothesis, the definition of free, and apply rule STAnno.
- **Case** ChkUnitIntro: Apply rule STUnitIntro. $M = ()$ is free of casts and matchfail.
- **Case**

$$\frac{\Gamma^S \vdash e_i^S \Leftarrow A_i^S \qquad +_i^? \leq \delta^S}{\Gamma^S \vdash \mathsf{inj}_i\, e_i^S \Leftarrow (A_1^S\, \delta^S\, A_2^S)}\ \mathsf{ChkSumIntro}$$

| | | |
|---|---|---|
| | $\Gamma^S \vdash e_i^S \Leftarrow A_i^S$ | Subderivation |
| | $\Gamma^S \vdash e_i^S : A_i^S \hookrightarrow M_i$ | By the induction hypothesis |
| | $M_i$ free | $''$ |
| | $A_1^S \leq A_1^S$ | By Lemma 2 (Reflexivity of subtyping) |
| | $A_2^S \leq A_2^S$ | By Lemma 2 (Reflexivity of subtyping) |
| | $+_i^? \leq \delta^S$ | Subderivation |
| | $A_1^S +_i^? A_2^S \leq A_1^S\, \delta^S\, A_2^S$ | By definition of $\leq$ |
| | $A_1^S +_i^? A_2^S \rightsquigarrow A_1^S\, \delta^S\, A_2^S$ | By Lemma 8 (Subtyping obeys directed consistency) |
| | $A_1^S +_i^? A_2^S \simeq A_1^S\, \delta^S\, A_2^S$ | By Lemma 17 (Directed consistency obeys Structural Equivalence) |
| | $A_1^S +_i^? A_2^S \Rightarrow A_1^S\, \delta^S\, A_2^S \hookrightarrow \mathcal{C}$ | By Theorem 17 |
| | $\Gamma^S \vdash \mathsf{inj}_i\, e_i^S : (A_1^S +_i^? A_2^S) \hookrightarrow \mathsf{inj}_i\, M_i$ | By rule STSumIntro |
| ☞ | $\Gamma^S \vdash \mathsf{inj}_i\, e_i^S : (A_1^S\, \delta^S\, A_2^S) \hookrightarrow \mathcal{C}[\mathsf{inj}_i\, M_i]$ | By rule STCSub |
| | $M_i$ free | By definition of free |
| ☞ | $\mathcal{C}[\mathsf{inj}_i\, M_i]$ free | By Lemma 65 (Gradual sums in static don't need casts) |

- **Case**

$$\frac{\begin{array}{l}\Gamma^S \vdash e_0^S \Rightarrow (A_1^S\, \delta^S\, A_2^S) \\ \delta^S \Rrightarrow +_i^* \qquad\qquad \Gamma^S, x : A_i^S \vdash e_i^S \Leftarrow A^S\end{array}}{\Gamma^S \vdash \mathsf{case}(e_0^S, \mathsf{inj}_i\, x.e_i^S) \Leftarrow A^S}\ \mathsf{ChkSumElim1}$$

| | | |
|---|---|---|
| | $\delta^S \Rrightarrow +_i^*$ | Subderivation |
| | $\delta^S = +_i$ | By Lemma 33 (Static looseness, II) |
| | $A_1^S \leq A_1^S$ | By Lemma 2 (Reflexivity of subtyping) |
| | $A_2^S \leq A_2^S$ | By Lemma 2 (Reflexivity of subtyping) |
| | $\delta^S \leq +_i^*$ | By definition of $\leq$ |
| | $A_1^S\, \delta^S\, A_2^S \leq A_1^S +_i^* A_2^S$ | By definition of $\leq$ |
| | $A_1^S\, \delta^S\, A_2^S \rightsquigarrow A_1^S +_i^* A_2^S$ | By Lemma 8 (Subtyping obeys directed consistency) |
| | $A_1^S\, \delta^S\, A_2^S \simeq A_1^S +_i^* A_2^S$ | By Lemma 17 (Directed consistency obeys Structural Equivalence) |
| | $A_1^S\, \delta^S\, A_2^S \Rightarrow A_1^S +_i^* A_2^S \hookrightarrow \mathcal{C}$ | By Theorem 17 |
| | $\Gamma^S \vdash e_0^S \Rightarrow (A_1^S\, \delta^S\, A_2^S)$ | Subderivation |
| | $\Gamma^S \vdash e_0^S : (A_1^S\, \delta^S\, A_2^S) \hookrightarrow M_0$ | By the induction hypothesis |
| | $M_0$ free | $''$ |
| | $\Gamma^S \vdash e_0^S : (A_1^S +_i^* A_2^S) \hookrightarrow \mathcal{C}[M_0]$ | By rule STCSub |
| | $\mathcal{C}[M_0]$ free | By Lemma 65 (Gradual sums in static don't need casts) |
| | $\Gamma^S, x : A_i^S \vdash e_i^S \Leftarrow A^S$ | Subderivation |
| | $\Gamma^S, x : A_i^S \vdash e_i^S : A^S \hookrightarrow M_i$ | By the induction hypothesis |
| | $M_i$ free | $''$ |
| ☞ | $\Gamma^S \vdash \mathsf{case}(e_0^S, \mathsf{inj}_i\, x.e_i^S) : A^S \hookrightarrow \mathsf{case}(\mathcal{C}[M_0], \mathsf{inj}_i\, x.M_i)$ | By rule STSumElim1 |
| ☞ | $\mathsf{case}(\mathcal{C}[M_0], \mathsf{inj}_i\, x.M_i)$ free | By definition of free |

- **Case** $\dfrac{\Gamma^{S} \vdash e_0^{S} \Rightarrow (A_1^{S}\,\delta^{S}\,A_2^{S}) \qquad \Gamma^{S}, x_1 : A_1^{S} \vdash e_1^{S} \Leftarrow A^{S} \\ \delta^{S} \Rrightarrow + \qquad \Gamma^{S}, x_2 : A_2^{S} \vdash e_2^{S} \Leftarrow A^{S}}{\Gamma^{S} \vdash \mathsf{case}(e_0^{S}, \mathrm{inj}_1\, x_1.e_1^{S}, \mathrm{inj}_2\, x_2.e_2^{S}) \Leftarrow A^{S}}$ ChkSumElim2

| | |
|---|---|
| $A_1^{S} \leq A_1^{S}$ | By Lemma 2 (Reflexivity of subtyping) |
| $A_2^{S} \leq A_2^{S}$ | By Lemma 2 (Reflexivity of subtyping) |
| $\delta^{S} \leq +$ | By Lemma 23 (All sums below +) |
| $A_1^{S}\,\delta^{S}\,A_2^{S} \leq A_1^{S} + A_2^{S}$ | By definition of $\leq$ |
| $A_1^{S}\,\delta^{S}\,A_2^{S} \rightsquigarrow A_1^{S} + A_2^{S}$ | By Lemma 8 (Subtyping obeys directed consistency) |
| $A_1^{S}\,\delta^{S}\,A_2^{S} \simeq A_1^{S} + A_2^{S}$ | By Lemma 17 (Directed consistency obeys Structural Equivalence) |
| $A_1^{S}\,\delta^{S}\,A_2^{S} \Rightarrow A_1^{S} + A_2^{S} \hookrightarrow \mathcal{C}$ | By Theorem 17 |

| | |
|---|---|
| $\Gamma^{S} \vdash e_0^{S} \Rightarrow (A_1^{S}\,\delta^{S}\,A_2^{S})$ | Subderivation |
| $\Gamma^{S} \vdash e_0^{S} : (A_1^{S}\,\delta^{S}\,A_2^{S}) \hookrightarrow M_0$ | By the induction hypothesis |
| $M_0$ free | $''$ |
| $\Gamma^{S} \vdash e_0^{S} : (A_1^{S} + A_2^{S}) \hookrightarrow \mathcal{C}[M_0]$ | By rule STCSub |
| $\mathcal{C}[M_0]$ free | By Lemma 67 (Static subtypes don't need casts) |

| | |
|---|---|
| $\Gamma^{S}, x_1 : A_1^{S} \vdash e_1^{S} \Leftarrow A^{S}$ | Subderivation |
| $\Gamma^{S}, x_1 : A_1^{S} \vdash e_1^{S} : A^{S} \hookrightarrow M_1$ | By the induction hypothesis |
| $M_1$ free | $''$ |

| | |
|---|---|
| $\Gamma^{S}, x_2 : A_2^{S} \vdash e_2^{S} \Leftarrow A^{S}$ | Subderivation |
| $\Gamma^{S}, x_2 : A_2^{S} \vdash e_2^{S} : A^{S} \hookrightarrow M_2$ | By the induction hypothesis |
| $M_2$ free | $''$ |

☞ $\Gamma^{S} \vdash \mathsf{case}(e_0^{S}, \mathrm{inj}_1\, x_1.e_1^{S}, \mathrm{inj}_2\, x_2.e_2^{S}) : A^{S} \hookrightarrow \mathsf{case}(\mathcal{C}[M_0], \mathrm{inj}_1\, x_1.M_1, \mathrm{inj}_2\, x_2.M_2)$ By rule STSumElim2

☞ $\mathsf{case}(\mathcal{C}[M_0], \mathrm{inj}_1\, x_1.M_1, \mathrm{inj}_2\, x_2.M_2)$ free By definition of free

- **Case** Chk$\rightarrow$Intro: Use the induction hypothesis, the definition of free, and apply rule ST$\rightarrow$Intro.
- **Case** Syn$\rightarrow$Elim: Use the induction hypothesis, the definition of free, and apply rule ST$\rightarrow$Elim. □