Gradual Intersection Types

Pedro Ângelo
Faculdade de Ciências & LIACC
Universidade do Porto
pedro.angelo@fc.up.pt

Mário Florido
Faculdade de Ciências & LIACC
Universidade do Porto
amf@dcc.fc.up.pt

Gradual typing integrates dynamic and static types. Since its introduction, it has been successfully applied to several extensions of simple types, including subtyping, parametric polymorphism and substructural types. This work studies its application to intersection type systems. We introduce a new approach to define a gradual version of the original intersection type system of Coppo and Dezani. We then present a new operational semantics for terms typed with static and dynamic intersection types, which enables dynamic type casts and identifies the causes for type errors in a framework inspired by the blame calculus. Finally we prove several properties of our system including a correctness criteria and soundness of the extension of the original gradual type system.

1 Introduction

The recent contributions of gradual typing [3, 4], integrating static and dynamic typing in a single program, are the basis for several type systems. Gradual typing allows to fine tune the distribution of static and dynamic checking in a program, thus harnessing the strengths of both typing disciplines. Regarding polymorphism, gradual typing has been sucessfully applied [8] to the well-known and widely used Hindley-Milner (HM) type system [10, 14], resulting in a gradual type system with polymorphism.

Intersection types [5, 6, 12, 1] were proposed as an alternative to the HM type system. They allow a different form of polymorphism, discrete polymorphism, in which all the (finite) instances of a type are explicitly expressed. Thus, type systems based on these types are able to type more programs than the HM type system, some are able to type all the strongly normalizing terms, and also allow for increased expressiveness when describing instances of polymorphic types.

In our work, we extend gradual typing with intersection types, resulting in a polymorphic system that contains all the expressive power of intersection types along with all the advantages of gradual typing. Our system allows the use of the dynamic type in instances of intersection types, and thus allows a single expression to be typed with dynamic and static types simultaneously. For example, in $(\lambda x : Int \cap Dyn\ .\ x^{Int} + x^{Dyn})\ 1$ different instances of the variable x are typed with different (static and dynamic) types.

In general, our system extends the Gradually Typed Lambda Calculus (GTLC) [3, 4], since besides a type system and the cast calculus, we also present a cast insertion procedure that inserts casts to test types at run-time, and an operational semantics to evaluate these casts. These casts are similar to those of [3], but modified to accomodate intersection types at run-time. This system adheres to the formal correctness criteria, put forth in [16, 3, 4], that guides the design of gradual languages. We also show how our system is a generalization of the GTLC, behaving as the GTLC when expressions are typed with only simple types.

A recent contribution that discussed the use of intersection types in a gradual setting is [2]. This work focused on extending semantic subtyping [7] with gradual types, where types were interpreted as sets of values. In [2], intersection types were interpreted as its corresponding set-theoretic intersection

Submitted to:
ITRS 2018

operator and typed overloaded functions which run a different code for each different type. Our work is fundamentally different from [2] in the sense that we follow the original motivation of intersection type systems, where functions with intersection types may be applied to arguments of different types but they always execute the same code for all of these types. Thus, in our work, intersection types are finitely parametric polymorphic types in opposition with the work reported in [2], where intersection types were used to overload function names which discriminated on the different types of their arguments.

This paper makes the following contributions:

- A gradual type system with intersection types, as well as a cast calculus and cast insertion, thus combining discrete polymorphism with gradual typing (Section 2).
- An operational semantics for the cast calculus, handling the reduction of run-time checks and with support for discrete polymorphism granted by the use of intersection types (Section 3).
- We show important properties, namely the correctness criteria for gradual typing, and we also show how our system is a generalization of the GTLC (Section 4).

2 Gradual Intersection Types

Intersection types add to simple types [9] *intersections of types* of the form $T_1 \cap \ldots \cap T_n$. Intersections of types are independent of the order and the number of occurrences of each type. We thus consider intersections of types modulo the following equivalence relations:

$$\begin{array}{rcll} T_1 \cap \ldots \cap T_i \cap T_{i+1} \cap \ldots \cap T_n & = & T_1 \cap \ldots \cap T_{i+1} \cap T_i \cap \ldots \cap T_n & \text{commutative} \\ T_1 \cap \ldots \cap T_i \cap T_i \cap \ldots \cap T_n & = & T_1 \cap \ldots \cap T_i \cap \ldots \cap T_n & \text{idempotent} \\ (T_1 \to T) \cap \ldots \cap (T_n \to T) & = & T_1 \cap \ldots \cap T_n \to T & \text{distributive} \end{array}$$

We follow the original definition in [5], so intersections are not allowed in the codomain of the arrow type, so $T_1 \to (T_2 \cap T_3)$ is not a valid type, but $(T_1 \cap T_2) \to (T_3 \cap T_4) \to T_5$ is. We do not distinguish between a singleton intersection of types and its sole element, and we assume that if $I_1 = T_1 \cap \ldots \cap T_n$ and $I_2 = T'_1 \cap \ldots \cap T'_m$ then we write $I_1 \cap I_2$ for $T_1 \cap \ldots \cap T_n \cap T'_1 \cap \ldots \cap T'_m$. We say that $T_1 \cap \ldots \cap T_n \subseteq T'_1 \cap \ldots \cap T'_m$ if and only if $\{T_1, \ldots, T_n\} \subseteq \{T'_1, \ldots, T'_m\}$. The intersection $\cap$ has a higher precedence than the arrow type, so $T_1 \cap T_2 \cap T_3 \to T_4$ is equal to $(T_1 \cap T_2 \cap T_3) \to T_4$.

2.1 Syntax and Type System

Here we define our Gradual Intersection Type System ($\vdash_{\cap G}$), or GITS for short. Our language is an explicitly typed λ-calculus with integers and booleans. The syntax and the type system are presented in Figure 1. Gradual typing uses an explicitly typed system with domain-type declarations in λ-abstractions to be able to declare them as dynamic. The same happens with our system, where the programmer controls the type checker behavior with explicit type declarations for function arguments. Rule T-Abs' restricts to binding just one instance of the intersection to the variable in the context of the (single) type judgement in the premise of the rule. This allows the programmer to explicitly control the generation of alternative types for terms (this extra rule goes back to Reynolds in [15]). We say that two types are consistent if the instances of both types are consistent. Considering the properties of intersection types, they are consistent with a simple type (*Int* for example), if all the instances are consistent with that simple type. Looking at the definition of pattern matching, we see that if the type of the function, in the application rule, is a function type, then pattern matching gets its domain and codomain. However,

Syntax

$$\begin{array}{l} \textit{Types } T ::= \ Int \mid Bool \mid Dyn \mid T \rightarrow T \mid T \cap \ldots \cap T \\ \textit{Expressions } e \ ::= x^T \mid \lambda x : T \ . \ e \mid e \ e \mid n \mid true \mid false \end{array}$$

$\boxed{\Gamma \vdash_{\cap G} e : T}$ Typing

$$\frac{\Gamma, x : T_1 \cap \ldots \cap T_n \vdash_{\cap G} \ e : T}{\Gamma \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n \ . \ e : T_1 \cap \ldots \cap T_n \rightarrow T} \text{ T-ABS} \qquad \frac{\begin{array}{cc} \Gamma \vdash_{\cap G} e_1 : PM & PM \rhd T_1 \cap \ldots \cap T_n \rightarrow T \\ \Gamma \vdash_{\cap G} e_2 : T_1' \cap \ldots \cap T_n' & T_1' \cap \ldots \cap T_n' \sim T_1 \cap \ldots \cap T_n \end{array}}{\Gamma \vdash_{\cap G} e_1 \ e_2 : T} \text{ T-APP}$$

$$\frac{x : T' \in \Gamma \qquad T \subseteq T'}{\Gamma \vdash_{\cap G} x^T : T} \text{ T-VAR} \qquad \frac{\Gamma, x : T_i \vdash_{\cap G} \ e : T}{\Gamma \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n \ . \ e : T_i \rightarrow T} \text{ T-ABS'} \qquad \frac{\Gamma \vdash_{\cap G} e : T_1 \ \cdots \ \Gamma \vdash_{\cap G} e : T_n}{\Gamma \vdash_{\cap G} e : T_1 \cap \ldots \cap T_n} \text{ T-GEN}$$

$$\frac{\Gamma \vdash_{\cap G} e : T_1 \cap \ldots \cap T_n}{\Gamma \vdash_{\cap G} e : T_i} \text{ T-INST} \qquad \frac{}{\Gamma \vdash_{\cap G} n : Int} \text{ T-INT} \qquad \frac{}{\Gamma \vdash_{\cap G} true : Bool} \text{ T-TRUE} \qquad \frac{}{\Gamma \vdash_{\cap G} false : Bool} \text{ T-FALSE}$$

$\boxed{T \sim T}$ Consistency

$$T \sim T \qquad T \sim Dyn \qquad Dyn \sim T \qquad \frac{T_1 \sim T_3 \qquad T_2 \sim T_4}{T_1 \rightarrow T_2 \sim T_3 \rightarrow T_4} \qquad \frac{T_1 \sim T_1' \ \cdots \ T_n \sim T_n'}{T_1 \cap \ldots \cap T_n \sim T_1' \cap \ldots \cap T_n'}$$

$\boxed{T \rhd T}$ Pattern Matching

$$T_1 \rightarrow T_2 \rhd T_1 \rightarrow T_2 \qquad\qquad Dyn \rhd Dyn \rightarrow Dyn$$

Figure 1: Gradual Intersection Type System ($\vdash_{\cap G}$)

if the function is typed dynamically, then it must be treated as a function type from dynamic to dynamic. Finally, if the type of the function is not a functional type and not dynamic then the typing rule for application is not applicable. Being an explicitly typed calculus we assume that expressions are always typed with an intersection type. For example, rule T-App assumes that e_2 is typed with $T_1' \cap \ldots \cap T_n'$. However, these rules also accept simple types, when $n = 1$.

In our correctness criteria we will relate our system with a standard statically typed λ-calculus with intersection types, here called the Static Intersection Type System ($\vdash_{\cap S}$), or SITS for short (presented in Appendix A for space limitation reasons). The Gradualizer [3] is a methodology to automatically generate a gradual type system from a given static type system. It accomplishes that purpose by individually inspecting the typing rules of a type system and transforming those rules into their gradual counterpart. It also generates the cast calculus typing rules and a cast insertion procedure from the static type system. We note that applying the gradualizer to the SITS results in our Gradual Intersection Type System, however our cast calculus and cast insertion procedure were designed by intuition.

2.2 Cast Calculus

In our system, casts only allow simple types. We are forced to consider how the intersection type is treated in the scope of the type system, and to rethink the definition of casts, as in [3, 4]. We solve this difficulty with the following reasoning: each instance of the intersection type generates a different cast during cast insertion. Therefore, all the instances produce various casts, which during evaluation are reduced independently. This is the key insight used in designing this system, and to implement it,

Syntax

$$\begin{aligned} &\mathit{Types}\ T ::= \mathit{Int} \mid \mathit{Bool} \mid \mathit{Dyn} \mid T \to T \\ &\mathit{Casts}\ c\ ::= c : T \Rightarrow^{l} T\ ^{cl} \mid \mathit{blame}\ T\ T\ l\ ^{cl} \mid \varnothing\ T\ ^{cl} \end{aligned}$$

$\boxed{\vdash_{\cap CI} c : (T,T)}$ Typing

$$\frac{\vdash_{\cap CI} c : (T,T_1) \qquad T_1 \sim T_2}{\vdash_{\cap CI} c : T_1 \Rightarrow^{l} T_2\ ^{cl} : (T,T_2)}\ \textsc{T-SingleCI} \qquad \frac{}{\vdash_{\cap CI} \mathit{blame}\ T_I\ T_F\ l\ ^{cl} : (T_I,T_F)}\ \textsc{T-BlameCI} \qquad \frac{}{\vdash_{\cap CI} \varnothing\ T\ ^{cl} : (T,T)}\ \textsc{T-EmptyCI}$$

Figure 2: Cast Intersection Type System ($\vdash_{\cap CI}$)

the cast as in the original contribution from [3, 4] is replaced with the cast intersection, $e : c_1 \cap \ldots \cap c_n$, where c_i represents a series of casts that are evaluated independently, and each of these c_i are related to an instance of the intersection type. The need for a system that handles the various casts in cast intersections gave rise to the Cast Intersection system, with each cast c_i being part of the Cast Intersection system. Therefore, our cast calculus is composed of both the Intersection Cast Calculus ($\vdash_{\cap CC}$), which is the counterpart to the Cast Calculus from [3], and the Cast Intersection Type System ($\vdash_{\cap CI}$), which types these casts c_i.

Cast Intersection Type System The Cast Intersection Type System ($\vdash_{\cap CI}$), or CITS for short, in Figure 2, is the type system for the sublanguage of casts. The sublanguage is composed of 3 terms: the single cast $c : T_1 \Rightarrow^{l} T_2\ ^{cl}$ casts a type T_1 to a type T_2; the blame cast $\mathit{blame}\ T_I\ T_F\ l\ ^{cl}$ represents a blame with initial type T_I, final type T_F and blame label l [3]; and the empty cast $\varnothing\ T\ ^{cl}$ is an identity cast (from T to T). No context is required in the typing rules of Figure 2 because there are no variables. The cast label cl is a mark used to compare casts depending on their origin, its purpose will be explained in Section 3.

A cast c can have many casts (single, blame or empty casts), and therefore, many source and target types, as defined in [4]. We adapt the concept of source and target type, and rename it as initial and final type, with initial type refering to the source type of the inner (left) most cast and final type refering to the target type of the outer (right) most cast. In the typing judgement $\vdash_{\cap CI} c : (T_I, T_F)$, c has initial type T_I and final type T_F.

Intersection Cast Calculus The Intersection Cast Calculus ($\vdash_{\cap CC}$), shown in Figure 3, is the counterpart in our system to the Cast Calculus from [3]. It introduces the cast intersection $e : c_1 \cap \ldots \cap c_n$ and its typing rules. In the GTLC, a cast is well typed if the source type of the cast is equal to the type of the sub expression. In our system, as per rule T-CastIntersection, a cast intersection is well typed if the intersection of the initial types of all casts c_i is equal to the type of the subexpression. The cast intersection then types with the intersection of the final types of all casts c_i. The Intersection Cast Calculus also introduzes the blame term, $\mathit{blame}_T\ l$, as described in the original contribution [4].

Rule T-App is not sufficient to type applications, since casts are typed with intersections only appearing in the top level of the type. For example, if $\Gamma \vdash_{\cap CC} e_1 : (\mathit{Int} \to \mathit{Int}) \cap (\mathit{Bool} \to \mathit{Int})$ and $\Gamma \vdash_{\cap CC} e_2 : \mathit{Int} \cap \mathit{Bool}$, then by rule T-App, $e_1\ e_2$ is not typeable. Rule T-App' is then required to type applications. During cast insertion, types are converted such that intersection types only appear in the top level of the type. Then, rule T-App' and T-CastIntersection only expect types with this restriction, and don't allow intersection types in their type variables.

Syntax

$$\begin{array}{l} \textit{Types } T ::= \textit{Int} \mid \textit{Bool} \mid \textit{Dyn} \mid T \to T \mid T \cap \ldots \cap T \\ \textit{Expressions } e ::= x^T \mid \lambda x : T \,.\, e \mid e\; e \mid n \mid \textit{true} \mid \textit{false} \mid e : c \cap \ldots \cap c \mid \textit{blame}_T\; l \end{array}$$

$\boxed{\Gamma \vdash_{\cap CC} e : T}$ Typing

Static Intersection Type System $(\vdash_{\cap S})$ *rules and*

$$\frac{\Gamma \vdash_{\cap CC} e_1 : (T_{11} \to T_{12}) \cap \ldots \cap (T_{n1} \to T_{n2}) \qquad \Gamma \vdash_{\cap CC} e_2 : T_{11} \cap \ldots \cap T_{n1}}{\Gamma \vdash_{\cap CC} e_1\; e_2 : T_{12} \cap \ldots \cap T_{n2}}\ \text{T-APP'}$$

$$\frac{\Gamma \vdash_{\cap CC} e : T_1'' \cap \ldots \cap T_n'' \qquad \vdash_{\cap CI} c_1 : (T_1', T_1) \cdots \vdash_{\cap CI} c_n : (T_n', T_n) \qquad T_1'' \cap \ldots \cap T_n'' = T_1' \cap \ldots \cap T_n'}{\Gamma \vdash_{\cap CC} e : c_1 \cap \ldots \cap c_n : T_1 \cap \ldots \cap T_n}\ \text{T-CASTINTERSECTION} \qquad \frac{}{\Gamma \vdash_{\cap CC} \textit{blame}_T\; l : T}\ \text{T-BLAME}$$

Figure 3: Intersection Cast Calculus $(\vdash_{\cap CC})$

2.3 Cast Insertion

The compilation of the Gradual Intersection Type System to the Intersection Cast Calculus (also called cast insertion) is displayed in Figure 4. This compilation to the Cast Calculus is written $\Gamma \vdash_{\cap CC} e \rightsquigarrow e' : T$, meaning that e is compiled to e' with type T in the type environment Γ, and it basically inserts run-time casts in subexpressions where the type system uses consistency to compare types. The cast insertion rules for the Intersection Cast Calculus are similar to the cast insertion rules for the cast calculus of [3], with the exception that the rules are adapted to deal with the Gradual Intersection Type System, which results in 3 aditional rules: C-Abs', C-Gen and C-Inst. Both compilation systems only insert casts in the application of terms, therefore rules that deal with other terms are similar. Regarding the rule for application, the two systems differ mainly due to different casts. Inserting cast intersections follows the same basic principle as cast insertion in [3]: we add a cast intersection to the expression on the left with initial types equal to the instances of the type of the expression and final types equal to the instances of the result of pattern matching, and we add a cast intersection to the expression on the right with initial types equal to the instances of the type of the expression and final types equal to the instances of the type given by the consistency relation. As we are dealing with intersection types, we must first retrieve the simple types that make up all the instances of the intersection type with the instances $(\trianglelefteq)$ relation presented in Figure 4. Then we add the casts with the cast insertion $(S,\ S,\ e \hookrightarrow e)$ relation.

Theorem (Instances of Intersection Types)**.** *We define the set S of* instances *of an intersection type T as the set obtained by $T \trianglelefteq S$. Given a type T, if $T \trianglelefteq S$ then every element of S is a simple type.*

The proof for this theorem can be found in Appendix B. This allows the system to deal with finite rank intersection types [11, 12, 13], by inserting as casts all its alternatives.

3 Normalization

Due to the existence of two systems, the Cast Intersection Type System $(\vdash_{\cap CI})$ and the Intersection Cast Calculus $(\vdash_{\cap CC})$, the reduction rules of our system are divided into two operational semantics: the Cast Intersection operational semantics $(\longrightarrow_{\cap CI})$ and the Intersection Cast Calculus operational semantics $(\longrightarrow_{\cap CC})$.

$\boxed{\Gamma \vdash_{\cap CC} e \rightsquigarrow e : T}$ Compilation

$$\frac{\Gamma, x : T_1 \cap \ldots \cap T_n \vdash_{\cap CC} e \rightsquigarrow e' : T}{\Gamma \vdash_{\cap CC} \lambda x : T_1 \cap \ldots \cap T_n \,.\, e \rightsquigarrow \lambda x : T_1 \cap \ldots \cap T_n \,.\, e' : T_1 \cap \ldots \cap T_n \to T}\ \textsc{C-Abs}$$

$$\frac{\begin{array}{c} \Gamma \vdash_{\cap CC} e_1 \rightsquigarrow e_1' : PM \\ PM \rhd T_1 \cap \ldots \cap T_n \to T \qquad \Gamma \vdash_{\cap CC} e_2 \rightsquigarrow e_2' : T_1' \cap \ldots \cap T_n' \qquad T_1' \cap \ldots \cap T_n' \sim T_1 \cap \ldots \cap T_n \qquad PM \trianglelefteq S_1 \\ T_1 \cap \ldots \cap T_n \to T \trianglelefteq S_2 \qquad T_1' \cap \ldots \cap T_n' \trianglelefteq S_3 \qquad T_1 \cap \ldots \cap T_n \trianglelefteq S_4 \qquad S_1,\ S_2,\ e_1' \hookrightarrow e_1'' \qquad S_3,\ S_4,\ e_2' \hookrightarrow e_2'' \end{array}}{\Gamma \vdash_{\cap CC} e_1\ e_2 \rightsquigarrow e_1''\ e_2'' : T}\ \textsc{C-App}$$

$$\frac{x : T' \in \Gamma \qquad T \subseteq T'}{\Gamma \vdash_{\cap CC} x^T \rightsquigarrow x^T : T}\ \textsc{C-Var} \qquad \frac{\Gamma, x : T_i \vdash_{\cap CC} e \rightsquigarrow e' : T}{\Gamma \vdash_{\cap CC} \lambda x : T_1 \cap \ldots \cap T_n \,.\, e \rightsquigarrow \lambda x : T_1 \cap \ldots \cap T_n \,.\, e' : T_i \to T}\ \textsc{C-Abs'}$$

$$\frac{\Gamma \vdash_{\cap CC} e \rightsquigarrow e' : T_1 \ \cdots\ \Gamma \vdash_{\cap CC} e \rightsquigarrow e' : T_n}{\Gamma \vdash_{\cap CC} e \rightsquigarrow e' : T_1 \cap \ldots \cap T_n}\ \textsc{C-Gen} \qquad \frac{\Gamma \vdash_{\cap CC} e \rightsquigarrow e' : T_1 \cap \ldots \cap T_n}{\Gamma \vdash_{\cap CC} e \rightsquigarrow e' : T_i}\ \textsc{C-Inst} \qquad \frac{}{\Gamma \vdash_{\cap CC} n \rightsquigarrow n : Int}\ \textsc{C-Int}$$

$$\frac{}{\Gamma \vdash_{\cap CC} true \rightsquigarrow true : Bool}\ \textsc{C-True} \qquad \frac{}{\Gamma \vdash_{\cap CC} false \rightsquigarrow false : Bool}\ \textsc{C-False}$$

$\boxed{\mathrm{T} \trianglelefteq \mathrm{S}}$ Instances

$$Int \trianglelefteq \{Int\} \qquad Bool \trianglelefteq \{Bool\} \qquad Dyn \trianglelefteq \{Dyn\}$$

$$\frac{T_1 \trianglelefteq \{T_{11}, \ldots, T_{1n}\}}{T_1 \to T_2 \trianglelefteq \{T_{11} \to T_2, \ldots, T_{1n} \to T_2\}} \qquad \frac{T_1 \trianglelefteq \{T_{11}, \ldots, T_{1m}\} \ \cdots\ T_n \trianglelefteq \{T_{n1}, \ldots, T_{nj}\}}{T_1 \cap \ldots \cap T_n \trianglelefteq \{T_{11}, \ldots, T_{1m}, \ldots, T_{n1}, \ldots, T_{nj}\}}$$

$\boxed{S,\ S,\ e \hookrightarrow e}$ Cast Insertion

$$\{T_{11}, \ldots, T_{1n}\},\ \{T_{21}, \ldots, T_{2n}\},\ e \hookrightarrow e : (\varnothing\, {T_{11}}^0 : T_{11} \Rightarrow^{l_1} {T_{21}}^0) \cap \ldots \cap (\varnothing\, {T_{1n}}^0 : T_{1n} \Rightarrow^{l_n} {T_{2n}}^0)$$

$$\{T_{11}, \ldots, T_{1n}\},\ \{T_2\},\ e \hookrightarrow e : (\varnothing\, {T_{11}}^0 : T_{11} \Rightarrow^{l_1} {T_2}^0) \cap \ldots \cap (\varnothing\, {T_{1n}}^0 : T_{1n} \Rightarrow^{l_n} {T_2}^0)$$

$$\{T_1\},\ \{T_{21}, \ldots, T_{2n}\},\ e \hookrightarrow e : (\varnothing\, {T_1}^0 : T_1 \Rightarrow^{l_1} {T_{21}}^0) \cap \ldots \cap (\varnothing\, {T_1}^0 : T_1 \Rightarrow^{l_n} {T_{2n}}^0)$$

Figure 4: Compilation to the Intersection Cast Calculus

Cast Intersection Reduction Rules The Cast Intersection operational semantics is presented in Figure 5. These rules are solely responsible for the reduction of casts pertaining a single instance of intersection types, and as such, casts only contain simple types. Therefore, these rules are very similar to the cast handler reduction rules in [4]. The original purpose of the rules in [4] was preserved in this system: rule E-PushBlameCI is responsible for triggering blame to the top level; rules E-IdentityCI, E-SucceedCI and E-FailCI detect the success or failure of casts; and rules E-GroundCI and E-ExpandCI mediate the transition between the two disciplines. Note that, as presented in [4], G is a ground type of T if $G \sim T$ and $G \neq T$, and also G is a ground type. Each rule in the Cast Intersection operational semantics has a counterpart in the cast handler reduction rules.

Due to the existence of the empty cast and the blame cast in this system, cast values remain similar to the original definition of values pertaining casts of [4], with a few exceptions. Casts from ground types to the dynamic type and casts from an arrow type to a different arrow type are kept as values, however we now recursively check if the sub-cast is also a value, or if the recursion stops with the empty cast. Regarding the new casts of the language, as neither blame casts nor empty casts can be further reduced, they are also considered values.

Syntax

$$\begin{array}{ll}
\mathit{Types}\ T ::= & \mathit{Int} \mid \mathit{Bool} \mid \mathit{Dyn} \mid T \to T \\
\mathit{Ground\ Types}\ G ::= & \mathit{Int} \mid \mathit{Bool} \mid \mathit{Dyn} \to \mathit{Dyn} \\
\mathit{Casts}\ c ::= & c : T \Rightarrow^{l} T^{\ cl} \mid \mathit{blame}\ T\ T\ l^{\ cl} \mid \varnothing\ T^{\ cl} \\
\mathit{Cast\ Values}\ \ cv ::= & cv1 \mid \mathit{blame}\ T\ T\ l^{\ cl} \\
cv1 ::= & \varnothing\ T^{\ cl} \mid cv1 : G \Rightarrow^{l} \mathit{Dyn}^{\ cl} \mid cv1 : T_1 \to T_2 \Rightarrow^{l} T_3 \to T_4^{\ cl}
\end{array}$$

$\boxed{c \longrightarrow_{\cap CI} c}$ Evaluation

$$\frac{}{\mathit{blame}\ T_I\ T_F\ l_1^{\ cl_1} : T_1 \Rightarrow^{l_2} T_2^{\ cl_2} \longrightarrow_{\cap CI} \mathit{blame}\ T_I\ T_2\ l_1^{\ cl_1}}\ \textsc{E-PushBlameCI}$$

$$\frac{\neg(\mathit{is\ cast\ value}\ c) \qquad c \longrightarrow_{\cap CI} c'}{c : T_1 \Rightarrow^{l} T_2^{\ cl} \longrightarrow_{\cap CI} c' : T_1 \Rightarrow^{l} T_2^{\ cl}}\ \textsc{E-EvaluateCI} \qquad \frac{}{cv1 : T \Rightarrow^{l} T^{\ cl} \longrightarrow_{\cap CI} cv1}\ \textsc{E-IdentityCI}$$

$$\frac{}{cv1 : G \Rightarrow^{l_1} \mathit{Dyn}^{\ cl_1} : \mathit{Dyn} \Rightarrow^{l_2} G^{\ cl_2} \longrightarrow_{\cap CI} cv1}\ \textsc{E-SucceedCI}$$

$$\frac{\neg(\mathit{same\ ground}\ G_1\ G_2) \qquad \vdash_{\cap CI} cv1 : (T_I, T)}{cv1 : G_1 \Rightarrow^{l_1} \mathit{Dyn}^{\ cl_1} : \mathit{Dyn} \Rightarrow^{l_2} G_2^{\ cl_2} \longrightarrow_{\cap CI} \mathit{blame}\ T_I\ G_2\ l_2^{\ cl_1}}\ \textsc{E-FailCI}$$

$$\frac{G\ \mathit{is\ ground\ type\ of}\ T \qquad \neg(\mathit{is\ ground\ type}\ T)}{cv1 : T \Rightarrow^{l} \mathit{Dyn}^{\ cl} \longrightarrow_{\cap CI} cv1 : T \Rightarrow^{l} G^{\ cl} : G \Rightarrow^{l} \mathit{Dyn}^{\ cl}}\ \textsc{E-GroundCI}$$

$$\frac{G\ \mathit{is\ ground\ type\ of}\ T \qquad \neg(\mathit{is\ ground\ type}\ T)}{cv1 : \mathit{Dyn} \Rightarrow^{l} T^{\ cl} \longrightarrow_{\cap CI} cv1 : \mathit{Dyn} \Rightarrow^{l} G^{\ cl} : G \Rightarrow^{l} T^{\ cl}}\ \textsc{E-ExpandCI}$$

Figure 5: Cast Intersection Operational Semantics ($\longrightarrow_{\cap CI}$)

Intersection Cast Calculus Reduction Rules The Intersection Cast Calculus operational semantics is presented in Figure 6. Being the counterpart to the Cast Calculus operational semantics from [4], these rules specify evaluation of the λ-calculus terms and the remaining terms of the language, including cast intersections. Updating the definition of casts from [3] to cast intersections ($e : c_1 \cap \ldots \cap c_n$) requires some reduction rules to be adapted as well.

In [4], cast handler reduction rules were a part of the operational semantics. However, with the new definition of casts, we require a rule to connect the Intersection Cast Calculus with the evaluation rules for casts, given by the Cast Intersection operational semantics. E-EvaluateCasts then establishes a bridge between the two operational semantics, by specifying that casts c_i in the cast intersection $e : c_1 \cap \ldots \cap c_n$ are reduced in parallel.

The simulation rule for the arrow type, C-BETA in [4], also requires adaptation, resulting in the rule E-SimulateArrow (we have not included the definition of the function simulateArrow due to space constraints, but it can be found in Appendix A). The key insight of this rule remains the same, only extended to deal with multiple casts. Casts that are not compatible with the arrow type are filtered out, so the only casts that are used in this step are empty casts with an arrow type ($\varnothing\ T_1 \to T_2^{\ cl}$) or casts from arrow types to arrow types, $c : T_1 \to T_2 \Rightarrow^{l} T_3 \to T_4^{\ cl}$, assuming c also follows these restrictions. Then we take the outermost cast and divide it, forming two casts and distributing these between the terms. For example $(v : (\varnothing\ T_1 \to T_2^{\ cl} : T_1 \to T_2 \Rightarrow^{l} T_3 \to T_4) \cap (\varnothing\ T_5 \to T_6^{\ cl}))\ v_2$ is evaluated to $(v : (\varnothing\ T_1 \to T_2^{\ cl}) \cap (\varnothing\ T_5 \to T_6^{\ cl}))\ (v_2 : (\varnothing\ T_3^{\ 1} : T_3 \Rightarrow^{l} T_1^{\ 1}) \cap (\varnothing\ T_5^{\ 2})) : (\varnothing\ T_2^{\ 1} : T_2 \Rightarrow^{l} T_4^{\ 1}) \cap (\varnothing\ T_6^{\ 2})$. The main

Syntax

$$\begin{aligned}
&\textit{Types } T ::= \textit{Int} \mid \textit{Bool} \mid \textit{Dyn} \mid T \to T \mid T \cap \ldots \cap T\\
&\textit{Expressions } e ::= x^T \mid \lambda x : T \ . \ e \mid e\ e \mid n \mid \textit{true} \mid \textit{false} \mid e : c \cap \ldots \cap c \mid \textit{blame}_T\ l\\
&\textit{Values } v ::= x^T \mid \lambda x : T \ . \ e \mid n \mid \textit{true} \mid \textit{false} \mid \textit{blame}_T\ l \mid v : cv_1 \cap \ldots \cap cv_n \textit{ such that}\\
&\qquad\qquad \neg(\forall_{i \in 1..n} \ . \ cv_i = \textit{blame } T\ T\ l^{\ cl}) \wedge \neg(\forall_{i \in 1..n} \ . \ cv_i = \varnothing\ T^{\ cl})
\end{aligned}$$

$\boxed{e \longrightarrow_{\cap CC} e}$ Evaluation

$$\frac{\Gamma \vdash_{\cap CC} (\textit{blame}_{T_2}\ l)\ e_2 : T_1}{(\textit{blame}_{T_2}\ l)\ e_2 \longrightarrow_{\cap CC} \textit{blame}_{T_1}\ l}\ \text{E-PushBlame1} \qquad \frac{\Gamma \vdash_{\cap CC} e_1\ (\textit{blame}_{T_2}\ l) : T_1}{e_1\ (\textit{blame}_{T_2}\ l) \longrightarrow_{\cap CC} \textit{blame}_{T_1}\ l}\ \text{E-PushBlame2}$$

$$\frac{\vdash_{\cap CI} c_1 : (T_1', T_1) \ \cdots \ \vdash_{\cap CI} c_n : (T_n', T_n)}{\textit{blame}_T\ l : c_1 \cap \ldots \cap c_n \longrightarrow_{\cap CC} \textit{blame}_{T_1 \cap \ldots \cap T_n}\ l}\ \text{E-PushBlameCast} \qquad \frac{e_1 \longrightarrow_{\cap CC} e_1'}{e_1\ e_2 \longrightarrow_{\cap CC} e_1'\ e_2}\ \text{E-App1}$$

$$\frac{e_2 \longrightarrow_{\cap CC} e_2'}{v_1\ e_2 \longrightarrow_{\cap CC} v_1\ e_2'}\ \text{E-App2} \qquad \frac{e \longrightarrow_{\cap CC} e'}{e : c_1 \cap \ldots \cap c_n \longrightarrow_{\cap CC} e' : c_1 \cap \ldots \cap c_n}\ \text{E-Evaluate}$$

$$\frac{}{(\lambda x : T_1 \cap \ldots \cap T_n \ . \ e)\ v \longrightarrow_{\cap CC} [x \mapsto v]e}\ \text{E-AppAbs}$$

$$\frac{\begin{array}{c}\textit{is value } (v_1 : cv_1 \cap \ldots \cap cv_n) \qquad \exists i \in 1..n \ . \ \textit{isArrowCompatible}(cv_i)\\ ((c_{11}, c_{12}, c_1^s), \ldots, (c_{m1}, c_{m2}, c_m^s)) = \textit{simulateArrow}(cv_1, \ldots, cv_n)\end{array}}{(v_1 : cv_1 \cap \ldots \cap cv_n)\ v_2 \longrightarrow_{\cap CC} (v_1 : c_1^s \cap \ldots \cap c_m^s)\ (v_2 : c_{11} \cap \ldots \cap c_{m1}) : c_{12} \cap \ldots \cap c_{m2}}\ \text{E-SimulateArrow}$$

$$\frac{\textit{is value } (v : cv_1 \cap \ldots \cap cv_n) \qquad v : c_1'' \cap \ldots \cap c_j'' = \textit{mergeCasts}(v : cv_1 \cap \ldots \cap cv_n : c_1' \cap \ldots \cap c_m')}{v : cv_1 \cap \ldots \cap cv_n : c_1' \cap \ldots \cap c_m' \longrightarrow_{\cap CC} v : c_1'' \cap \ldots \cap c_j''}\ \text{E-MergeCasts}$$

$$\frac{\neg(\forall i \in 1..n \ . \ \textit{is cast value } c_i) \qquad c_1 \longrightarrow_{\cap CI} cv_1 \ \ldots \ c_n \longrightarrow_{\cap CI} cv_n}{v : c_1 \cap \ldots \cap c_n \longrightarrow_{\cap CC} v : cv_1 \cap \ldots \cap cv_n}\ \text{E-EvaluateCasts}$$

$$\frac{}{v : \textit{blame } I_1\ F_1\ l_1^{\ cl_1} \cap \ldots \cap \textit{blame } I_n\ F_n\ l_n^{\ cl_n} \longrightarrow_{\cap CC} \textit{blame}_{(F_1 \cap \ldots \cap F_n)}\ l_1}\ \text{E-PropagateBlame}$$

$$\frac{}{v : \varnothing\ T_1^{\ cl_1} \cap \ldots \cap \varnothing\ T_n^{\ cl_n} \longrightarrow_{\cap CC} v}\ \text{E-RemoveEmpty}$$

Figure 6: Intersection Cast Calculus Operational Semantics ($\longrightarrow_{\cap CC}$)

difference with C-BETA from [4] is that E-SimulateArrow applies the same treatment to various casts c_i. The marks cl are used to distinguish casts that originate from different instances, thus preventing these casts from being merged together. We use integers as marks and we assign marks to casts based on their index in cast intersections. The neutral mark is 0.

Rule E-MergeCasts merges two cast intersections to allow the evaluation of casts (the definition of mergeCasts, found in Appendix A, was not included due to space limitations). We take the casts from the sub expression and merge every cast against every cast in the expression, provided these casts will be typeable under $\vdash_{\cap CI}$ and share a compatible mark cl. For example, merging the cast $\varnothing\ T_1^{\ cl_1} : T_1 \Rightarrow^l T_2^{\ cl_1}$ with $\varnothing\ T_3^{\ cl_2} : T_3 \Rightarrow^l T_4^{\ cl_2}$ results in $\varnothing\ T_1^{\ cl_1} : T_1 \Rightarrow^l T_2^{\ cl_1} : T_3 \Rightarrow^l T_4^{\ cl_2}$ assuming T_2 equals T_3, thus preserving typeability under $\vdash_{\cap CI}$, and cl_1 and cl_2 are compatible. The two marks cl_1 and cl_2 are compatible if $cl_1 = cl_2$ or either cl_1 or cl_2 is a neutral mark. As rule E-SimulateArrow filters casts and rule E-MergeCasts removes casts that do not type check, we are potentially removing instances of intersection types. Thus our definition of type preservation is a weaker one.

Substitution We change the standard substitution definition to properly instantiate cast intersections, here done by the function $I(T,e)$. Substitution is thus the standard one with the following change:

$$[x \mapsto e]x^T = I(T,e)$$

The definition of $I(T,e)$ follows:

$$I(T, e : c_1 \cap \ldots \cap c_n) = e : c'_1 \cap \ldots \cap c'_m$$
$$\quad \textit{where } T \trianglelefteq S \textit{ and } c'_1 \cap \ldots \cap c'_m = \{c \mid c \in \{c_1, \ldots, c_n\} \textit{ and } \vdash_{\cap CI} c : (T', T'') \textit{ and } T'' \in S\}$$
$$I(T,e) = e, \; \textit{if e is not a cast intersection}$$

One illustrating example of reduction follows:

$(\lambda x : Int \cap Dyn \,.\, x^{Dyn})\ 1 \rightsquigarrow$ (cast insertion)
$((\lambda x : Int \cap Dyn \,.\, x^{Dyn}) : ((\varnothing\ Int \to Dyn\ ^0 : Int \to Dyn \Rightarrow^l Int \to Dyn\ ^0) \cap$
$(\varnothing\ Dyn \to Dyn\ ^0 : Dyn \to Dyn \Rightarrow^l Dyn \to Dyn^0)))$
$(1 : (\varnothing\ Int\ ^0 : Int \Rightarrow^l Int\ ^0) \cap (\varnothing\ Int\ ^0 : Int \Rightarrow^l Dyn\ ^0)) \longrightarrow^*_{\cap CC}$ (E-EvaluateCasts and E-IdentityCI)
$(\lambda x : Int \cap Dyn \,.\, x^{Dyn})\ (1 : (\varnothing\ Int\ ^0) \cap (\varnothing\ Int\ ^0 : Int \Rightarrow^l Dyn\ ^0)) \longrightarrow_{\cap CC}$ (E-AppAbs)
$1 : (\varnothing\ Int\ ^0 : Int \Rightarrow^l Dyn\ ^0)$

4 Correctness Criteria

Consider a partial order $\sqsubseteq$ [3] meaning that a type, or a term, is less precise than another type, or a term (the formal definition of $\sqsubseteq$ is on Appendix A). We want to ensure that less precise programs behave the same way as more precise ones. This property, along with other properties such as those related to type safety, are ensured by correctness criteria, put forth in [16, 3, 4]. Our system observes these correctness criteria, and as such has the following properties (proofs can be found in Appendix B):

Conservative Extension: If e is fully static and T is a static type, then $\Gamma \vdash_{\cap S} e : T \iff \Gamma \vdash_{\cap G} e : T$.
Monotonicity w.r.t. precision: If $\Gamma \vdash_{\cap G} e : T$ and $e' \sqsubseteq e$ then $\Gamma \vdash_{\cap G} e' : T'$ and $T' \sqsubseteq T$.
Type preservation of cast insertion: If $\Gamma \vdash_{\cap G} e : T$ then $\Gamma \vdash_{\cap CC} e \rightsquigarrow e' : T$ and $\Gamma \vdash_{\cap CC} e' : T$.
Monotonicity of cast insertion: If $\Gamma \vdash_{\cap CC} e_1 \rightsquigarrow e'_1 : T_1$ and $\Gamma \vdash_{\cap CC} e_2 \rightsquigarrow e'_2 : T_2$ and $e_1 \sqsubseteq e_2$ then $e'_1 \sqsubseteq e'_2$.
Conservative Extension: If e is fully static, then $e \longrightarrow_{\cap S} e' \iff e \longrightarrow_{\cap CC} e'$.
Type preservation: If $\Gamma \vdash_{\cap CC} e : T_1 \cap \ldots \cap T_n$ and $e \longrightarrow_{\cap CC} e'$ then $\Gamma \vdash_{\cap CC} e' : T_1 \cap \ldots \cap T_m$ such that $m \leq n$.
Progress: If $\Gamma \vdash_{\cap CC} e : T$ then either e is a value or there exists an e' such that $e \longrightarrow_{\cap CC} e'$.

In our system, we simulate simple casts [3, 4] as an instance of cast intersections $e : c_1 \cap \ldots \cap c_n$, when $n = 1$. For example, the expression $1 : Int \Rightarrow^l Dyn$ can be represented in our system as $1 : (\varnothing\ Int\ ^0 : Int \Rightarrow^l Dyn\ ^0)$. The diferent representations are comparable using the equality relation $=_c$ (defined in Appendix A). In fact, our system is an extension to the GTLC, and each program in the GTLC is typed with the same type in our system, and its evaluation produces the same result. The proofs for the following properties are in Appendix B.

Conservative Extension to the GTLC: If e is annotated with only simple types and T is a simple type:
1. then $\Gamma \vdash_G e : T \iff \Gamma \vdash_{\cap G} e : T$.
2. then $\Gamma \vdash_{CC} e \rightsquigarrow e_1 : T \iff \Gamma \vdash_{\cap CC} e \rightsquigarrow e_2 : T$ and $e_1 =_c e_2$.
3. if $\Gamma \vdash_{CC} e_1 : T$, $\Gamma \vdash_{\cap CC} e_2 : T$ and $e_1 =_c e_2$ then $e_1 \longrightarrow_{CC} e'_1 \iff e_2 \longrightarrow_{\cap CC} e'_2$, and $e'_1 =_c e'_2$.

Acknowledgements

We thank Jeremy Siek and Ronald Garcia for providing clarifications on gradual typing. We also thank the anonymous ITRS reviewers for their comments and suggestions, which helped us improve the quality of this paper. This work is partially funded by FCT within project Elven POCI-01-0145-FEDER-016844, Project 9471 - Reforçar a Investigação, o Desenvolvimento Tecnológico e a Inovação (Project 9471-RIDTI) and by Fundo Comunitário Europeu FEDER.

References

[1] Steffen van Bakel (1995): *Intersection type assignment systems*. Theoretical Computer Science 151(2), pp. 385 – 435, doi:10.1016/0304-3975(95)00073-6. 13th Conference on Foundations of Software Technology and Theoretical Computer Science.

[2] Giuseppe Castagna & Victor Lanvin (2017): *Gradual Typing with Union and Intersection Types*. Proc. ACM Program. Lang. 1(ICFP), pp. 41:1–41:28, doi:10.1145/3110285.

[3] Matteo Cimini & Jeremy G. Siek (2016): *The Gradualizer: A Methodology and Algorithm for Generating Gradual Type Systems*. SIGPLAN Not. 51(1), pp. 443–455, doi:10.1145/2914770.2837632.

[4] Matteo Cimini & Jeremy G. Siek (2017): *Automatically Generating the Dynamic Semantics of Gradually Typed Languages*. SIGPLAN Not. 52(1), pp. 789–803, doi:10.1145/3093333.3009863.

[5] M. Coppo & M. Dezani-Ciancaglini (1980): *An extension of the basic functionality theory for the λ-calculus*. Notre Dame J. Formal Logic 21(4), pp. 685–693, doi:10.1305/ndjfl/1093883253.

[6] M. Coppo, M. Dezani-Ciancaglini & B. Venneri: *Functional Characters of Solvable Terms*. Mathematical Logic Quarterly 27(2-6), pp. 45–58, doi:10.1002/malq.19810270205.

[7] Alain Frisch, Giuseppe Castagna & Véronique Benzaken (2008): *Semantic Subtyping: Dealing Set-theoretically with Function, Union, Intersection, and Negation Types*. J. ACM 55(4), pp. 19:1–19:64, doi:10.1145/1391289.1391293.

[8] Ronald Garcia & Matteo Cimini (2015): *Principal Type Schemes for Gradual Programs*. SIGPLAN Not. 50(1), pp. 303–315, doi:10.1145/2775051.2676992.

[9] J. Roger Hindley (1997): *Basic Simple Type Theory*. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, doi:10.1017/CBO9780511608865.

[10] R. Hindley (1969): *The Principal Type-Scheme of an Object in Combinatory Logic*. Transactions of the American Mathematical Society 146, pp. 29–60. Available at `http://www.jstor.org/stable/1995158`.

[11] T. Jim (1995): *Rank 2 Type Systems and Recursive Definitions*. Technical Report, Cambridge, MA, USA.

[12] A. J. Kfoury & J. B. Wells (1999): *Principality and Decidable Type Inference for Finite-rank Intersection Types*. In: Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '99, ACM, New York, NY, USA, pp. 161–174, doi:10.1145/292540.292556.

[13] A.J. Kfoury & J.B. Wells (2004): *Principality and type inference for intersection types using expansion variables*. Theoretical Computer Science 311(1), pp. 1 – 70, doi:10.1016/j.tcs.2003.10.032.

[14] Robin Milner (1978): *A theory of type polymorphism in programming*. Journal of Computer and System Sciences 17(3), pp. 348 – 375, doi:10.1016/0022-0000(78)90014-4.

[15] John C. Reynolds (1997): *Design of the Programming Language Forsythe*, pp. 173–233. Birkhäuser Boston, Boston, MA, doi:10.1007/978-1-4612-4118-8_9.

[16] Jeremy G. Siek, Michael M. Vitousek, Matteo Cimini & John Tang Boyland (2015): *Refined Criteria for Gradual Typing*. In: 1st Summit on Advances in Programming Languages (SNAPL 2015), Leibniz International Proceedings in Informatics (LIPIcs) 32, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 274–293, doi:10.4230/LIPIcs.SNAPL.2015.274.

A Additional Definitions

Syntax

$$\begin{aligned}&\mathit{Types}\ T ::= \ \mathit{Int} \mid \mathit{Bool} \mid T \to T \mid T \cap \ldots \cap T\\&\mathit{Expressions}\ e\ ::= x^T \mid \lambda x : T\ .\ e \mid e\ e \mid n \mid \mathit{true} \mid \mathit{false}\end{aligned}$$

$\boxed{\Gamma \vdash_{\cap S} e : T}$ Typing

$$\frac{\Gamma, x : T_1 \cap \ldots \cap T_n \vdash_{\cap S}\ e : T}{\Gamma \vdash_{\cap S} \lambda x : T_1 \cap \ldots \cap T_n\ .\ e : T_1 \cap \ldots \cap T_n \to T}\ \text{T-ABS}$$

$$\frac{\Gamma \vdash_{\cap S} e_1 : T_1 \cap \ldots \cap T_n \to T \qquad \Gamma \vdash_{\cap S} e_2 : T_1 \cap \ldots \cap T_n}{\Gamma \vdash_{\cap S} e_1\ e_2 : T}\ \text{T-APP} \qquad \frac{x : T' \in \Gamma \qquad T \subseteq T'}{\Gamma \vdash_{\cap S} x^T : T}\ \text{T-VAR}$$

$$\frac{\Gamma, x : T_i \vdash_{\cap S}\ e : T}{\Gamma \vdash_{\cap S} \lambda x : T_1 \cap \ldots \cap T_n\ .\ e : T_i \to T}\ \text{T-ABS'} \qquad \frac{\Gamma \vdash_{\cap S} e : T_1\ \cdots\ \Gamma \vdash_{\cap S} e : T_n}{\Gamma \vdash_{\cap S} e : T_1 \cap \ldots \cap T_n}\ \text{T-GEN}$$

$$\frac{\Gamma \vdash_{\cap S} e : T_1 \cap \ldots \cap T_n}{\Gamma \vdash_{\cap S} e : T_i}\ \text{T-INST} \qquad \frac{}{\Gamma \vdash_{\cap S} n : \mathit{Int}}\ \text{T-INT} \qquad \frac{}{\Gamma \vdash_{\cap S} \mathit{true} : \mathit{Bool}}\ \text{T-TRUE}$$

$$\frac{}{\Gamma \vdash_{\cap S} \mathit{false} : \mathit{Bool}}\ \text{T-FALSE}$$

Figure 7: Static Intersection Type System ($\vdash_{\cap S}$)

$$\boxed{\langle c \rangle^{cl} = c}$$

$$\langle c : T_1 \Rightarrow^l T_2 \ ^{cl} \rangle^{cl'} = \langle c \rangle^{cl'} : T_1 \Rightarrow^l T_2 \ ^{cl'}$$

$$\langle blame\ T_I\ T_F\ l\ ^{cl'} \rangle^{cl} = blame\ T_I\ T_F\ l\ ^{cl}$$

$$\langle \varnothing\ T\ ^{cl'} \rangle^{cl} = \varnothing\ T\ ^{cl}$$

$$\boxed{isArrowCompatible(c) = Bool}$$

$$isArrowCompatible(c : T_{11} \to T_{12} \Rightarrow^l T_{21} \to T_{22}\ ^{cl}) = isArrowCompatible(c)$$

$$isArrowCompatible(\varnothing\ (T_1 \to T_2)\ ^{cl}) = True$$

$$\boxed{separateIntersectionCast(c) = (c, c)}$$

$$separateIntersectionCast(c : T_1 \Rightarrow^l T_2\ ^{cl}) = (\varnothing\ T_1\ ^{cl} : T_1 \Rightarrow^l T_2\ ^{cl}, c)$$

$$separateIntersectionCast(\varnothing\ T\ ^{cl}) = (\varnothing\ T\ ^{cl}, \varnothing\ T\ ^{cl})$$

$$\boxed{breakdownArrowType(c) = (c, c)}$$

$$\begin{gathered} breakdownArrowType(\varnothing\ T_{11} \to T_{12}\ ^{cl} : T_{11} \to T_{12} \Rightarrow^l T_{21} \to T_{22}\ ^{cl}) = \\ (\varnothing\ T_{21}\ ^{cl} : T_{21} \Rightarrow^l T_{11}\ ^{cl}, \varnothing\ T_{12}\ ^{cl} : T_{12} \Rightarrow^l T_{22}\ ^{cl}) \end{gathered}$$

$$breakdownArrowType(\varnothing\ T_1 \to T_2\ ^{cl}) = (\varnothing\ T_1\ ^{cl}, \varnothing\ T_2\ ^{cl}\ )$$

$$\boxed{simulateArrow(c_1, \ldots, c_n) = ((c_{11}, c_{12}, c^s_1), \ldots, (c_{m1}, c_{m2}, c^s_m))}$$

$$\frac{\begin{gathered}(c'_1, \ldots, c'_m) = filter\ isArrowCompatible\ (c_1, \ldots, c_n) \\ ((c^f_1, c^s_1), \ldots, (c^f_m, c^s_m)) = map\ separateIntersectionCast\ (\langle c'_1 \rangle^0, \ldots, \langle c'_m \rangle^0) \\ ((c_{11}, c_{12}), \ldots, (c_{m1}, c_{m2})) = map\ breakdownArrowType\ (\langle c^f_1 \rangle^1, \ldots, \langle c^f_m \rangle^m)\end{gathered}}{simulateArrow(c_1, \ldots, c_n) = ((c_{11}, c_{12}, c^s_1), \ldots, (c_{m1}, c_{m2}, c^s_m))}$$

Figure 8: Definitions for auxiliary semantic functions

$\boxed{getCastLabel(c) = cl}$

$$getCastLabel(c : T_1 \Rightarrow^l T_2\ ^{cl}) = cl$$

$$getCastLabel(blame\ T_I\ T_F\ l\ ^{cl}) = cl$$

$$getCastLabel(\varnothing\ T\ ^{cl}) = cl$$

$\boxed{sameCastLabel(c,c) = Bool}$

$$sameCastLabel(c_1,c_2) = getCastLabel(c_1) == 0$$

$$sameCastLabel(c_1,c_2) = getCastLabel(c_2) == 0$$

$$sameCastLabel(c_1,c_2) = getCastLabel(c_1) == getCastLabel(c_2)$$

$\boxed{joinCasts(c,c) = c}$

$$joinCasts(c : T_1 \Rightarrow^l T_2\ ^{cl}, c') = joinCasts(c,c') : T_1 \Rightarrow^l T_2\ ^{cl}$$

$$joinCasts(blame\ T_I\ T_F\ l\ ^{cl}, c) = blame\ T_I\ T_F\ l\ ^{cl}$$

$$joinCasts(\varnothing\ T\ ^{cl}, c) = \langle c \rangle^{cl}$$

$\boxed{mergeCasts(e) = e}$

$$\frac{\begin{array}{c}(c'_1,\ldots,c'_o) = [joinCast\ y\ x \mid x \leftarrow (c_{11},\ldots,c_{1m}),\ y \leftarrow (c_{21},\ldots,c_{2n}),\\ sameCastLabel\ y\ x\ \&\&\ \vdash_{\cap CI} y : (T_I,T_F)\ \&\&\ \vdash_{\cap CI} x : (T'_I,T'_F)\ \&\&\ T_I == T'_F]\end{array}}{mergeCasts(e : c_{11} \cap \ldots \cap c_{1m} : c_{21} \cap \ldots \cap c_{2n}) = e : c'_1 \cap \ldots \cap c'_o}$$

Figure 9: Definitions for auxiliary semantic functions

$\boxed{T \sqsubseteq T}$ Type Precision

$$Dyn \sqsubseteq T \qquad T \sqsubseteq T \qquad \frac{T_1 \sqsubseteq T_3 \qquad T_2 \sqsubseteq T_4}{T_1 \to T_2 \sqsubseteq T_3 \to T_4} \qquad \frac{T_1 \sqsubseteq T_1' \ \cdots \ T_n \sqsubseteq T_n'}{T_1 \cap \ldots \cap T_n \sqsubseteq T_1' \cap \ldots \cap T_n'}$$

$$\frac{T \sqsubseteq T_1 \ \cdots \ T \sqsubseteq T_n}{T \sqsubseteq T_1 \cap \ldots \cap T_n} \qquad \frac{T_1 \sqsubseteq T \ \cdots \ T_n \sqsubseteq T}{T_1 \cap \ldots \cap T_n \sqsubseteq T}$$

$\boxed{c \sqsubseteq c}$ Cast Precision

$$\frac{c \sqsubseteq c' \qquad T_1 \sqsubseteq T_1' \qquad T_2 \sqsubseteq T_2'}{c : T_1 \Rightarrow^{l} T_2 \ ^{cl} \sqsubseteq c' : T_1' \Rightarrow^{l'} T_2' \ ^{cl'}} \qquad \frac{c \sqsubseteq c' \qquad \vdash_{\cap CI} c' : (T', T) \qquad T_1 \sqsubseteq T \qquad T_2 \sqsubseteq T}{c : T_1 \Rightarrow^{l} T_2 \ ^{cl} \sqsubseteq c'}$$

$$\frac{c \sqsubseteq c' \qquad \vdash_{\cap CI} c : (T', T) \qquad T \sqsubseteq T_1 \qquad T \sqsubseteq T_2}{c \sqsubseteq c' : T_1 \Rightarrow^{l} T_2 \ ^{cl}} \qquad \frac{\vdash_{\cap CI} c : (T_I, T_F) \qquad T_I \sqsubseteq T_I' \qquad T_F \sqsubseteq T_F'}{c \sqsubseteq blame \ T_I' \ T_F' \ l' \ ^{cl'}}$$

$$\frac{T \sqsubseteq T'}{\varnothing \ T \ ^{cl} \sqsubseteq \varnothing \ T' \ ^{cl'}}$$

$\boxed{e \sqsubseteq e}$ Expression Precision

$$\frac{T \sqsubseteq T'}{x^{T} \sqsubseteq x^{T'}} \qquad \frac{T \sqsubseteq T' \qquad e \sqsubseteq e'}{\lambda x : T \ . \ e \sqsubseteq \lambda x : T' \ . \ e'} \qquad \frac{e_1 \sqsubseteq e_1' \qquad e_2 \sqsubseteq e_2'}{e_1 \ e_2 \sqsubseteq e_1' \ e_2'} \qquad n \sqsubseteq n \qquad true \sqsubseteq true$$

$$false \sqsubseteq false \qquad \frac{e \sqsubseteq e' \qquad c_1 \sqsubseteq c_1' \ \cdots \ c_n \sqsubseteq c_n'}{e : c_1 \cap \ldots \cap c_n \sqsubseteq e' : c_1' \cap \ldots \cap c_n'}$$

$$\frac{e \sqsubseteq e' \qquad \Gamma \vdash_{\cap CC} e' : T \qquad \vdash_{\cap CI} c_1 : (T_1', T_1) \ \cdots \ \vdash_{\cap CI} c_n : (T_n', T_n) \qquad T_1 \cap \ldots \cap T_n \sqsubseteq T}{e : c_1 \cap \ldots \cap c_n \sqsubseteq e'}$$

$$\frac{e \sqsubseteq e' \qquad \Gamma \vdash_{\cap CC} e : T \qquad \vdash_{\cap CI} c_1 : (T_1', T_1) \ \cdots \ \vdash_{\cap CI} c_n : (T_n', T_n) \qquad T \sqsubseteq T_1 \cap \ldots \cap T_n}{e \sqsubseteq e' : c_1 \cap \ldots \cap c_n}$$

$$\frac{\Gamma \vdash_{\cap CC} e : T \qquad T \sqsubseteq T'}{e \sqsubseteq blame_{T'} \ l}$$

Figure 10: Precision ($\sqsubseteq$)

$\boxed{e =_c e}$ Equality of Casts

$$x =_c x^T \qquad n =_c n \qquad \mathit{true} =_c \mathit{true} \qquad \mathit{false} =_c \mathit{false} \qquad \mathit{blame}_T\ l =_c \mathit{blame}_T\ l$$

$$\frac{e =_c e'}{\lambda x : T\ .\ e =_c \lambda x : T\ .\ e'} \qquad \frac{e_1 =_c e_1' \qquad e_2 =_c e_2'}{e_1\ e_2 =_c e_1'\ e_2'} \qquad \frac{e =_c e'}{e =_c e' : (\varnothing\ T^{\ cl})}$$

$$\mathit{blame}_T\ l =_c e : (\mathit{blame}\ T'\ T\ l^{\ cl}) \qquad \frac{e =_c e' : c}{e : T_1 \Rightarrow^l T_2 =_c e' : (c : T_1 \Rightarrow^l T_2^{\ cl})}$$

Figure 11: Equality of Casts

B Proofs

Lemma B.1 (Consistency reduces to equality when comparing static types)**.** *If T_1 and T_2 are static types then $T_1 = T_2 \iff T_1 \sim T_2$.*

Proof. We proceed by structural induction on T_1.

Base cases:

- $T_1 = Int$.
  - If $Int = Int$ then by the definition of $\sim$, $Int \sim Int$.
  - If $Int \sim Int$, then $Int = Int$.
- $T_1 = Bool$.
  - If $Bool = Bool$ then by the definition of $\sim$, $Bool \sim Bool$.
  - If $Bool \sim Bool$, then $Bool = Bool$.

Induction step:

- $T_1 = T_{11} \to T_{12}$.
  - If $T_{11} \to T_{12} = T_{21} \to T_{22}$, for some T_{21} and T_{22}, then $T_{11} = T_{21}$ and $T_{12} = T_{22}$. By the induction hypothesis, $T_{11} \sim T_{21}$ and $T_{12} \sim T_{22}$. Therefore, by the definition of $\sim$, $T_{11} \to T_{12} \sim T_{21} \to T_{22}$.
  - If $T_{11} \to T_{12} \sim T_2$, then by the definition of $\sim$, $T_2 = T_{21} \to T_{22}$ and $T_{11} \sim T_{21}$ and $T_{12} \sim T_{22}$. By the induction hypothesis, $T_{11} = T_{21}$ and $T_{12} = T_{22}$. Therefore, $T_{11} \to T_{12} = T_{21} \to T_{22}$.
- $T_1 = T_{11} \cap \ldots \cap T_{1n}$.
  - If $T_{11} \cap \ldots \cap T_{1n} = T_2$, then $\exists T_{21} \ldots T_{2n} \,.\, T_2 = T_{21} \cap \ldots \cap T_{2n}$ and $T_{11} = T_{21}$ and $\ldots$ and $T_{1n} = T_{2n}$. By the induction hypothesis, $T_{11} \sim T_{21}$ and $\ldots$ and $T_{1n} \sim T_{2n}$. Therefore, by the definition of $\sim$, $T_{11} \cap \ldots \cap T_{1n} \sim T_{21} \cap \ldots \cap T_{2n}$.
  - If $T_{11} \cap \ldots \cap T_{1n} \sim T_2$, then either:
    - $\exists T_{21} \ldots T_{2n} \,.\, T_2 = T_{21} \cap \ldots \cap T_{2n}$ and $T_{11} \sim T_{21}$ and $\ldots$ and $T_{1n} \sim T_{2n}$. By the induction hypothesis, $T_{11} = T_{21}$ and $\ldots$ and $T_{1n} = T_{2n}$. Therefore, $T_{11} \cap \ldots \cap T_{1n} = T_{21} \cap \ldots \cap T_{2n}$.
    - $T_{11} \sim T_2$ and $\ldots$ and $T_{1n} \sim T_2$. By the induction hypothesis, $T_{11} = T_2$ and $\ldots$ and $T_{1n} = T_2$. As $T_2 \cap \ldots \cap T_2 = T_2$, then $T_{11} \cap \ldots \cap T_{1n} = T_2$.

$\square$

Lemma B.2 (Type preservation of $\longrightarrow_{\cap CI}$)**.** *If $c \longrightarrow_{\cap CI} c'$ and $\vdash_{\cap CI} c : (T_I, T_F)$ then $\vdash_{\cap CI} c' : (T_I, T_F)$.*

Proof. We proceed by induction on the length of the derivation tree of $\longrightarrow_{\cap CI}$.

Base cases:

- Rule E-PushBlameCI. If $\vdash_{\cap CI}$ *blame* $T_I\ T_F\ l_1\ ^{cl_1} : T_1 \Rightarrow^{l_2} T_2\ ^{cl_2} : (T_I, T_2)$ and by rule E-PushBlameCI, *blame* $T_I\ T_F\ l_1\ ^{cl_1} : T_1 \Rightarrow^{l_2} T_2\ ^{cl_2} \longrightarrow_{\cap CI}$ *blame* $T_I\ T_2\ l_1\ ^{cl_1}$, then by rule T-BlameCI, $\vdash_{\cap CI}$ *blame* $T_I\ T_2\ l_1\ ^{cl_1} : (T_I, T_2)$.
- Rule E-IdentityCI. If $\vdash_{\cap CI} cv1 : T \Rightarrow^{l} T\ ^{cl} : (T_I, T)$, then by rule T-SingleCI, $\vdash_{\cap CI} cv1 : (T_I, T)$. By rule E-IdentityCI, $cv1 : T \Rightarrow^{l} T\ ^{cl} \longrightarrow_{\cap CI} cv1$.
- Rule E-SucceedCI. If $\vdash_{\cap CI} cv1 : G \Rightarrow^{l_1} Dyn\ ^{cl_1} : Dyn \Rightarrow^{l_2} G\ ^{cl_2} : (T_I, G)$, then by rule T-SingleCI, $\vdash_{\cap CI} cv1 : (T_I, G)$. By rule E-SucceedCI, $cv1 : G \Rightarrow^{l_1} Dyn\ ^{cl_1} : Dyn \Rightarrow^{l_2} G\ ^{cl_2} \longrightarrow_{\cap CI} cv1$.

- Rule E-FailCI. If $\vdash_{\cap CI} cv1 : G_1 \Rightarrow^{l_1} Dyn^{\ cl_1} : Dyn \Rightarrow^{l_2} G_2^{\ cl_2} : (T_I, G_2)$, and by rule E-FailCI, $cv1 : G_1 \Rightarrow^{l_1} Dyn^{\ cl_1} : Dyn \Rightarrow^{l_2} G_2^{\ cl_2} \longrightarrow_{\cap CI} blame\ T_I\ G_2\ l_2^{\ cl_1}$ then by rule T-BlameCI, $\vdash_{\cap CI} blame\ T_I\ G_2\ l_2^{\ cl_1} : (T_I, G_2)$.
- Rule E-GroundCI. If $\vdash_{\cap CI} cv1 : T \Rightarrow^{l} Dyn^{\ cl} : (T_I, Dyn)$ then by rule T-SingleCI, $\vdash_{\cap CI} cv1 : (T_I, T)$. By rule E-GroundCI, $cv1 : T \Rightarrow^{l} Dyn^{\ cl} \longrightarrow_{\cap CI} cv1 : T \Rightarrow^{l} G^{\ cl} : G \Rightarrow^{l} Dyn^{\ cl}$, then by rule T-SingleCI, $\vdash_{\cap CI} cv1 : T \Rightarrow^{l} G^{\ cl} : G \Rightarrow^{l} Dyn^{\ cl} : (T_I, Dyn)$.
- Rule E-ExpandCI. If $\vdash_{\cap CI} cv1 : Dyn \Rightarrow^{l} T^{\ cl} : (T_I, T)$ then by rule T-SingleCI, $\vdash_{\cap CI} cv1 : (T_I, Dyn)$. By rule E-ExpandCI, $cv1 : Dyn \Rightarrow^{l} T^{\ cl} \longrightarrow_{\cap CI} cv1 : Dyn \Rightarrow^{l} G^{\ cl} : G \Rightarrow^{l} T^{\ cl}$, then by rule T-SingleCI, $\vdash_{\cap CI} cv1 : Dyn \Rightarrow^{l} G^{\ cl} : G \Rightarrow^{l} T^{\ cl} : (T_I, T)$.

Induction step:

- Rule E-EvaluateCI. If $\vdash_{\cap CI} c : T_1 \Rightarrow^{l} T_2^{\ cl} : (T_I, T_2)$ then by rule T-SingleCI, $\vdash_{\cap CI} c : (T_I, T_1)$. By rule E-EvaluateCI, $c \longrightarrow_{\cap CI} c'$. By the induction hypothesis, $\vdash_{\cap CI} c' : (T_I, T_1)$. By rule E-EvaluateCI, $c : T_1 \Rightarrow^{l} T_2^{\ cl} \longrightarrow_{\cap CI} c' : T_1 \Rightarrow^{l} T_2^{\ cl}$, then by rule T-SingleCI, $\vdash_{\cap CI} c' : T_1 \Rightarrow^{l} T_2^{\ cl} : (T_I, T_2)$.

□

Lemma B.3 (Progress of $\longrightarrow_{\cap CI}$)**.** *If $\Gamma \vdash_{\cap CI} c : (T_I, T_F)$ then either c is a cast value or there exists a c' such that $c \longrightarrow_{\cap CI} c'$.*

Proof. We proceed by induction on the length of the derivation tree of $\vdash_{\cap CI} c : (T, T)$.

Base cases:

- Rule T-BlameCI. As $\vdash_{\cap CI} blame\ T_I\ T_F\ l^{\ cl} : (T_I, T_F)$ and $blame\ T_I\ T_F\ l^{\ cl}$ is a cast value, it is proved.
- Rule T-EmptyCI. As $\vdash_{\cap CI} \varnothing\ T^{\ cl} : (T, T)$ and $\varnothing\ T^{\ cl}$ is a cast value, it is proved.

Induction step:

- Rule T-SingleCI. If $\vdash_{\cap CI} c : T_1 \Rightarrow^{l} T_2^{\ cl} : (T_I, T_2)$ then by rule T-SingleCI, $\vdash_{\cap CI} c : (T_I, T_1)$. By the induction hypothesis, either c is a cast value or there is a c' such that $c \longrightarrow_{\cap CI} c'$. If c is a cast value, then c can either be of the form $blame\ T_I\ T_F\ l^{\ cl}$, in which case by rule E-PushBlameCI, $blame\ T_I\ T_F\ l_1^{\ cl_1} : T_1 \Rightarrow^{l_2} T_2^{\ cl_2} \longrightarrow_{\cap CI} blame\ T_I\ T_2\ l_1^{\ cl_1}$ or c is a cast value 1. If c is a cast value 1 then $c : T_1 \Rightarrow^{l} T_2{}^{c}l$ can be of one of the folowing forms:
  - $cv1 : T \Rightarrow^{l} T^{\ cl}$. Then by rule E-IdentityCI, $cv1 : T \Rightarrow^{l} T^{\ cl} \longrightarrow_{\cap CI} cv1$.
  - $cv1 : G \Rightarrow^{l_1} Dyn^{\ cl_1} : Dyn \Rightarrow^{l_2} G^{\ cl_2}$. Then by rule E-SucceedCI, $cv1 : G \Rightarrow^{l_1} Dyn^{\ cl_1} : Dyn \Rightarrow^{l_2} G^{\ cl_2} \longrightarrow_{\cap CI} cv1$.
  - $cv1 : G_1 \Rightarrow^{l_1} Dyn^{\ cl_1} : Dyn \Rightarrow^{l_2} G_2^{\ cl_2}$. Then by rule E-FailCI, $cv1 : G_1 \Rightarrow^{l_1} Dyn^{\ cl_1} : Dyn \Rightarrow^{l_2} G_2^{\ cl_2} \longrightarrow_{\cap CI} blame\ T_I\ G_2\ l_2^{\ cl_1}$.
  - $cv1 : T \Rightarrow^{l} Dyn^{\ cl}$. Then by rule E-GroundCI, $cv1 : T \Rightarrow^{l} Dyn^{\ cl} \longrightarrow_{\cap CI} cv1 : T \Rightarrow^{l} G^{\ cl} : G \Rightarrow^{l} Dyn^{\ cl}$.
  - $cv1 : Dyn \Rightarrow^{l} T^{\ cl}$. Then by rule E-ExpandCI, $cv1 : Dyn \Rightarrow^{l} T^{\ cl} \longrightarrow_{\cap CI} cv1 : Dyn \Rightarrow^{l} G^{\ cl} : G \Rightarrow^{l} T^{\ cl}$.

  If there is a c' such that $c \longrightarrow_{\cap CI} c'$, then by rule E-EvaluateCI, $c : T_1 \Rightarrow^{l} T_2{}^{c}l \longrightarrow_{\cap CI} c' : T_1 \Rightarrow^{l} T_2{}^{c}l$.

□

Conservative Extension: If e is fully static and T is a static type, then $\Gamma \vdash_{\cap S} e : T \iff \Gamma \vdash_{\cap G} e : T$.
Monotonicity w.r.t. precision: If $\Gamma \vdash_{\cap G} e : T$ and $e' \sqsubseteq e$ then $\Gamma \vdash_{\cap G} e' : T'$ and $T' \sqsubseteq T$.
Type preservation of cast insertion: If $\Gamma \vdash_{\cap G} e : T$ then $\Gamma \vdash_{\cap CC} e \rightsquigarrow e' : T$ and $\Gamma \vdash_{\cap CC} e' : T$.
Monotonicity of cast insertion: If $\Gamma \vdash_{\cap CC} e_1 \rightsquigarrow e'_1 : T_1$ and $\Gamma \vdash_{\cap CC} e_2 \rightsquigarrow e'_2 : T_2$ and $e_1 \sqsubseteq e_2$ then $e'_1 \sqsubseteq e'_2$.
Conservative Extension: If e is fully static, then $e \longrightarrow_{\cap S} e' \iff e \longrightarrow_{\cap CC} e'$.
Type preservation: If $\Gamma \vdash_{\cap CC} e : T_1 \cap \ldots \cap T_n$ and $e \longrightarrow_{\cap CC} e'$ then $\Gamma \vdash_{\cap CC} e' : T_1 \cap \ldots \cap T_m$ such that $m \leq n$.
Progress: If $\Gamma \vdash_{\cap CC} e : T$ then either e is a value or there exists an e' such that $e \longrightarrow_{\cap CC} e'$.

Theorem B.1 (Conservative Extension)**.** *Depends on Lemma B.1. If e is fully static and T is a static type, then* $\Gamma \vdash_{\cap S} e : T \iff \Gamma \vdash_{\cap G} e : T$.

Proof. We proceed by induction on the length of the derivation tree of $\vdash_{\cap S}$ and $\vdash_{\cap G}$ for the right and left direction of the implication, respectively.

Base cases:

- Rule T-Var.
  - If $\Gamma \vdash_{\cap S} x^T : T$, then $x : T' \in \Gamma$ and $T \subseteq T'$. Therefore, $\Gamma \vdash_{\cap G} x^T : T$.
  - If $\Gamma \vdash_{\cap G} x^T : T$, then $x : T' \in \Gamma$ and $T \subseteq T'$. Therefore, $\Gamma \vdash_{\cap S} e^T : T$.
- Rule T-Int.
  - If $\Gamma \vdash_{\cap S} n : Int$, then $\Gamma \vdash_{\cap G} n : Int$.
  - If $\Gamma \vdash_{\cap G} n : Int$, then $\Gamma \vdash_{\cap S} n : Int$.
- Rule T-True.
  - If $\Gamma \vdash_{\cap S} true : Bool$, then $\Gamma \vdash_{\cap G} true : Bool$.
  - If $\Gamma \vdash_{\cap G} true : Bool$, then $\Gamma \vdash_{\cap S} true : Bool$.
- Rule T-False.
  - If $\Gamma \vdash_{\cap S} false : Bool$, then $\Gamma \vdash_{\cap G} false : Bool$.
  - If $\Gamma \vdash_{\cap G} false : Bool$, then $\Gamma \vdash_{\cap S} false : Bool$.

Induction step:

- Rule T-Abs.
  - If $\Gamma \vdash_{\cap S} \lambda x : T_1 \cap \ldots \cap T_n \;.\; e : T_1 \cap \ldots \cap T_n \rightarrow T$, then $\Gamma, x : T_1 \cap \ldots \cap T_n \vdash_{\cap S} e : T$. By the induction hypothesis, $\Gamma, x : T_1 \cap \ldots \cap T_n \vdash_{\cap G} e : T$. Therefore, $\Gamma \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n \;.\; e : T_1 \cap \ldots \cap T_n \rightarrow T$.
  - If $\Gamma \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n \;.\; e : T_1 \cap \ldots \cap T_n \rightarrow T$, then $\Gamma, x : T_1 \cap \ldots \cap T_n \vdash_{\cap G} e : T$. By the induction hypothesis, $\Gamma, x : T_1 \cap \ldots \cap T_n \vdash_{\cap S} e : T$. Therefore, $\Gamma \vdash_{\cap S} \lambda x : T_1 \cap \ldots \cap T_n \;.\; e : T_1 \cap \ldots \cap T_n \rightarrow T$.
- Rule T-Abs'.
  - If $\Gamma \vdash_{\cap S} \lambda x : T_1 \cap \ldots \cap T_n \;.\; e : T_i \rightarrow T$, then $\Gamma, x : T_i \vdash_{\cap S} e : T$. By the induction hypothesis, $\Gamma, x : T_i \vdash_{\cap G} e : T$. Therefore, $\Gamma \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n \;.\; e : T_i \rightarrow T$.
  - If $\Gamma \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n \;.\; e : T_i \rightarrow T$, then $\Gamma, x : T_i \vdash_{\cap G} e : T$. By the induction hypothesis, $\Gamma, x : T_i \vdash_{\cap S} e : T$. Therefore, $\Gamma \vdash_{\cap S} \lambda x : T_1 \cap \ldots \cap T_n \;.\; e : T_i \rightarrow T$.
- Rule T-App.

  - If $\Gamma \vdash_{\cap S} e_1\ e_2 : T$ then $\Gamma \vdash_{\cap S} e_1 : T_1 \cap \ldots \cap T_n \to T$ and $\Gamma \vdash_{\cap S} e_2 : T_1 \cap \ldots \cap T_n$. By the induction hypothesis, $\Gamma \vdash_{\cap G} e_1\ : T_1 \cap \ldots \cap T_n \to T$ and $\Gamma \vdash_{\cap G} e_2 : T_1 \cap \ldots \cap T_n$. By the definition of $\rhd$, $T_1 \cap \ldots \cap T_n \to T \rhd T_1 \cap \ldots \cap T_n \to T$. By the definition of $\sim$, $T_1 \cap \ldots \cap T_n \sim T_1 \cap \ldots \cap T_n$. Therefore, $\Gamma \vdash_{\cap G} e_1\ e_2 : T$.
  - If $\Gamma \vdash_{\cap G} e_1\ e_2 : T$ then $\Gamma \vdash_{\cap G} e_1 : PM$, $PM \rhd T_1 \cap \ldots \cap T_n \to T$, $\Gamma \vdash_{\cap G} e_2 : T_1' \cap \ldots \cap T_n'$ and $T_1' \cap \ldots \cap T_n' \sim T_1 \cap \ldots \cap T_n$. By the definition of $\rhd$, $PM = T_1 \cap \ldots \cap T_n \to T$, therefore $\Gamma \vdash_{\cap G} e_1 : T_1 \cap \ldots \cap T_n \to T$. By Lemma B.1, $T_1' \cap \ldots \cap T_n' = T_1 \cap \ldots \cap T_n$, and therefore $\Gamma \vdash_{\cap G} e_2 : T_1 \cap \ldots \cap T_n$. By the induction hypothesis, $\Gamma \vdash_{\cap S} e_1\ : T_1 \cap \ldots \cap T_n \to T$ and $\Gamma \vdash_{\cap S} e_2 : T_1 \cap \ldots \cap T_n$. Therefore, $\Gamma \vdash_{\cap S} e_1\ e_2 : T$.

- Rule T-Gen.
  - If $\Gamma \vdash_{\cap S} e : T_1 \cap \ldots \cap T_n$ then $\Gamma \vdash_{\cap S} e : T_1$ and … and $\Gamma \vdash_{\cap S} e : T_n$. By the induction hypothesis, $\Gamma \vdash_{\cap G} e : T_1$ and … and $\Gamma \vdash_{\cap G} e : T_n$. Therefore, $\Gamma \vdash_{\cap G} e : T_1 \cap \ldots \cap T_n$.
  - If $\Gamma \vdash_{\cap G} e : T_1 \cap \ldots \cap T_n$ then $\Gamma \vdash_{\cap G} e : T_1$ and … and $\Gamma \vdash_{\cap G} e : T_n$. By the induction hypothesis, $\Gamma \vdash_{\cap S} e : T_1$ and … and $\Gamma \vdash_{\cap S} e : T_n$. Therefore $\Gamma \vdash_{\cap S} e : T_1 \cap \ldots \cap T_n$.

- Rule T-Inst.
  - If $\Gamma \vdash_{\cap S} e : T_i$ then $\Gamma \vdash_{\cap S} e : T_1 \cap \ldots \cap T_n$, such that $T_i \in \{T_1, \ldots, T_n\}$. By the induction hypothesis, $\Gamma \vdash_{\cap G} e : T_1 \cap \ldots \cap T_n$. As $T_i \in \{T_1, \ldots, T_n\}$, then $\Gamma \vdash_{\cap G} e : T_i$.
  - If $\Gamma \vdash_{\cap G} e : T_i$ then $\Gamma \vdash_{\cap G} e : T_1 \cap \ldots \cap T_n$, such that $T_i \in \{T_1, \ldots, T_n\}$. By the induction hypothesis, $\Gamma \vdash_{\cap S} e : T_1 \cap \ldots \cap T_n$. As $T_i \in \{T_1, \ldots, T_n\}$, then $\Gamma \vdash_{\cap S} e : T_i$.

□

Theorem B.2 (Monotonicity w.r.t. precision)**.** *If* $\Gamma \vdash_{\cap G} e : T$ *and* $e' \sqsubseteq e$ *then* $\Gamma \vdash_{\cap G} e' : T'$ *and* $T' \sqsubseteq T$.

Proof. We proceed by induction on the length of the derivation tree of $\Gamma \vdash_{\cap G} e : T$.

Base cases:

- Rule T-Var. If $\Gamma \vdash_{\cap G} x^T : T$ and $x^{T'} \sqsubseteq x^T$, then $\Gamma \vdash_{\cap G} x^{T'} : T'$ and $T' \sqsubseteq T$.
- Rule T-Int. If $\Gamma \vdash_{\cap G} n : Int$ and $n \sqsubseteq n$, then $\Gamma \vdash_{\cap G} n : Int$ and $Int \sqsubseteq Int$.
- Rule T-True. If $\Gamma \vdash_{\cap G} true : Bool$ and $true \sqsubseteq true$, then $\Gamma \vdash_{\cap G} true : Bool$ and $Bool \sqsubseteq Bool$.
- Rule T-False. If $\Gamma \vdash_{\cap G} false : Bool$ and $false \sqsubseteq false$, then $\Gamma \vdash_{\cap G} false : Bool$ and $Bool \sqsubseteq Bool$.

Induction step:

- Rule T-Abs. If $\Gamma \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n\ .\ e : T_1 \cap \ldots \cap T_n \to T$ and $\lambda x : T_1' \cap \ldots \cap T_n'\ .\ e' \sqsubseteq \lambda x : T_1 \cap \ldots \cap T_n\ .\ e$, then by rule T-Abs, $\Gamma, x : T_1 \cap \ldots \cap T_n \vdash_{\cap G} e : T$, and by the definition of $\sqsubseteq$, $T_1' \cap \ldots \cap T_n' \sqsubseteq T_1 \cap \ldots \cap T_n$ and $e' \sqsubseteq e$. By the induction hypothesis, $\Gamma, x : T_1' \cap \ldots \cap T_n' \vdash_{\cap G} e' : T'$ and $T' \sqsubseteq T$. By rule T-Abs, $\Gamma \vdash_{\cap G} \lambda x : T_1' \cap \ldots \cap T_n'\ .\ e' : T_1' \cap \ldots \cap T_n' \to T'$, and by the definition of $\sqsubseteq$, $T_1' \cap \ldots \cap T_n' \to T' \sqsubseteq T_1 \cap \ldots \cap T_n \to T$.
- Rule T-Abs'. If $\Gamma \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n\ .\ e : T_i \to T$ and $\lambda x : T_1' \cap \ldots \cap T_n'\ .\ e' \sqsubseteq \lambda x : T_1 \cap \ldots \cap T_n\ .\ e$, then by rule T-Abs', $\Gamma, x : T_i \vdash_{\cap G} e : T$, and by the definition of $\sqsubseteq$, $T_1' \cap \ldots \cap T_n' \sqsubseteq T_1 \cap \ldots \cap T_n$ and $e' \sqsubseteq e$. By the induction hypothesis, $\Gamma, x : T_i' \vdash_{\cap G} e' : T'$ and $T' \sqsubseteq T$. By rule T-Abs', $\Gamma \vdash_{\cap G} \lambda x : T_1' \cap \ldots \cap T_n'\ .\ e' : T_i' \to T'$, and by the definition of $\sqsubseteq$, $T_i' \to T' \sqsubseteq T_i \to T$.

- Rule T-App. If $\Gamma \vdash_{\cap G} e_1\ e_2 : T$ and $e'_1\ e'_2 \sqsubseteq e_1\ e_2$ then by rule T-App, $\Gamma \vdash_{\cap G} e_1 : PM$, $PM \rhd T_{11} \cap \ldots \cap T_{1n} \to T$, $\Gamma \vdash_{\cap G} e_2 : T_{21} \cap \ldots \cap T_{2n}$, and $T_{21} \cap \ldots \cap T_{2n} \sim T_{11} \cap \ldots \cap T_{1n}$, and by the definition of $\sqsubseteq$, $e'_1 \sqsubseteq e_1$ and $e'_2 \sqsubseteq e_2$. By the induction hypothesis, $\Gamma \vdash_{\cap G} e'_1 : PM'\ and\ PM' \sqsubseteq PM\ and\ PM' \rhd T'_{11} \cap \ldots \cap T'_{1n} \to T'$ and $\Gamma \vdash_{\cap G} e'_2 : T'_{21} \cap \ldots \cap T'_{2n}\ and\ T'_{21} \cap \ldots \cap T'_{2n} \sqsubseteq T_{21} \cap \ldots \cap T_{2n}\ and\ T'_{21} \cap \ldots \cap T'_{2n} \sim T'_{11} \cap \ldots \cap T'_{1n}$. By the definition of $\sqsubseteq$ and $\rhd$, $T'_{11} \cap \ldots \cap T'_{1n} \to T' \sqsubseteq T_{11} \cap \ldots \cap T_{1n} \to T$, and therefore, $T' \sqsubseteq T$. As $\Gamma \vdash_{\cap G} e'_1\ e'_2 : T'$, it is proved.
- Rule T-Gen. If $\Gamma \vdash_{\cap G} e : T_1 \cap \ldots \cap T_n$ and $e' \sqsubseteq e$, then by rule T-Gen, $\Gamma \vdash_{\cap G} e : T_1$ and ... and $\Gamma \vdash_{\cap G} e : T_n$. By the induction hypothesis, $\Gamma \vdash_{\cap G} e' : T'_1\ and\ T'_1 \sqsubseteq T_1$ and ... and $\Gamma \vdash_{\cap G} e' : T'_n\ and\ T'_n \sqsubseteq T_n$. Then by rule T-Gen, $\Gamma \vdash_{\cap G} e' : T'_1 \cap \ldots \cap T'_n$ and by the definition of $\sqsubseteq$, $T'_1 \cap \ldots \cap T'_n \sqsubseteq T_1 \cap \ldots \cap T_n$.
- Rule T-Inst. If $\Gamma \vdash_{\cap G} e : T_i$ and $e' \sqsubseteq e$, then by rule T-Inst, $\Gamma \vdash_{\cap G} e : T_1 \cap \ldots \cap T_n$ such that $T_i \in \{T_1, \ldots, T_n\}$. By the induction hypothesis, $\Gamma \vdash_{\cap G} e' : T'_1 \cap \ldots \cap T'_n$ and $T'_1 \cap \ldots \cap T'_n \sqsubseteq T_1 \cap \ldots \cap T_n$. Therefore, by rule T-Inst, $\Gamma \vdash_{\cap G} e' : T'_i$ and by the definition of $\sqsubseteq$, $T'_i \sqsubseteq T_i$.

□

Theorem B.3 (Type preservation of cast insertion). *If* $\Gamma \vdash_{\cap G} e : T$ *then* $\Gamma \vdash_{\cap CC} e \rightsquigarrow e' : T$ *and* $\Gamma \vdash_{\cap CC} e' : T$.

Proof. We proceed by induction on the length of the derivation tree of $\Gamma \vdash_{\cap G} e : T$.

Base cases:

- Rule T-Var. If $\Gamma \vdash_{\cap G} x^T : T$, then by rule T-Var, $x : T' \in \Gamma$ and $T \subseteq T'$. By rule C-Var, $\Gamma \vdash_{\cap CC} x^T \rightsquigarrow x^T : T$ and by rule T-Var, $\Gamma \vdash_{\cap CC} x^T : T$.
- Rule T-Int. As $\Gamma \vdash_{\cap G} n : Int$, then by rule C-Int, $\Gamma \vdash_{\cap CC} n \rightsquigarrow n : Int$ and by rule T-Int, $\Gamma \vdash_{\cap CC} n : Int$.
- Rule T-True. As $\Gamma \vdash_{\cap G} true : Bool$, then by rule C-True, $\Gamma \vdash_{\cap CC} true \rightsquigarrow true : Bool$ and by rule T-True, $\Gamma \vdash_{\cap CC} true : Bool$.
- Rule T-False. As $\Gamma \vdash_{\cap G} false : Bool$, then by rule C-False, $\Gamma \vdash_{\cap CC} false \rightsquigarrow false : Bool$ and by rule T-False, $\Gamma \vdash_{\cap CC} false : Bool$, it is proved.

Induction step:

- Rule T-Abs. If $\Gamma \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n\ .\ e : T_1 \cap \ldots \cap T_n \to T$ then by rule T-Abs, $\Gamma, x : T_1 \cap \ldots \cap T_n \vdash_{\cap G} e : T$. By the induction hypothesis, $\Gamma, x : T_1 \cap \ldots \cap T_n \vdash_{\cap CC} e \rightsquigarrow e' : T$ and $\Gamma, x : T_1 \cap \ldots \cap T_n \vdash_{\cap CC} e' : T$. By rule C-Abs, $\Gamma \vdash_{\cap CC} \lambda x : T_1 \cap \ldots \cap T_n\ .\ e \rightsquigarrow \lambda x : T_1 \cap \ldots \cap T_n\ .\ e' : T_1 \cap \ldots \cap T_n \to T$ and by rule T-Abs, $\Gamma \vdash_{\cap CC} \lambda x : T_1 \cap \ldots \cap T_n\ .\ e' : T_1 \cap \ldots \cap T_n \to T$.
- Rule T-Abs'. If $\Gamma \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n\ .\ e : T_i \to T$ then by rule T-Abs', $\Gamma, x : T_i \vdash_{\cap G} e : T$. By the induction hypothesis, $\Gamma, x : T_i \vdash_{\cap CC} e \rightsquigarrow e' : T$ and $\Gamma, x : T_i \vdash_{\cap CC} e' : T$. By rule C-Abs', $\Gamma \vdash_{\cap CC} \lambda x : T_1 \cap \ldots \cap T_n\ .\ e \rightsquigarrow \lambda x : T_1 \cap \ldots \cap T_n\ .\ e' : T_i \to T$ and by rule T-Abs', $\Gamma \vdash_{\cap CC} \lambda x : T_1 \cap \ldots \cap T_n\ .\ e' : T_i \to T$.
- Rule T-App. If $\Gamma \vdash_{\cap G} e_1\ e_2 : T$ then by rule T-App, $\Gamma \vdash_{\cap G} e_1 : PM$, $PM \rhd T_1 \cap \ldots \cap T_n \to T$, $\Gamma \vdash_{\cap G} e_2 : T'_1 \cap \ldots \cap T'_n$ and $T'_1 \cap \ldots \cap T'_n \sim T_1 \cap \ldots \cap T_n$. By the induction hypothesis, $\Gamma \vdash_{\cap CC} e_1 \rightsquigarrow e'_1 : PM$ and $\Gamma \vdash_{\cap CC} e'_1 : PM$, and $\Gamma \vdash_{\cap CC} e_2 \rightsquigarrow e'_2 : T'_1 \cap \ldots \cap T'_n$ and $\Gamma \vdash_{\cap CC} e'_2 : T'_1 \cap \ldots \cap T'_n$. Therefore, by rule C-App, $\Gamma \vdash_{\cap CC} e_1\ e_2 \rightsquigarrow e''_1\ e''_2 : T$. By the definition of $\trianglelefteq$ and S, S, $e \hookrightarrow e$, by rule T-CastIntersection, $\Gamma \vdash_{\cap CC} e''_1 : T_1 \to T \cap \ldots \cap T_n \to T$ and $\Gamma \vdash_{\cap CC} e''_2 : T_1 \cap \ldots \cap T_n$. By rule T-App', $\Gamma \vdash_{\cap CC} e''_1\ e''_2 : T \cap \ldots \cap T$ and then by the properties of intersection types (modulo repetitions), $\Gamma \vdash_{\cap CC} e''_1\ e''_2 : T$.

- Rule T-Gen. If $\Gamma \vdash_{\cap G} e : T_1 \cap \ldots \cap T_n$ then by rule T-Gen, $\Gamma \vdash_{\cap G} e : T_1$ and $\ldots$ and $\Gamma \vdash_{\cap G} e : T_n$. By the induction hypothesis, $\Gamma \vdash_{\cap CC} e \rightsquigarrow e' : T_1$ and $\ldots$ and $\Gamma \vdash_{\cap CC} e \rightsquigarrow e' : T_n$, and $\Gamma \vdash_{\cap CC} e' : T_1$ and $\ldots$ and $\Gamma \vdash_{\cap CC} e' : T_n$. By rule C-Gen, $\Gamma \vdash_{\cap CC} e \rightsquigarrow e' : T_1 \cap \ldots \cap T_n$ and by rule T-Gen, $\Gamma \vdash_{\cap CC} e' : T_1 \cap \ldots \cap T_n$.
- Rule T-Inst. If $\Gamma \vdash_{\cap G} e : T_i$ then by rule T-Inst, $\Gamma \vdash_{\cap G} e : T_1 \cap \ldots \cap T_n$, such that $T_i \in \{T_1, \ldots, T_n\}$. By the induction hypothesis, $\Gamma \vdash_{\cap CC} e \rightsquigarrow e' : T_1 \cap \ldots \cap T_n$ and $\Gamma \vdash_{\cap CC} e' : T_1 \cap \ldots \cap T_n$. By rule C-Inst, $\Gamma \vdash_{\cap CC} e \rightsquigarrow e' : T_i$ and by rule T-Inst, $\Gamma \vdash_{\cap CC} e' : T_i$.

□

Theorem B.4 (Monotonicity w.r.t precision of cast insertion). *If* $\Gamma \vdash_{\cap CC} e_1 \rightsquigarrow e'_1 : T_1$ *and* $\Gamma \vdash_{\cap CC} e_2 \rightsquigarrow e'_2 : T_2$ *and* $e_1 \sqsubseteq e_2$ *then* $e'_1 \sqsubseteq e'_2$ *and* $T_1 \sqsubseteq T_2$.

Proof. We proceed by induction on the length of the derivation tree of $\Gamma \vdash_{\cap CC} e_1 \rightsquigarrow e'_1 : T$.

Base cases:

- Rule C-Var. If $\Gamma \vdash_{\cap CC} x^T \rightsquigarrow x^T : T$ and $\Gamma \vdash_{\cap CC} x^{T'} \rightsquigarrow x^{T'} : T'$, and $x^T \sqsubseteq x^{T'}$, then $x^T \sqsubseteq x^{T'}$ and $T \sqsubseteq T'$.
- Rule C-Int. If $\Gamma \vdash_{\cap CC} n \rightsquigarrow n : Int$, $\Gamma \vdash_{\cap CC} n \rightsquigarrow n : Int$ and $n \sqsubseteq n$, then $n \sqsubseteq n$ and $Int \sqsubseteq Int$.
- Rule C-True. If $\Gamma \vdash_{\cap CC} true \rightsquigarrow true : Bool$, $\Gamma \vdash_{\cap CC} true \rightsquigarrow true : Bool$ and $true \sqsubseteq true$, then $true \sqsubseteq true$ and $Bool \sqsubseteq Bool$.
- Rule C-False. If $\Gamma \vdash_{\cap CC} false \rightsquigarrow false : Bool$, $\Gamma \vdash_{\cap CC} false \rightsquigarrow false : Bool$ and $false \sqsubseteq false$, then $false \sqsubseteq false$ and $Bool \sqsubseteq Bool$.

Induction step:

- Rule C-Abs. If $\Gamma \vdash_{\cap CC} \lambda x : T_{11} \cap \ldots \cap T_{1n} \,.\, e_1 \rightsquigarrow \lambda x : T_{11} \cap \ldots \cap T_{1n} \,.\, e'_1 : T_{11} \cap \ldots \cap T_{1n} \to T_1$ and $\Gamma \vdash_{\cap CC} \lambda x : T_{21} \cap \ldots \cap T_{2n} \,.\, e_2 \rightsquigarrow \lambda x : T_{21} \cap \ldots \cap T_{2n} \,.\, e'_2 : T_{21} \cap \ldots \cap T_{2n} \to T_2$ and $\lambda x : T_{11} \cap \ldots \cap T_{1n} \,.\, e_1 \sqsubseteq \lambda x : T_{21} \cap \ldots \cap T_{2n} \,.\, e_2$ then by rule C-Abs, $\Gamma, x : T_{11} \cap \ldots \cap T_{1n} \vdash_{\cap CC} e_1 \rightsquigarrow e'_1 : T_1$ and $\Gamma, x : T_{21} \cap \ldots \cap T_{2n} \vdash_{\cap CC} e_2 \rightsquigarrow e'_2 : T_2$ and by the definition of $\sqsubseteq$, $T_{11} \cap \ldots \cap T_{1n} \sqsubseteq T_{21} \cap \ldots \cap T_{2n}$ and $e_1 \sqsubseteq e_2$. By the induction hypothesis, $e'_1 \sqsubseteq e'_2$ and $T_1 \sqsubseteq T_2$. Therefore, by the definition of $\sqsubseteq$, $\lambda x : T_{11} \cap \ldots \cap T_{1n} \,.\, e'_1 \sqsubseteq \lambda x : T_{21} \cap \ldots \cap T_{2n} \,.\, e'_2$ and $T_{11} \cap \ldots \cap T_{1n} \to T_1 \sqsubseteq T_{21} \cap \ldots \cap T_{2n} \to T_2$.
- Rule C-Abs'. If $\Gamma \vdash_{\cap CC} \lambda x : T_{11} \cap \ldots \cap T_{1n} \,.\, e_1 \rightsquigarrow \lambda x : T_{11} \cap \ldots \cap T_{1n} \,.\, e'_1 : T_{1i} \to T_1$, such that $T_{1i} \in \{T_{11}, \ldots, T_{1n}\}$, and $\Gamma \vdash_{\cap CC} \lambda x : T_{21} \cap \ldots \cap T_{2n} \,.\, e_2 \rightsquigarrow \lambda x : T_{21} \cap \ldots \cap T_{2n} \,.\, e'_2 : T_{2i} \to T_2$, such that $T_{2i} \in \{T_{21}, \ldots, T_{2n}\}$, and $\lambda x : T_{11} \cap \ldots \cap T_{1n} \,.\, e_1 \sqsubseteq \lambda x : T_{21} \cap \ldots \cap T_{2n} \,.\, e_2$ then by the definition of C-Abs', $\Gamma, x : T_{1i} \vdash_{\cap CC} e_1 \rightsquigarrow e'_1 : T_1$ and $\Gamma, x : T_{2i} \vdash_{\cap CC} e_2 \rightsquigarrow e'_2 : T_2$ and by the definition of $\sqsubseteq$, $T_{11} \cap \ldots \cap T_{1n} \sqsubseteq T_{21} \cap \ldots \cap T_{2n}$ and $e_1 \sqsubseteq e_2$ and therefore $T_{1i} \sqsubseteq T_{2i}$. By the induction hypothesis, $e'_1 \sqsubseteq e'_2$ and $T_1 \sqsubseteq T_2$. Therefore, by the definition of $\sqsubseteq$, $\lambda x : T_{11} \cap \ldots \cap T_{1n} \,.\, e'_1 \sqsubseteq \lambda x : T_{21} \cap \ldots \cap T_{2n} \,.\, e'_2$ and $T_{1i} \to T_1 \sqsubseteq T_{2i} \to T_2$.
- Rule C-App. If $\Gamma \vdash_{\cap CC} e_{11}\ e_{12} \rightsquigarrow e''_{11}\ e''_{12} : T_1$ and $\Gamma \vdash_{\cap CC} e_{21}\ e_{22} \rightsquigarrow e''_{21}\ e''_{22} : T_2$ and $e_{11}\ e_{12} \sqsubseteq e_{21}\ e_{22}$ then by rule C-App, $\Gamma \vdash_{\cap CC} e_{11} \rightsquigarrow e'_{11} : PM_1$ and $PM_1 \rhd T_{11} \cap \ldots \cap T_{1n} \to T_1$ and $\Gamma \vdash_{\cap CC} e_{12} \rightsquigarrow e'_{12} : T'_{11} \cap \ldots \cap T'_{1n}$ and $T'_{11} \cap \ldots \cap T'_{1n} \sim T_{11} \cap \ldots \cap T_{1n}$ and $PM_1 \trianglelefteq S_{11}$ and $T_{11} \cap \ldots \cap T_{1n} \to T_1 \trianglelefteq S_{12}$ and $T'_{11} \cap \ldots \cap T'_{1n} \trianglelefteq S_{13}$ and $T_{11} \cap \ldots \cap T_{1n} \trianglelefteq S_{14}$ and S_{11}, S_{12}, $e'_{11} \hookrightarrow e''_{11}$ and S_{13}, S_{14}, $e'_{12} \hookrightarrow e''_{12}$ and $\Gamma \vdash_{\cap CC} e_{21} \rightsquigarrow e'_{21} : PM_2$ and $PM_2 \rhd T_{21} \cap \ldots \cap T_{2n} \to T_2$ and $\Gamma \vdash_{\cap CC} e_{22} \rightsquigarrow e'_{22} : T'_{21} \cap \ldots \cap T'_{2n}$ and $T'_{21} \cap \ldots \cap T'_{2n} \sim T_{21} \cap \ldots \cap T_{2n}$ and $PM_2 \trianglelefteq S_{21}$ and $T_{21} \cap \ldots \cap T_{2n} \to T_2 \trianglelefteq S_{22}$ and $T'_{21} \cap \ldots \cap T'_{2n} \trianglelefteq S_{23}$ and $T_{21} \cap \ldots \cap T_{2n} \trianglelefteq S_{24}$ and S_{21}, S_{22}, $e'_{21} \hookrightarrow e''_{21}$ and S_{23}, S_{24}, $e'_{22} \hookrightarrow e''_{22}$. As, by the definition of $\sqsubseteq$,

$e_{11} \sqsubseteq e_{21}$ and $e_{12} \sqsubseteq e_{22}$ then by the induction hypothesis, $e'_{11} \sqsubseteq e'_{21}$ and $PM_1 \sqsubseteq PM_2$ and $e'_{12} \sqsubseteq e'_{22}$ and $T'_{11} \cap \ldots \cap T'_{1n} \sqsubseteq T'_{21} \cap \ldots \cap T'_{2n}$. By the definition of $\rhd$, we have that $PM_1 = T_{11} \cap \ldots \cap T_{1n} \to T_1$ and $PM_2 = T_{21} \cap \ldots \cap T_{2n} \to T_2$ and so $T_{11} \cap \ldots \cap T_{1n} \to T_1 \sqsubseteq T_{21} \cap \ldots \cap T_{2n} \to T_2$ and therefore by the definition of $\sqsubseteq$, $T_1 \sqsubseteq T_2$. As by the definition of $\unlhd$, S, S, $e \hookrightarrow e$ and $\sqsubseteq$, $e''_{11} \sqsubseteq e''_{21}$ and $e''_{12} \sqsubseteq e''_{22}$, then by the definition of $\sqsubseteq$, $e''_{11}\ e''_{12} \sqsubseteq e''_{21}\ e''_{22}$ and $T_1 \sqsubseteq T_2$.

- Rule C-Gen. If $\Gamma \vdash_{\cap CC} e_1 \rightsquigarrow e'_1 : T_{11} \cap \ldots \cap T_{1n}$ and $\Gamma \vdash_{\cap CC} e_2 \rightsquigarrow e'_2 : T_{21} \cap \ldots \cap T_{2n}$ and $e_1 \sqsubseteq e_2$ then by rule C-Gen, $\Gamma \vdash_{\cap CC} e_1 \rightsquigarrow e'_1 : T_{11}$ and ... and $\Gamma \vdash_{\cap CC} e_1 \rightsquigarrow e'_1 : T_{1n}$ and $\Gamma \vdash_{\cap CC} e_2 \rightsquigarrow e'_2 : T_{21}$ and ... and $\Gamma \vdash_{\cap CC} e_2 \rightsquigarrow e'_2 : T_{2n}$. By the induction hypothesis, $e'_1 \sqsubseteq e'_2$ and $T_{11} \sqsubseteq T_{21}$ and ... and $T_{1n} \sqsubseteq T_{2n}$, and therefore by the definition of $\sqsubseteq$, $T_{11} \cap \ldots \cap T_{1n} \sqsubseteq T_{21} \cap \ldots \cap T_{2n}$.

- Rule C-Inst. If $\Gamma \vdash_{\cap CC} e_1 \rightsquigarrow e'_1 : T_{1i}$ and $\Gamma \vdash_{\cap CC} e_2 \rightsquigarrow e'_2 : T_{2i}$ and $e_1 \sqsubseteq e_2$ then by rule C-Inst, $\Gamma \vdash_{\cap CC} e_1 \rightsquigarrow e'_1 : T_{11} \cap \ldots \cap T_{1n}$ and $\Gamma \vdash_{\cap CC} e_2 \rightsquigarrow e'_2 : T_{21} \cap \ldots \cap T_{2n}$. By the induction hypothesis, $e'_1 \sqsubseteq e'_2$ and $T_{11} \cap \ldots \cap T_{1n} \sqsubseteq T_{21} \cap \ldots \cap T_{2n}$, and therefore, by the definition of $\sqsubseteq$, $T_{1i} \sqsubseteq T_{2i}$.

□

Corollary B.4.1 (Monotonicity of cast insertion)**.** *Corollary of Theorem B.4. If* $\Gamma \vdash_{\cap CC} e_1 \rightsquigarrow e'_1 : T_1$ *and* $\Gamma \vdash_{\cap CC} e_2 \rightsquigarrow e'_2 : T_2$ *and* $e_1 \sqsubseteq e_2$ *then* $e'_1 \sqsubseteq e'_2$.

Theorem B.5 (Conservative Extension)**.** *If e is fully static, then* $e \longrightarrow_{\cap S} e' \iff e \longrightarrow_{\cap CC} e'$.

Proof. We proceed by induction on the length of the derivation tree of $\longrightarrow_{\cap S}$ and $\longrightarrow_{\cap CC}$ for the right and left direction of the implication, respectively.

Base cases:

- Rule E-AppAbs. If $(\lambda x : T_1 \cap \ldots \cap T_n \ .\ e)\ v \longrightarrow_{\cap S} [x \mapsto v]e$ and $(\lambda x : T_1 \cap \ldots \cap T_n \ .\ e)\ v \longrightarrow_{\cap CC} [x \mapsto v]e$, then it is proved.

Induction step:

- Rule E-App1.
  - If $e_1\ e_2 \longrightarrow_{\cap S} e'_1\ e_2$ then by rule E-App1, $e_1 \longrightarrow_{\cap S} e'_1$. By the induction hypothesis, $e_1 \longrightarrow_{\cap CC} e'_1$. Therefore, by rule E-App1, $e_1\ e_2 \longrightarrow_{\cap CC} e'_1\ e_2$
  - If $e_1\ e_2 \longrightarrow_{\cap CC} e'_1\ e_2$ then by rule E-App1, $e_1 \longrightarrow_{\cap CC} e'_1$. By the induction hypothesis, $e_1 \longrightarrow_{\cap S} e'_1$. Therefore, by rule E-App1, $e_1\ e_2 \longrightarrow_{\cap S} e'_1\ e_2$
- Rule E-App2.
  - If $v_1\ e_2 \longrightarrow_{\cap S} v_1\ e'_2$ then by rule E-App2, $e_2 \longrightarrow_{\cap S} e'_2$. By the induction hypothesis, $e_2 \longrightarrow_{\cap CC} e'_2$. Therefore, by rule E-App2, $v_1\ e_2 \longrightarrow_{\cap CC} v_1\ e'_2$
  - If $v_1\ e_2 \longrightarrow_{\cap CC} v_1\ e'_2$ then by rule E-App2, $e_2 \longrightarrow_{\cap CC} e'_2$. By the induction hypothesis, $e_2 \longrightarrow_{\cap S} e'_2$. Therefore, by rule E-App2, $v_1\ e_2 \longrightarrow_{\cap S} v_1\ e'_2$

□

Theorem B.6 (Type preservation)**.** *Depends on Lemmas B.2 and B.3. If* $\Gamma \vdash_{\cap CC} e : T_1 \cap \ldots \cap T_n$ *and* $e \longrightarrow_{\cap CC} e'$ *then* $\Gamma \vdash_{\cap CC} e' : T_1 \cap \ldots \cap T_m$ *such that* $m \leq n$.

Proof. We proceed by induction on the length of the derivation tree of $\longrightarrow_{\cap CC}$.

Base cases:

- Rule E-PushBlame1. If $\Gamma \vdash_{\cap CC} blame_{T_2}\ l\ e_2 : T_1$ and $blame_{T_2}\ l\ e_2 \longrightarrow_{\cap CC} blame_{T_1}\ l$ then by rule T-Blame, $\Gamma \vdash_{\cap CC} blame_{T_1}\ l : T_1$.

- Rule E-PushBlame2. If $\Gamma \vdash_{\cap CC} e_1\ blame_{T_2}\ l : T_1$ and $e_1\ blame_{T_2}\ l \longrightarrow_{\cap CC} blame_{T_1}\ l$ then by rule T-Blame, $\Gamma \vdash_{\cap CC} blame_{T_1}\ l : T_1$.

- Rule E-PushBlameCast. If $\Gamma \vdash_{\cap CC} blame_T\ l : c_1 \cap \ldots \cap c_n : T_1 \cap \ldots \cap T_n$ and $blame_T\ l : c_1 \cap \ldots \cap c_n \longrightarrow_{\cap CC} blame_{T_1 \cap \ldots \cap T_n}\ l$ then by rule T-Blame, $\Gamma \vdash_{\cap CC} blame_{T_1 \cap \ldots \cap T_n}\ l : T_1 \cap \ldots \cap T_n$.

- Rule E-AppAbs. If $\Gamma \vdash_{\cap CC} (\lambda x : T_1 \cap \ldots \cap T_n\ .\ e)\ v : T$ then either by rule T-App, $\Gamma \vdash_{\cap CC} \lambda x : T_1 \cap \ldots \cap T_n\ .\ e : T_1 \cap \ldots \cap T_n \to T$ or by rule T-App', $\Gamma \vdash_{\cap CC} \lambda x : T_1 \cap \ldots \cap T_n\ .\ e : T_1 \to T \cap \ldots \cap T_n \to T$ and $\Gamma \vdash_{\cap CC} v : T_1 \cap \ldots \cap T_n$ (x does not occur in Γ). Moreover, by rule T-Abs, $\Gamma, x : T_1 \cap \ldots \cap T_n \vdash_{\cap CC} e : T$. By rule E-AppAbs, $(\lambda x : T_1 \cap \ldots \cap T_n\ .\ e)\ v \longrightarrow_{\cap CC} [x \mapsto v]e$. To obtain $\Gamma \vdash_{\cap CC} [x \mapsto v]e : T$, it is sufficient to replace, in the proof of $\Gamma, x : T_1 \cap \ldots \cap T_n \vdash_{\cap CC} e : T$, the statements $x : T_i$ (introduzed by the rules T-Var and T-Inst) by the deductions of $\Gamma \vdash_{\cap CC} v : T_i$ for $1 \leq i \leq n$. (Proof adapted from [5])

- Rule E-SimulateArrow. If $\Gamma \vdash_{\cap CC} (v_1 : cv_1 \cap \ldots \cap cv_n)\ v_2 : T_{12} \cap \ldots \cap T_{n2}$, then by rule T-App', $\Gamma \vdash_{\cap CC} v_1 : cv_1 \cap \ldots \cap cv_n : T_1 \cap \ldots \cap T_n$ such that $\exists i \in 1..n\ .\ T_i = T_{i1} \to T_{i2}$ and $\Gamma \vdash_{\cap CC} v_2 : T_{11} \cap \ldots \cap T_{n1}$. As $\Gamma \vdash_{\cap CC} v_1 : cv_1 \cap \ldots \cap cv_n : T_1 \cap \ldots \cap T_n$, then by rule T-CastIntersection, $\Gamma \vdash_{\cap CC} v_1 : T''_1 \cap \ldots \cap T''_l$ and $\vdash_{\cap CI} cv_1 : (I_1, T_1)$ and $\ldots$ and $\vdash_{\cap CI} cv_n : (I_n, T_n)$ such that $\{I_1, \ldots, I_n\} \subseteq \{T''_1, \ldots, T''_l\}$ and $I_1 \cap \ldots \cap I_n = T''_1 \cap \ldots \cap T''_n$ and $n \leq l$. For the sake of simplicity lets elide cast labels and blame labels. By the definition of SimulateArrow, we have that $c'_1 = c''_1 : T'_{11} \to T'_{12} \Rightarrow T_{11} \to T_{12}$ and $\ldots$ and $c'_m = c''_m : T'_{m1} \to T'_{m2} \Rightarrow T_{m1} \to T_{m2}$, for some $m \leq n$. Also, $c_{11} = \varnothing\ T_{11}\ : T_{11} \Rightarrow T'_{11}$ and $\ldots$ and $c_{m1} = \varnothing\ T_{m1}\ : T_{m1} \Rightarrow T'_{m1}$ and $c_{12} : \varnothing\ T'_{12}\ : T'_{12} \Rightarrow T_{12}$ and $\ldots$ and $c_{m2} = \varnothing\ T'_{m2}\ : T'_{m2} \Rightarrow T_{m2}$ and $\vdash_{\cap CI} c^s_1 : (I_1, T'_{11} \to T'_{12})$ and $\ldots$ and $\vdash_{\cap CI} c^s_m : (I_m, T'_{m1} \to T'_{m2})$. As by rule T-Gen and T-Inst $\Gamma \vdash_{\cap CC} v_1 : T''_1 \cap \ldots \cap T''_m$ and $I_1 \cap \ldots \cap I_m = T''_1 \cap \ldots \cap T''_m$, then by rule T-CastIntersection, $\Gamma \vdash_{\cap CC} v_1 : c^s_1 \cap \ldots \cap c^s_m : T'_{11} \to T'_{12} \cap \ldots \cap T'_{m1} \to T'_{m2}$. As by rule T-Gen and T-Inst $\Gamma \vdash_{\cap CC} v_2 : T_{11} \cap \ldots \cap T_{m1}$ and $\vdash_{\cap CI} c_{11} : (T_{11}, T'_{11})$ and $\ldots$ and $\vdash_{\cap CI} c_{m1} : (T_{m1}, T'_{m1})$ then by rule T-CastIntersection, $\Gamma \vdash_{\cap CC} v_2 : c_{11} \cap \ldots \cap c_{m1} : T'_{11} \cap \ldots \cap T'_{m1}$. Therefore, by rule T-App', $\Gamma \vdash_{\cap CC} (v_1 : c^s_1 \cap \ldots \cap c^s_m)\ (v_2 : c_{11} \cap \ldots \cap c_{m1}) : T'_{12} \cap \ldots \cap T'_{m2}$. As $\vdash_{\cap CI} c_{12} : (T'_{12}, T_{12})$ and $\ldots$ and $\vdash_{\cap CI} c_{m2} : (T'_{m2}, T_{m2})$, then by rule T-CastIntersection, $\Gamma \vdash_{\cap CC} (v_1 : c^s_1 \cap \ldots \cap c^s_m)\ (v_2 : c_{11} \cap \ldots \cap c_{m1}) : c_{12} \cap \ldots \cap c_{m2} : T_{12} \cap \ldots \cap T_{m2}$. By rule E-SimulateArrow, $(v_1 : cv_1 \cap \ldots \cap cv_n)\ v_2 \longrightarrow_{\cap CC} (v_1 : c^s_1 \cap \ldots \cap c^s_m)\ (v_2 : c_{11} \cap \ldots \cap c_{m1}) : c_{12} \cap \ldots \cap c_{m2}$, therefore it is proved.

- Rule E-MergeCasts. If $\Gamma \vdash_{\cap CC} v : cv_1 \cap \ldots \cap cv_n : c'_1 \cap \ldots \cap c'_m : F'_1 \cap \ldots \cap F'_m$ then by rule T-CastIntersections, $\Gamma \vdash_{\cap CC} v : cv_1 \cap \ldots \cap cv_n : F_1 \cap \ldots \cap F_n$ and $\vdash_{\cap CI} c'_1 : (I'_1, F'_1)$ and $\ldots$ and $\vdash_{\cap CI} c'_m : (I'_m, F'_m)$ such that $\{I'_1, \ldots, I'_m\} \subseteq \{F_1, \ldots, F_n\}$ and $I'_1 \cap \ldots \cap I'_m = F_1 \cap \ldots \cap F_m$ and $m \leq n$. As $\Gamma \vdash_{\cap CC} v : cv_1 \cap \ldots \cap cv_n : F_1 \cap \ldots \cap F_n$ then by rule T-CastIntersection, $\Gamma \vdash_{\cap CC} v : T_1 \cap \ldots \cap T_l$ and $\vdash_{\cap CI} cv_1 : (I_1, F_1)$ and $\ldots$ and $\vdash_{\cap CI} cv_n : (I_n, F_n)$ such that $\{I_1, \ldots, I_n\} \subseteq \{T_1, \ldots, T_l\}$ and $I_1 \cap \ldots \cap I_n = T_1 \cap \ldots \cap T_n$ and $n \leq l$. By the definition of mergeCasts, $\vdash_{\cap CI} c''_1 : (I''_1, F''_1)$ and $\ldots$ and $\vdash_{\cap CI} c''_j : (I''_j, F''_j)$ and such that $\{I''_1, \ldots, I''_j\} \subseteq \{T_1, \ldots, T_l\}$ and $I''_1 \cap \ldots \cap I''_j = T_1 \cap \ldots \cap T_j$ and $\{F''_1, \ldots, F''_j\} \subseteq \{F'_1, \ldots, F'_m\}$ and $F''_1 \cap \ldots \cap F''_j = F'_1 \cap \ldots \cap F'_j$ and $j \leq l$ and $j \leq m$. By rule T-Gen and T-Inst, $\Gamma \vdash_{\cap CC} v : T_1 \cap \ldots \cap T_j$ and therefore by rule T-CastIntersection, $\Gamma \vdash_{\cap CC} v : c''_1 \cap \ldots \cap c''_j : F''_1 \cap \ldots \cap F''_j$. By rule E-MergeCasts, $v : cv_1 \cap \ldots \cap cv_n : c'_1 \cap \ldots \cap c'_m \longrightarrow_{\cap CC} v : c''_1 \cap \ldots \cap c''_j$.

- Rule E-EvaluateCasts. If $\Gamma \vdash_{\cap CC} v : c_1 \cap \ldots \cap c_n : T_1 \cap \ldots \cap T_n$ then by rule T-CastIntersection, $\Gamma \vdash_{\cap CC} v : T'_1 \cap \ldots \cap T'_n$ and $\vdash_{\cap CI} c_1 : (I_1, T_1)$ and $\ldots$ and $\vdash_{\cap CI} c_n : (I_n, T_n)$ and $I_1 \cap \ldots \cap I_n = T'_1 \cap \ldots \cap T'_n$. By rule E-EvaluateCasts, $c_1 \longrightarrow_{\cap CI} cv_1$ and $\ldots$ and $c_n \longrightarrow_{\cap CI} cv_n$. By Lemmas B.2 and

B.3, $\vdash_{\cap CI} cv_1 : (I_1, T_1)$ and $\ldots$ and $\vdash_{\cap CI} cv_n : (I_n, T_n)$. Therefore by rule T-CastIntersection, $\Gamma \vdash_{\cap CC} v : cv_1 \cap \ldots \cap cv_n : T_1 \cap \ldots \cap T_n$. By rule E-EvaluateCasts, $v : c_1 \cap \ldots \cap c_n \longrightarrow_{\cap CC} v : cv_1 \cap \ldots \cap cv_n$.

- Rule E-PropagateBlame. If $\Gamma \vdash_{\cap CC} v : \mathit{blame}\ T_1'\ T_1\ l_1\ ^{m_1} \cap \ldots \cap \mathit{blame}\ T_n'\ T_n\ l_n\ ^{m_n} : T_1 \cap \ldots \cap T_n$ and by rule E-PropagateBlame $v : \mathit{blame}\ T_1'\ T_1\ l_1\ ^{m_1} \cap \ldots \cap \mathit{blame}\ T_n'\ T_n\ l_n\ ^{m_n} \longrightarrow_{\cap CC} \mathit{blame}_{(T_1 \cap \ldots \cap T_n)}\ l_1$, then by rule T-Blame, $\Gamma \vdash_{\cap CC} \mathit{blame}_{(T_1 \cap \ldots \cap T_n)}\ l_1 : T_1 \cap \ldots \cap T_n$.
- Rule E-RemoveEmpty. If $\Gamma \vdash_{\cap CC} v : \varnothing\ T_1\ ^{m_1} \cap \ldots \cap \varnothing\ T_n\ ^{m_n} : T_1 \cap \ldots \cap T_n$, then by rule T-CastIntersection, $\Gamma \vdash_{\cap CC} v : T_1 \cap \ldots \cap T_n$ and $\vdash_{\cap CI} \varnothing\ T_1\ ^{m_1} : (T_1, T_1)$ and $\ldots$ and $\vdash_{\cap CI} \varnothing\ T_n\ ^{m_n} : (T_n, T_n)$. Therefore, by rule E-RemoveEmpty, $v : \varnothing\ T_1\ ^{m_1} \cap \ldots \cap \varnothing\ T_n\ ^{m_n} \longrightarrow_{\cap CC} v$.

Induction step:

- Rule E-App1. There are two possibilities:
  - If $\Gamma \vdash_{\cap CC} e_1\ e_2 : T$, then by rule T-App, $\Gamma \vdash_{\cap CC} e_1 : T_1 \cap \ldots \cap T_n \to T$ and $\Gamma \vdash_{\cap CC} e_2 : T_1 \cap \ldots \cap T_n$. By rule E-App1, $e_1 \longrightarrow_{\cap CI} e_1'$, so by the induction hypothesis, $\Gamma \vdash_{\cap CC} e_1' : T_1 \cap \ldots \cap T_n \to T$. As by rule E-App1, $e_1\ e_2 \longrightarrow_{\cap CI} e_1'\ e_2$, then by rule T-App, $\Gamma \vdash_{\cap CC} e_1'\ e_2 : T$.
  - If $\Gamma \vdash_{\cap CC} e_1\ e_2 : T_{12} \cap \ldots \cap T_{n2}$, then by rule T-App', $\Gamma \vdash_{\cap CC} e_1 : T_{11} \to T_{12} \cap \ldots \cap T_{n1} \to T_{n2}$ and $\Gamma \vdash_{\cap CC} e_2 : T_{11} \cap \ldots \cap T_{n1}$. By rule E-App1, $e_1 \longrightarrow_{\cap CI} e_1'$, so by the induction hypothesis, $\Gamma \vdash_{\cap CC} e_1' : T_{11} \to T_{12} \cap \ldots \cap T_{n1} \to T_{n2}$. As by rule E-App1, $e_1\ e_2 \longrightarrow_{\cap CI} e_1'\ e_2$, then by rule T-App', $\Gamma \vdash_{\cap CC} e_1'\ e_2 : T_{12} \cap \cdots \cap T_{n2}$.
- Rule E-App2. There are two possibilities:
  - If $\Gamma \vdash_{\cap CC} v_1\ e_2 : T$, then by rule T-App, $\Gamma \vdash_{\cap CC} v_1 : T_1 \cap \ldots \cap T_n \to T$ and $\Gamma \vdash_{\cap CC} e_2 : T_1 \cap \ldots \cap T_n$. By rule E-App2, $e_2 \longrightarrow_{\cap CI} e_2'$, so by the induction hypothesis, $\Gamma \vdash_{\cap CC} e_2' : T_1 \cap \ldots \cap T_n$. As by rule E-App2, $v_1\ e_2 \longrightarrow_{\cap CI} v_1\ e_2'$, then by rule T-App, $\Gamma \vdash_{\cap CC} v_1\ e_2' : T$.
  - If $\Gamma \vdash_{\cap CC} v_1\ e_2 : T_{12} \cap \ldots \cap T_{n2}$, then by rule T-App', $\Gamma \vdash_{\cap CC} v_1 : T_{11} \to T_{12} \cap \ldots \cap T_{n1} \to T_{n2}$ and $\Gamma \vdash_{\cap CC} e_2 : T_{11} \cap \ldots \cap T_{n1}$. By rule E-App2, $e_2 \longrightarrow_{\cap CI} e_2'$, so by the induction hypothesis, $\Gamma \vdash_{\cap CC} e_2' : T_{11} \cap \ldots \cap T_{n1}$. As by rule E-App1, $v_1\ e_2 \longrightarrow_{\cap CI} v_1\ e_2'$, then by rule T-App', $\Gamma \vdash_{\cap CC} v_1\ e_2' : T_{12} \cap \cdots \cap T_{n2}$..
- Rule E-Evaluate. If $\Gamma \vdash_{\cap CC} e : c_1 \cap \ldots \cap c_n : T_1 \cap \ldots \cap T_n$, then by rule T-CastIntersection, $\Gamma \vdash_{\cap CC} e : T_1' \cap \ldots \cap T_n'$, $\vdash_{\cap CI} c_1 : T_1$ and $\ldots$ and $\vdash_{\cap CI} c_n : T_n$ and $\mathit{initialType}(c_1) \cap \ldots \cap \mathit{initialType}(c_n) = T_1' \cap \ldots \cap T_n'$. By rule E-Evaluate, $e \longrightarrow_{\cap CI} e'$, so by the induction hypothesis, $\Gamma \vdash_{\cap CC} e' : T$. As by rule E-Evaluate, $e : c_1 \cap \ldots \cap c_n \longrightarrow_{\cap CI} e' : c_1 \cap \ldots \cap c_n$, then by rule T-CastIntersection, $\Gamma \vdash_{\cap CC} e' : c_1 \cap \ldots \cap c_n : T_1 \cap \ldots \cap T_n$.

$\square$

Theorem B.7 (Progress)**.** *If $\Gamma \vdash_{\cap CC} e : T$ then either e is a value or there exists an e' such that $e \longrightarrow_{\cap CC} e'$.*

Proof. We proceed by induction on the length of the derivation tree of $\Gamma \vdash_{\cap CC} e : T$.

Base cases:

- Rule T-Var. If $\Gamma \vdash_{\cap CC} x^T : T$, then x^T is a value.
- Rule T-Int. If $\Gamma \vdash_{\cap CC} n : \mathit{Int}$ then n is a value.
- Rule T-True. If $\Gamma \vdash_{\cap CC} \mathit{true} : \mathit{Bool}$ then true is a value.
- Rule T-False. If $\Gamma \vdash_{\cap CC} \mathit{false} : \mathit{Bool}$ then false is a value.

Induction step:

- Rule T-Abs. If $\Gamma \vdash_{\cap CC} \lambda x : T_1 \cap \ldots \cap T_n \,.\, e : T_1 \cap \ldots \cap T_n \rightarrow T$ then $\lambda x : T_1 \cap \ldots \cap T_n \,.\, e$ is a value.
- Rule T-Abs'. If $\Gamma \vdash_{\cap CC} \lambda x : T_1 \cap \ldots \cap T_n \,.\, e : T_i \rightarrow T$ then $\lambda x : T_1 \cap \ldots \cap T_n \,.\, e$ is a value.
- Rule T-App. If $\Gamma \vdash_{\cap CC} e_1\ e_2 : T$ then by rule T-App, $\Gamma \vdash_{\cap CC} e_1 : T_1 \cap \ldots \cap T_n \rightarrow T$ and $\Gamma \vdash_{\cap CC} e_2 : T_1 \cap \ldots \cap T_n$. By the induction hypothesis, e_1 is either a value or there is a e_1' such that $e_1 \longrightarrow_{\cap CC} e_1'$ and e_2 is either a value or there is a e_2' such that $e_2 \longrightarrow_{\cap CC} e_2'$. If e_1 is a value, then by rule E-PushBlame1, $(\mathit{blame}_{T_2}\ l)\ e_2 \longrightarrow_{\cap CC} \mathit{blame}_{T_1}\ l$. If e_2 is a value, then by rule E-PushBlame2, $e_1\ (\mathit{blame}_{T_2}\ l) \longrightarrow_{\cap CC} \mathit{blame}_{T_1}\ l$. If e_1 is not a value, then by rule E-App1, $e_1\ e_2 \longrightarrow_{\cap CC} e_1'\ e_2$. If e_1 is a value and e_2 is not a value, then by rule E-App2, $v_1\ e_2 \longrightarrow_{\cap CC} v_1\ e_2'$. If both e_1 and e_2 are values then e_1 must be a λ-abstraction $(\lambda x : T_1 \cap \ldots \cap T_n \,.\, e)$, and by rule E-AppAbs $(\lambda x : T_1 \cap \ldots \cap T_n \,.\, e)\ v_2 \longrightarrow_{\cap CC} [x \mapsto v_2]e$.
- Rule T-Gen. If $\Gamma \vdash_{\cap CC} e : T_1 \cap \ldots \cap T_n$ then by rule T-Gen, $\Gamma \vdash_{\cap CC} e : T_1$ and ... and $\Gamma \vdash_{\cap CC} e : T_n$. By the induction hypothesis, either e is a value or there exists an e' such that $e \longrightarrow_{\cap CC} e'$.
- Rule T-Inst. If $\Gamma \vdash_{\cap CC} e : T_i$ then by rule T-Inst, $\Gamma \vdash_{\cap CC} e : T_1 \cap \ldots \cap T_n$, such that $T_i \in \{T_1, \ldots, T_n\}$. By the induction hypothesis, either e is a value or there exists an e' such that $e \longrightarrow_{\cap CC} e'$.
- Rule T-App'. If $\Gamma \vdash_{\cap CC} e_1\ e_2 : T_{12} \cap \ldots \cap T_{n2}$ then by rule T-App', $\Gamma \vdash_{\cap CC} e_1 : T_{11} \rightarrow T_{12} \cap \ldots \cap T_{n1} \rightarrow T_{n2}$ and $\Gamma \vdash_{\cap CC} e_2 : T_{11} \cap \ldots \cap T_{n1}$. By the induction hypothesis, e_1 is either a value or there is a e_1' such that $e_1 \longrightarrow_{\cap CC} e_1'$ and e_2 is either a value or there is a e_2' such that $e_2 \longrightarrow_{\cap CC} e_2'$. If e_1 is a value, then by rule E-PushBlame1, $(\mathit{blame}_{T_2}\ l)\ e_2 \longrightarrow_{\cap CC} \mathit{blame}_{T_1}\ l$. If e_2 is a value, then by rule E-PushBlame2, $e_1\ (\mathit{blame}_{T_2}\ l) \longrightarrow_{\cap CC} \mathit{blame}_{T_1}\ l$. If e_1 is not a value, then by rule E-App1, $e_1\ e_2 \longrightarrow_{\cap CC} e_1'\ e_2$. If e_1 is a value and e_2 is not a value, then by rule E-App2, $v_1\ e_2 \longrightarrow_{\cap CC} v_1\ e_2'$. If both e_1 and e_2 are values then e_1 must be a λ-abstraction $(\lambda x : T_{11} \rightarrow T_{12} \cap \ldots \cap T_{n1} \rightarrow T_{n2}.\ e)$, and by rule E-AppAbs $(\lambda x : T_{11} \rightarrow T_{12} \cap \ldots \cap T_{n1} \rightarrow T_{n2} \,.\, e)\ v_2 \longrightarrow_{\cap CC} [x \mapsto v_2]e$.
- Rule T-CastIntersection. If $\Gamma \vdash_{\cap CC} e : c_1 \cap \ldots \cap c_n : T_1 \cap \ldots \cap T_n$ then by rule T-CastIntersection, $\Gamma \vdash_{\cap CC} e : T_1' \cap \ldots \cap T_n'$. By the induction hypothesis, e is either a value, or there is an e' such that $e \longrightarrow_{\cap CC} e'$. If e is a value, then either by rule E-EvaluateCasts, $v : c_1 \cap \ldots \cap c_n \longrightarrow_{\cap CC} v : cv_1 \cap \ldots \cap cv_n$, or by rule E-PushBlameCast, $\mathit{blame}_{T_1' \cap \ldots \cap T_n'}\ l : c_1 \cap \ldots \cap c_n \longrightarrow_{\cap CC} \mathit{blame}_{T_1 \cap \ldots \cap T_n}\ l$. If there is an e' such that $e \longrightarrow_{\cap CC} e'$, then by rule E-Evaluate, $e : c_1 \cap \ldots \cap c_n \longrightarrow_{\cap CC} e' : c_1 \cap \ldots \cap c_n$.
- Rule T-Blame. If $\Gamma \vdash_{\cap CC} \mathit{blame}_T\ l : T$ then $\mathit{blame}_T\ l$ is a value.

$\square$

Theorem B.8 (Instances of Intersection Types)**.** *We define the set S of* instances *of an intersection type T as the set obtained by $T \trianglelefteq S$. Given a type T, if $T \trianglelefteq S$ then every element of S is a simple type.*

Proof. We proceed by structural induction on T.

Base cases:

- $T = Int$. If $Int \trianglelefteq \{Int\}$ then Int is the only instance of Int and Int is a simple type.
- $T = Bool$. If $Bool \trianglelefteq \{Bool\}$ then $Bool$ is the only instance of $Bool$ and $Bool$ is a simple type.
- $T = Dyn$. If $Dyn \trianglelefteq \{Dyn\}$ then Dyn is the only instance of Dyn and Dyn is a simple type.

Induction step:

- $T = T_1 \to T_2$. If $T_1 \to T_2 \trianglelefteq \{T_{11} \to T_2, \ldots, T_{1n} \to T_2\}$ then by the definition of $\trianglelefteq$, $T_1 \trianglelefteq \{T_{11}, \ldots, T_{1n}\}$. By the induction hypothesis, $\{T_{11}, \ldots, T_{1n}\}$ is the set of all the instances of T_1 and T_{11} and ... and T_{1n} are all simple types. As T_2 is a simple type, then T_2 is the only instance of T_2. Therefore, $\{T_{11} \to T_2, \ldots, T_{1n} \to T_2\}$ is the set of all the instances of $T_1 \to T_2$ and $T_{11} \to T_2$ and ... and $T_{1n} \to T_2$ are all simple types.
- $T = T_1 \cap \ldots \cap T_n$. If $T_1 \cap \ldots \cap T_n \trianglelefteq \{T_{11}, \ldots, T_{1m}, \ldots, T_{n1}, \ldots, T_{nj}\}$ then by the definition of $\trianglelefteq$, $T_1 \trianglelefteq \{T_{11}, \ldots, T_{1m}\}$ and ... and $T_n \trianglelefteq \{T_{n1}, \ldots, T_{nj}\}$. By the induction hypothesis, $\{T_{11}, \ldots, T_{1m}\}$ is the set of all the instances of T_1 and T_{11} and ... and T_{1m} are all simple types and ... and $\{T_{n1}, \ldots, T_{nj}\}$ is the set of all the instances of T_n and T_{n1} and ... and T_{nj} are all simple types. Then, $\{T_{11}, \ldots, T_{1m}, \ldots, T_{n1}, \ldots, T_{nj}\}$ is the set of all the instance of $T_1 \cap \ldots \cap T_n$ and T_{11} and ... and T_{1m} and ... and T_{n1} and ... and T_{nj} are all simple types.

□

Conservative Extension to the GTLC: If e is annotated with only simple types and T is a simple type:

1. then $\Gamma \vdash_G e : T \iff \Gamma \vdash_{\cap G} e : T$.
2. then $\Gamma \vdash_{CC} e \rightsquigarrow e_1 : T \iff \Gamma \vdash_{\cap CC} e \rightsquigarrow e_2 : T$ and $e_1 =_c e_2$.
3. if $\Gamma \vdash_{CC} e_1 : T$, $\Gamma \vdash_{\cap CC} e_2 : T$ and $e_1 =_c e_2$ then $e_1 \longrightarrow_{CC} e'_1 \iff e_2 \longrightarrow_{\cap CC} e'_2$, and $e'_1 =_c e'_2$.

Theorem B.9 (Conservative Extension to the GTLC)**.** *If e is annotated with only simple types and T is a simple type, then $\Gamma \vdash_G e : T \iff \Gamma \vdash_{\cap G} e : T$.*

Proof. We will first prove the right direction of the implication, that if $\Gamma \vdash_G e : T$ then $\Gamma \vdash_{\cap G} e : T$. We proceed by induction on the length of the derivation tree of $\vdash_G$.

Base cases:

- Rule T-Var. If $\Gamma \vdash_G x : T$, then by rule T-Var, $x : T \in \Gamma$. As we are dealing with only simple types $T \subseteq T$ and therefore, $\Gamma \vdash_{\cap G} x^T : T$.
- Rule T-Int. If $\Gamma \vdash_G n : Int$, then by rule T-Int, $\Gamma \vdash_{\cap G} n : Int$.
- Rule T-True. If $\Gamma \vdash_G true : Bool$, then by rule T-True, $\Gamma \vdash_{\cap G} true : Bool$.
- Rule T-False. If $\Gamma \vdash_G false : Bool$, then by rule T-False, $\Gamma \vdash_{\cap G} false : Bool$.

Induction step:

- Rule T-Abs. If $\Gamma \vdash_G \lambda x : T_1 \,.\, e : T_1 \to T_2$, then by rule T-Abs, $\Gamma, x : T_1 \vdash_G e : T_2$. By the induction hypothesis, $\Gamma, x : T_1 \vdash_{\cap G} e : T_2$. Therefore, by rule T-Abs, $\Gamma \vdash_{\cap G} \lambda x : T_1 \,.\, e : T_1 \to T_2$.

- Rule T-App. If $\Gamma \vdash_G e_1\ e_2 : T_2$ then by rule T-App, $\Gamma \vdash_G e_1 : PM$, $PM \rhd T_1 \to T_2$, $\Gamma \vdash_G e_2 : T_1'$ and $T_1' \sim T_1$. By the induction hypothesis, $\Gamma \vdash_{\cap G} e_1 : PM$ and $\Gamma \vdash_{\cap G} e_2 : T_1'$. Therefore, by rule T-App, $\Gamma \vdash_{\cap G} e_1\ e_2 : T_2$.

We will now prove the left direction of the implication, that if $\Gamma \vdash_{\cap G} e : T$ then $\Gamma \vdash_G e : T$. We proceed by induction on the length of the derivation tree of $\vdash_{\cap G}$.

Base cases:

- Rule T-Var. If $\Gamma \vdash_{\cap G} x^T : T$, then by rule T-Var, $x : T \in \Gamma$ and $T \sqsubseteq T$. Therefore, $\Gamma \vdash_G x : T$.
- Rule T-Int. If $\Gamma \vdash_{\cap G} n : Int$, then by rule T-Int, $\Gamma \vdash_G n : Int$.
- Rule T-True. If $\Gamma \vdash_{\cap G} true : Bool$, then by rule T-True, $\Gamma \vdash_G true : Bool$.
- Rule T-False. If $\Gamma \vdash_{\cap G} false : Bool$, then by rule T-False, $\Gamma \vdash_G false : Bool$.

Induction step:

- Rule T-Abs. If $\Gamma \vdash_{\cap G} \lambda x : T_1 : e : T_1 \to T_2$, then by rule T-Abs, $\Gamma, x : T_1 \vdash_{\cap G} e : T_2$. By the induction hypothesis, $\Gamma, x : T_1 \vdash_G e : T_2$. Therefore, by rule T-Abs, $\Gamma \vdash_G \lambda x : T_1\ .\ e : T_1 \to T_2$.
- Rule T-Abs'. If $\Gamma \vdash_{\cap G} \lambda x : T_1 : e : T_1 \to T_2$, then by rule T-Abs', $\Gamma, x : T_1 \vdash_{\cap G} e : T_2$. By the induction hypothesis, $\Gamma, x : T_1 \vdash_G e : T_2$. Therefore, by rule T-Abs, $\Gamma \vdash_G \lambda x : T_1\ .\ e : T_1 \to T_2$.
- Rule T-App. If $\Gamma \vdash_{\cap G} e_1\ e_2 : T_2$ then by rule T-App, $\Gamma \vdash_{\cap G} e_1 : PM$, $PM \rhd T_1 \to T_2$, $\Gamma \vdash_{\cap G} e_2 : T_1'$ and $T_1' \sim T_1$. By the induction hypothesis, $\Gamma \vdash_G e_1 : PM$ and $\Gamma \vdash_G e_2 : T_1'$. Therefore, by rule T-App, $\Gamma \vdash_G e_1\ e_2 : T_2$.
- Rule T-Gen. If $\Gamma \vdash_{\cap G} e : T$, then by rule T-Gen, $\Gamma \vdash_{\cap G} e : T$. By the induction hypothesis, $\Gamma \vdash_G e : T$.
- Rule T-Inst. If $\Gamma \vdash_{\cap G} e : T$, then by rule T-Inst, $\Gamma \vdash_{\cap G} e : T$. By the induction hypothesis, $\Gamma \vdash_G e : T$.

□

Theorem B.10 (Conservative Extension to the GTLC). *If e is annotated with only simple types and T is a simple type then* $\Gamma \vdash_{CC} e \rightsquigarrow e_1 : T \iff \Gamma \vdash_{\cap CC} e \rightsquigarrow e_2 : T$ *and* $e_1 =_c e_2$.

Proof. We will first prove the right direction of the implication, that if $\Gamma \vdash_{CC} e \rightsquigarrow e_1 : T$ then $\Gamma \vdash_{\cap CC} e \rightsquigarrow e_2 : T$ and $e_1 =_c e_2$. We proceed by induction on the length of the derivation tree of $\Gamma \vdash_{CC} e \rightsquigarrow e_1 : T$.

Base cases:

- Rule C-Var. If $\Gamma \vdash_{CC} x \rightsquigarrow x : T$, then by rule C-Var, $x : T \in \Gamma$. As we are dealing with only simple types, $T \sqsubseteq T$, and therefore, by rule C-Var, $\Gamma \vdash_{\cap CC} x^T \rightsquigarrow x^T : T$.
- Rule C-Int. If $\Gamma \vdash_{CC} n \rightsquigarrow n : Int$, then by rule C-Int, $\Gamma \vdash_{\cap CC} n \rightsquigarrow n : Int$.
- Rule C-True. If $\Gamma \vdash_{CC} true \rightsquigarrow true : Bool$, then by rule C-True, $\Gamma \vdash_{\cap CC} true \rightsquigarrow true : Bool$.
- Rule C-False. If $\Gamma \vdash_{CC} false \rightsquigarrow false : Bool$, then by rule C-False, $\Gamma \vdash_{\cap CC} false \rightsquigarrow false : Bool$.

Induction step:

- Rule C-Abs. If $\Gamma \vdash_{CC} \lambda x : T_1\ .\ e \rightsquigarrow \lambda x : T_1\ .\ e' : T_1 \to T_2$, then by rule C-Abs, $\Gamma, x : T_1 \vdash_{CC} e \rightsquigarrow e' : T_2$. By the induction hypothesis, $\Gamma, x : T_1 \vdash_{\cap CC} e \rightsquigarrow e' : T_2$. Therefore, by rule C-Abs, $\Gamma \vdash_{\cap CC} \lambda x : T_1\ .\ e \rightsquigarrow \lambda x : T_1\ .\ e' : T_1 \to T_2$.

- Rule C-App. If $\Gamma \vdash_{CC} e_1\ e_2 \rightsquigarrow (e_1' : PM \Rightarrow^l T_1 \to T_2)\ (e_2' : T_1' \Rightarrow^l T_1) : T_2$, then by rule C-App, $\Gamma \vdash_{CC} e_1 \rightsquigarrow e_1' : PM$, $PM \rhd T_1 \to T_2$, $\Gamma \vdash_{CC} e_2 \rightsquigarrow e_2' : T_1'$ and $T_1' \sim T_1$. By the induction hypothesis, $\Gamma \vdash_{\cap CC} e_1 \rightsquigarrow e_1' : PM$ and $\Gamma \vdash_{\cap CC} e_2 \rightsquigarrow e_2' : T_1'$. By definition of $\trianglelefteq$, $PM \trianglelefteq \{PM\}$, $T_1 \to T_2 \trianglelefteq \{T_1 \to T_2\}$, $T_1' \trianglelefteq \{T_1'\}$ and $T_1 \trianglelefteq \{T_1\}$. By the definition of $\hookrightarrow$, $\{PM\}$, $\{T_1 \to T_2\}$, $e_1' \hookrightarrow e_1' : \varnothing\ PM\ ^0 : PM \Rightarrow^l T_1 \to T_2\ ^0$ and $\{T_1'\}$, $\{T_1\}$, $e_2' \hookrightarrow e_2' : \varnothing\ T_1'\ ^0 : T_1' \Rightarrow^l T_1\ ^0$. Therefore, $\Gamma \vdash_{\cap CC} e_1\ e_2 \rightsquigarrow (e_1' : \varnothing\ PM\ ^0 : PM \Rightarrow^l T_1 \to T_2\ ^0)\ (e_2' : \varnothing\ T_1'\ ^0 : T_1' \Rightarrow^l T_1\ ^0) : T_2$. By the definition of $=_c$, $(e_1' : PM \Rightarrow^l T_1 \to T_2) =_c (e_1' : \varnothing\ PM\ ^0 : PM \Rightarrow^l T_1 \to T_2\ ^0)$ and $(e_2' : T_1' \Rightarrow^l T_1) =_c (e_2' : \varnothing\ T_1'\ ^0 : T_1' \Rightarrow^l T_1\ ^0)$. Therefore, $(e_1' : PM \Rightarrow^l T_1 \to T_2)\ (e_2' : T_1' \Rightarrow^l T_1) =_c (e_1' : \varnothing\ PM\ ^0 : PM \Rightarrow^l T_1 \to T_2\ ^0)\ (e_2' : \varnothing\ T_1'\ ^0 : T_1' \Rightarrow^l T_1\ ^0)$.

We will now prove the left direction of the implication, that if $\Gamma \vdash_{\cap CC} e \rightsquigarrow e_2 : T$ then $\Gamma \vdash_{CC} e \rightsquigarrow e_1 : T$ and $e_1 =_c e_2$. We proceed by induction on the length of the derivation tree of $\Gamma \vdash_{\cap CC} e \rightsquigarrow e_2 : T$.

Base cases:

- Rule C-Var. If $\Gamma \vdash_{\cap CC} x^T \rightsquigarrow x^T : T$, then by rule C-Var, $x : T \in \Gamma$ and $T \subseteq T$. Therefore, by rule C-Var, $\Gamma \vdash_{CC} x \rightsquigarrow x : T$.
- Rule C-Int. If $\Gamma \vdash_{\cap CC} n \rightsquigarrow n : Int$, then by rule C-Int, $\Gamma \vdash_{CC} n \rightsquigarrow n : Int$.
- Rule C-True. If $\Gamma \vdash_{\cap CC} true \rightsquigarrow true : Bool$, then by rule C-True, $\Gamma \vdash_{CC} true \rightsquigarrow true : Bool$.
- Rule C-False. If $\Gamma \vdash_{\cap CC} false \rightsquigarrow false : Bool$, then by rule C-False, $\Gamma \vdash_{CC} false \rightsquigarrow false : Bool$.

Induction step:

- Rule C-Abs. If $\Gamma \vdash_{\cap CC} \lambda x : T_1\ .\ e \rightsquigarrow \lambda x : T_1\ .\ e' : T_1 \to T_2$, then by rule C-Abs, $\Gamma, x : T_1 \vdash_{\cap CC} e \rightsquigarrow e' : T_2$. By the induction hypothesis, $\Gamma, x : T_1 \vdash_{CC} e \rightsquigarrow e' : T_2$. Therefore, by rule C-Abs, $\Gamma \vdash_{CC} \lambda x : T_1\ .\ e \rightsquigarrow \lambda x : T_1\ .\ e' : T_1 \to T_2$.
- Rule C-Abs' If $\Gamma \vdash_{\cap CC} \lambda x : T_1\ .\ e \rightsquigarrow \lambda x : T_1\ .\ e' : T_1 \to T_2$, then by rule C-Abs', $\Gamma, x : T_1 \vdash_{\cap CC} e \rightsquigarrow e' : T_2$. By the induction hypothesis, $\Gamma, x : T_1 \vdash_{CC} e \rightsquigarrow e' : T_2$. Therefore, by rule C-Abs, $\Gamma \vdash_{CC} \lambda x : T_1\ .\ e \rightsquigarrow \lambda x : T_1\ .\ e' : T_1 \to T_2$.
- Rule C-App. If $\Gamma \vdash_{\cap CC} e_1\ e_2 \rightsquigarrow e_1''\ e_2'' : T_2$ then by rule C-App, $\Gamma \vdash_{\cap CC} e_1 \rightsquigarrow e_1' : PM$, $PM \rhd T_1 \to T_2$, $\Gamma \vdash_{\cap CC} e_2 \rightsquigarrow e_2' : T_1'$, $T_1' \sim T_1$, $PM \trianglelefteq S_1$, $T_1 \to T_2 \trianglelefteq S_2$, $T_1' \trianglelefteq S_3$, $T_1 \trianglelefteq S_4$, S_1, S_2, $e_1' \hookrightarrow e_1''$ and S_3, S_4, $e_2' \hookrightarrow e_2''$. Since $e_1\ e_2$ is annotated with only simple types, then by the definition of $\trianglelefteq$, $e_1'' = (e_1' : \varnothing\ PM\ ^0 : PM \Rightarrow^l T_1 \to T_2\ ^0)$ and $e_2'' = (e_2' : \varnothing\ T_1'\ ^0 : T_1' \Rightarrow^l T_1\ ^0)$. By the induction hypothesis, $\Gamma \vdash_{CC} e_1 \rightsquigarrow e_1' : PM$ and $\Gamma \vdash_{CC} e_2 \rightsquigarrow e_2' : T_1'$. Therefore, by rule C-App, $\Gamma \vdash_{CC} e_1\ e_2 \rightsquigarrow (e_1' : PM \Rightarrow^l T_1 \to T_2)\ (e_2' : T_1' \Rightarrow^l T_1) : T_2$. By the definition of $=_c$, $(e_1' : PM \Rightarrow^l T_1 \to T_2) =_c (e_1' : \varnothing\ PM\ ^0 : PM \Rightarrow^l T_1 \to T_2\ ^0)$ and $(e_2' : T_1' \Rightarrow^l T_1) =_c (e_2' : \varnothing\ T_1'\ ^0 : T_1' \Rightarrow^l T_1\ ^0)$. Therefore, $(e_1' : PM \Rightarrow^l T_1 \to T_2)\ (e_2' : T_1' \Rightarrow^l T_1) =_c (e_1' : \varnothing\ PM\ ^0 : PM \Rightarrow^l T_1 \to T_2\ ^0)\ (e_2' : \varnothing\ T_1'\ ^0 : T_1' \Rightarrow^l T_1\ ^0)$.
- Rule C-Gen. If $\Gamma \vdash_{\cap CC} e \rightsquigarrow e' : T$ then by rule C-Gen, $\Gamma \vdash_{\cap CC} e \rightsquigarrow e' : T$. By the induction hypothesis, $\Gamma \vdash_{CC} e \rightsquigarrow e' : T$.
- Rule C-Inst. If $\Gamma \vdash_{\cap CC} e \rightsquigarrow e' : T$ then by rule C-Inst, $\Gamma \vdash_{\cap CC} e \rightsquigarrow e' : T$. By the induction hypothesis, $\Gamma \vdash_{CC} e \rightsquigarrow e' : T$.

□

Theorem B.11 (Conservative Extension to the GTLC)**.** *Depends on Theorem B.7. If e_2 are annotated with only simple types, T is a simple type, $\Gamma \vdash_{CC} e_1 : T$, $\Gamma \vdash_{\cap CC} e_2 : T$ and $e_1 =_c e_2$ then $e_1 \longrightarrow_{CC} e_1' \iff e_2 \longrightarrow_{\cap CC} e_2'$, and $e_1' =_c e_2'$.*

Proof. We will first prove the right direction of the implication, that if $e_1 \longrightarrow_{CC} e_1'$ then $e_2 \longrightarrow^*_{\cap CC} e_2'$ and $e_1 =_c e_2$. We proceed by induction on the length of the derivation tree of $e_1 =_c e_2$.

Base cases:

- $x =_c x^T$. As x doesn't reduce by $\longrightarrow_{CC}$, this case is not considered.
- $n =_c n$. As n doesn't reduce by $\longrightarrow_{CC}$, this case is not considered.
- $true =_c true$. As $true$ doesn't reduce by $\longrightarrow_{CC}$, this case is not considered.
- $false =_c false$. As $false$ doesn't reduce by $\longrightarrow_{CC}$, this case is not considered.
- $blame_T\ l =_c blame_T\ l$. As $blame_T\ l$ doesn't reduce by $\longrightarrow_{CC}$, this case is not considered.
- $blame_T\ l =_c e : (blame\ T'\ T\ l^{\ cl})$. As $blame_T\ l$ doesn't reduce by $\longrightarrow_{CC}$, this case is not considered.

Induction step:

- $\lambda x : T\ .\ e =_c \lambda x : T\ .\ e'$. As $\lambda x : T\ .\ e$ doesn't reduce by $\longrightarrow_{CC}$, this case is not considered.
- $e_1\ e_2 =_c e_3\ e_4$. There are six possibilities:
  - Rule E-PushBlame1. If $blame_{T' \to T}\ l\ e_2 = e_3\ e_4$ and $blame_{T' \to T}\ l\ e_2 \longrightarrow_{CC} blame_T\ l$ then by the definition of $=_c$, $blame_{T' \to T}\ l =_c e_3$. There are two possibilities. By the definition of $=_c$ and by applying rule E-RemoveEmpty zero or more times, either
    - $e_3 \longrightarrow^*_{\cap CC} blame_{T' \to T}\ l$. By rule E-App1, $e_3\ e_4 \longrightarrow^*_{\cap CC} blame_{T' \to T}\ l\ e_4$. By rule E-PushBlame1, $blame_{T' \to T}\ l\ e_4 \longrightarrow^*_{\cap CC} blame_T\ l$ and $blame_T\ l =_c blame_T\ l$.
    - $e_3 \longrightarrow^*_{\cap CC} e : (blame\ T''\ (T' \to T)\ l^{\ cl})$. By repeated application of rule E-Evaluate and by Theorem B.7, $e : blame\ T''\ (T' \to T)\ l^{\ cl}) \longrightarrow^*_{\cap CC} v : blame\ T''\ (T' \to T)\ l^{\ cl})$. By rule E-PropagateBlame, $v : blame\ T''\ (T' \to T)\ l^{\ cl}) \longrightarrow^*_{\cap CC} blame_{T' \to T}\ l$. By rule E-App1, $e_3\ e_4 \longrightarrow^*_{\cap CC} blame_{T' \to T}\ l\ e_4$. By rule E-PushBlame1, $blame_{T' \to T}\ l\ e_4 \longrightarrow^*_{\cap CC} blame_T\ l$ and $blame_T\ l =_c blame_T\ l$.
  - Rule E-PushBlame2. If $e_1\ blame_{T'}\ l = e_3\ e_4$ and $e_1\ blame_{T'}\ l \longrightarrow_{CC} blame_T\ l$ then by the definition of $=_c$, $blame_{T'}\ l =_c e_4$. There are two possibilities. By the definition of $=_c$ and by applying rule E-RemoveEmpty zero or more times, either
    - $e_4 \longrightarrow^*_{\cap CC} blame_{T'}\ l$. By rule E-App2, $e_3\ e_4 \longrightarrow^*_{\cap CC} e_3\ blame_{T'}\ l$. By rule E-PushBlame2, $e_3\ blame_{T'}\ l \longrightarrow^*_{\cap CC} blame_T\ l$ and $blame_T\ l =_c blame_T\ l$.
    - $e_4 \longrightarrow^*_{\cap CC} e : blame\ T''\ T'\ l^{\ cl}$. By repeated application of rule E-Evaluate and by Theorem B.7, $e : blame\ T''\ T'\ l^{\ cl} \longrightarrow^*_{\cap CC} v : blame\ T''\ T'\ l^{\ cl}$. By rule E-PropagateBlame, $v : blame\ T''\ T'\ l^{\ cl} \longrightarrow^*_{\cap CC} blame_{T'}\ l$. By rule E-App2, $e_3\ e_4 \longrightarrow^*_{\cap CC} e_3\ blame_{T'}\ l$. By rule E-PushBlame2, $e_3\ blame_{T'}\ l \longrightarrow^*_{\cap CC} blame_T\ l$ and $blame_T\ l =_c blame_T\ l$.
  - Rule E-App1. If $e_1\ e_2 =_c e_3\ e_4$ and $e_1\ e_2 \longrightarrow_{CC} e_1'\ e_2$ then by the definition of $=_c$, $e_1 =_c e_3$ and $e_2 =_c e_4$, and by rule E-App1, $e_1 \longrightarrow_{CC} e_1'$. By the induction hypothesis, $e_3 \longrightarrow_{\cap CC} e_3'$ and $e_1' =_c e_3'$. Then by rule E-App1, $e_3\ e_4 \longrightarrow_{\cap CC} e_3'\ e_4$. By definition of $=_c$, $e_1'\ e_2 =_c e_3'\ e_4$.
  - Rule E-App2. If $v_1\ e_2 =_c e_3\ e_4$ and $v_1\ e_2 \longrightarrow_{CC} v_1\ e_2'$ then by the definition of $=_c$, $v_1 =_c e_3$ and $e_2 =_c e_4$, and by rule E-App2, $e_2 \longrightarrow_{CC} e_2'$. By the induction hypothesis, $e_4 \longrightarrow_{\cap CC} e_4'$ and $e_2' =_c e_4'$. By definition of $=_c$, and by applying rule E-RemoveEmpty zero or more times, $e_3 \longrightarrow^*_{\cap CC} v_1$. If $e_3 \longrightarrow^*_{\cap CC} v_1'$ such that $v_1 =_c v_1'$, by rule E-App1, $e_3\ e_4 \longrightarrow_{\cap CC} v_1'\ e_4$, and by rule E-App2, $v_1'\ e_4 \longrightarrow_{\cap CC} v_1'\ e_4'$. By definition of $=_c$, $v_1\ e_2' =_c v_1'\ e_4'$.

- Rule E-AppAbs. If $(\lambda x : T' \,.\, e)\ v =_c e_3\ e_4$ and $(\lambda x : T' \,.\, e)\ v \longrightarrow_{CC} [x \mapsto v]e$ then by the definition of $=_c$, $(\lambda x : T' \,.\, e) =_c e_3$ and $v =_c e_4$. By the definition of $=_c$ and by applying rule E-RemoveEmpty zero or more times, $e_3 \longrightarrow^*_{\cap CC} \lambda x : T' \,.\, e'$ and $e_4 \longrightarrow^*_{\cap CC} v'$, such that, by definition of $=_c$, $(\lambda x : T' \,.\, e) =_c (\lambda x : T' \,.\, e')$ and $v =_c v'$ and $e =_c e'$. By rule E-AppAbs, $(\lambda x : T' \,.\, e')\ v' \longrightarrow_{\cap CC} [x \mapsto v']e'$ and by definition of $=_c$, $[x \mapsto v]e =_c [x \mapsto v']e'$.
- Rule C-BETA. If $(v_1 : T_1 \to T_2 \Rightarrow^l T_3 \to T_4)\ v_2 =_c e_3\ e_4$ and $(v_1 : T_1 \to T_2 \Rightarrow^l T_3 \to T_4)\ v_2 \longrightarrow_{CC} (v_1\ (v_2 : T_3 \Rightarrow^l T_1)) : T_2 \Rightarrow^l T_4$ then by the definition of $=_c$, $v_1 : T_1 \to T_2 \Rightarrow^l T_3 \to T_4 =_c e_3$ and $v_2 =_c e_4$. By definition of $=_c$ and by applying rule E-RemoveEmpty zero or more times, $e_3 \longrightarrow^*_{\cap CC} v'_1 : (\varnothing\ T_1 \to T_2\ ^{cl} : T_1 \to T_2 \Rightarrow^l T_3 \to T_4)$ such that $v_1 =_c v'_1$, and $e_4 \longrightarrow^*_{\cap CC} v'_2$ such that $v_2 =_c v'_2$. By rule E-SimulateArrow, $(v'_1 : (\varnothing\ T_1 \to T_2\ ^{cl} : T_1 \to T_2 \Rightarrow^l T_3 \to T_4))\ v'_2 \longrightarrow_{\cap CC} ((v'_1 : \varnothing\ T_1 \to T_2\ ^{cl})\ (v'_2 : (\varnothing\ T_3\ ^0 : T_3 \Rightarrow^l T_1\ ^0))) : (\varnothing\ T_2\ ^0 : T_2 \Rightarrow^l T_4\ ^0)$. By the definition of $=_c$, $(v_1\ (v_2 : T_3 \Rightarrow^l T_1)) : T_2 \Rightarrow^l T_4 =_c ((v'_1 : \varnothing\ T_1 \to T_2\ ^{cl})\ (v'_2 : (\varnothing\ T_3\ ^0 : T_3 \Rightarrow^l T_1\ ^0))) : (\varnothing\ T_2\ ^0 : T_2 \Rightarrow^l T_4\ ^0)$.

• $e_1 =_c e_2 : (\varnothing\ T\ ^{cl})$. If $e_1 =_c e_2 : \varnothing\ T\ ^{cl}$ and $e_1 \longrightarrow_{CC} e'_1$ then by the definition of $=_c$, $e_1 =_c e_2$. By the induction hypothesis, $e_2 \longrightarrow_{\cap CC} e'_2$ and $e'_1 =_c e'_2$. By rule E-Evaluate, $e_2 : \varnothing\ T\ ^{cl} \longrightarrow_{\cap CC} e'_2 : \varnothing\ T\ ^{cl}$. As $e'_1 =_c e'_2$ then by definition of $=_c$, $e'_1 =_c e'_2 : \varnothing\ T\ ^{cl}$.

• $e : T_1 \Rightarrow^l T_2 =_c e' : (c : T_1 \Rightarrow^l T_2\ ^{cl})$. There are seven possibilities:

- Rule E-Evaluate. If $e_1 : T_1 \Rightarrow^l T_2 =_c e$ and $e_1 : T_1 \Rightarrow^l T_2 \longrightarrow_{CC} e'_1 : T_1 \Rightarrow^l T_2$, then by the definition of $=_c$ and by applying rule E-Evaluate zero or more times, $e \longrightarrow^*_{\cap CC} e_2 : (c : T_1 \Rightarrow^l T_2\ ^{cl})$ such that $e_1 =_c e_2 : c$, and by rule E-Evaluate, $e_1 \longrightarrow_{CC} e'_1$. By the induction hypothesis, $e_2 : c \longrightarrow^*_{\cap CC} e'_2 : c$ and $e'_1 =_c e'_2 : c$. If $e_2 : c \longrightarrow^*_{\cap CC} e'_2 : c$ then by rule E-Evaluate, $e_2 \longrightarrow^*_{\cap CC} e'_2$. By rule E-Evaluate, $e_2 : (c : T_1 \Rightarrow^l T_2\ ^{cl}) \longrightarrow_{\cap CC} e'_2 : (c : T_1 \Rightarrow^l T_2\ ^{cl})$. As $e'_1 =_c e'_2 : c$ then by the definition of $=_c$, $e'_1 : T_1 \Rightarrow^l T_2 =_c e'_2 : (c : T_1 \Rightarrow^l T_2\ ^{cl})$.
- Rule CTX-BLAME. If $blame_{T_1}\ l : T_1 \Rightarrow^l T_2 =_c e$ and $blame_{T_1}\ l : T_1 \Rightarrow^l T_2 \longrightarrow_{CC} blame_{T_2}\ l$ then there are three possibilities. By the definition of $=_c$ and by applying rule E-RemoveEmpty zero or more times, either
  * $e \longrightarrow^*_{\cap CC} blame_{T_1}\ l : (\varnothing\ T_1\ ^{cl} : T_1 \Rightarrow^l T_2\ ^{cl})$. By rule E-PushBlameCast, $blame_{T_1}\ l : (\varnothing\ T_1\ ^{cl} : T_1 \Rightarrow^l T_2\ ^{cl}) \longrightarrow_{\cap CC} blame_{T_2}\ l$ and $blame_{T_2}\ l =_c blame_{T_2}\ l$.
  * $e \longrightarrow^*_{\cap CC} e' : (blame\ T'\ T_1\ l\ ^{cl} : T_1 \Rightarrow^l T_2\ ^{cl})$. By repeated application of rule E-Evaluate and by Theorem B.7, $e' : (blame\ T'\ T_1\ l\ ^{cl} : T_1 \Rightarrow^l T_2\ ^{cl}) \longrightarrow^*_{\cap CC} v : (blame\ T'\ T_1\ l\ ^{cl} : T_1 \Rightarrow^l T_2\ ^{cl})$. By rule E-EvaluateCasts and by rule E-PushBlameCI, $v : (blame\ T'\ T_1\ l\ ^{cl} : T_1 \Rightarrow^l T_2\ ^{cl}) \longrightarrow^*_{\cap CC} v : (blame\ T'\ T_2\ l\ ^{cl})$. By rule E-PropagateBlame, $v : (blame\ T'\ T_2\ l\ ^{cl}) \longrightarrow^*_{\cap CC} blame_{T_2}\ l$ and $blame_{T_2}\ l =_c blame_{T_2}\ l$.
  * $e \longrightarrow^*_{\cap CC} e' : (blame\ T'\ T_1\ l\ ^{cl}) : (\varnothing\ T_1\ ^{cl} : T_1 \Rightarrow^l T_2\ ^{cl})$. By repeated application of rule E-Evaluate and by Theorem B.7, $e' : (blame\ T'\ T_1\ l\ ^{cl} : T_1 \Rightarrow^l T_2\ ^{cl}) \longrightarrow^*_{\cap CC} v : (blame\ T'\ T_1\ l\ ^{cl}) : (\varnothing\ T_1\ ^{cl} : T_1 \Rightarrow^l T_2\ ^{cl})$. By rule E-MergeCasts, $v : (blame\ T'\ T_1\ l\ ^{cl}) : (\varnothing\ T_1\ ^{cl} : T_1 \Rightarrow^l T_2\ ^{cl}) \longrightarrow_{\cap CC} v : (blame\ T'\ T_1\ l\ ^{cl} : T_1 \Rightarrow^l T_2\ ^{cl})$. By rule E-EvaluateCasts and by rule E-PushBlameCI, $v : (blame\ T'\ T_1\ l\ ^{cl} : T_1 \Rightarrow^l T_2\ ^{cl}) \longrightarrow^*_{\cap CC} v : (blame\ T'\ T_2\ l\ ^{cl})$. By rule E-PropagateBlame, $v : (blame\ T'\ T_2\ l\ ^{cl}) \longrightarrow^*_{\cap CC} blame_{T_2}\ l)$ and $blame_{T_2}\ l =_c blame_{T_2}\ l$.
- Rule ID-BASE and Rule ID-STAR. If $v : T \Rightarrow^l T =_c e$ and $v : T \Rightarrow^l T \longrightarrow_{CC} v$, then by the definition of $=_c$ and by applying rule E-RemoveEmpty zero or more times, $e \longrightarrow^*_{\cap CC} v' : (cv : T \Rightarrow^l T\ ^{cl})$, such that $v =_c v' : cv$. By rule E-EvaluateCasts and by rule E-IdentityCI, $v' : (cv : T \Rightarrow^l T\ ^{cl}) \longrightarrow_{\cap CC} v' : cv$ and $v =_c v' : cv$.

- Rule SUCCEED. If $v : G \Rightarrow^{l_1} Dyn : Dyn \Rightarrow^{l_2} G =_c e$ and $v : G \Rightarrow^{l_1} Dyn : Dyn \Rightarrow^{l_2} G \longrightarrow_{CC} v$ then there are two possibilities. By definition of $=_c$ and by applying rule E-RemoveEmpty zero or more times, either
  - $e \longrightarrow^*_{\cap CC} v' : (cv : G \Rightarrow^{l_1} Dyn\ ^{cl} : Dyn \Rightarrow^{l_2} G\ ^{cl})$ or
  - $e \longrightarrow^*_{\cap CC} v' : (cv : G \Rightarrow^{l_1} Dyn\ ^{cl}) : (\varnothing\ Dyn\ ^{cl} : Dyn \Rightarrow^{l_2} G\ ^{cl})$

  such that $v =_c v' : cv$. As, by rule E-MergeCasts, $v' : (cv : G \Rightarrow^{l_1} Dyn\ ^{cl}) : (\varnothing\ Dyn\ ^{cl} : Dyn \Rightarrow^{l_2} G\ ^{cl}) \longrightarrow_{\cap CC} v' : (cv : G \Rightarrow^{l_1} Dyn\ ^{cl} : Dyn \Rightarrow^{l_2} G\ ^{cl})$, we only need to address the first case. By rule E-EvaluateCasts and by rule E-SucceedCI, $v' : (cv : G \Rightarrow^{l_1} Dyn\ ^{cl} : Dyn \Rightarrow^{l_2} G\ ^{cl}) \longrightarrow_{\cap CC} v' : cv$ and $v =_c v' : cv$.
- Rule FAIL. If $v : G_1 \Rightarrow^{l_1} Dyn : Dyn \Rightarrow^{l_2} G_2 =_c e$ and $v : G_1 \Rightarrow^{l_1} Dyn : Dyn \Rightarrow^{l_2} G_2 \longrightarrow_{CC} blame_{G_2}\ l_2$ then there are two possibilities. By definition of $=_c$ and by applying rule E-RemoveEmpty zero or more times, either
  - $e \longrightarrow^*_{\cap CC} v' : (cv : G_1 \Rightarrow^{l_1} Dyn\ ^{cl} : Dyn \Rightarrow^{l_2} G_2\ ^{cl})$ or
  - $e \longrightarrow^*_{\cap CC} v' : (cv : G_1 \Rightarrow^{l_1} Dyn\ ^{cl}) : (\varnothing\ Dyn\ ^{cl} : Dyn \Rightarrow^{l_2} G_2\ ^{cl})$

  such that $v =_c v' : cv$. As, by rule E-MergeCasts, $v' : (cv : G_1 \Rightarrow^{l_1} Dyn\ ^{cl}) : (\varnothing\ Dyn\ ^{cl} : Dyn \Rightarrow^{l_2} G_2\ ^{cl}) \longrightarrow_{\cap CC} v' : (cv : G_1 \Rightarrow^{l_1} Dyn\ ^{cl} : Dyn \Rightarrow^{l_2} G_2\ ^{cl})$, we only need to address the first case. By rule E-EvaluateCasts and by rule E-FailCI, $v' : (cv : G_1 \Rightarrow^{l_1} Dyn\ ^{cl} : Dyn \Rightarrow^{l_2} G_2\ ^{cl}) \longrightarrow_{\cap CC} v' : blame\ T_I\ G_2\ l_2\ ^{cl}$. By rule E-PropagateBlame, $v' : blame\ T_I\ G_2\ l_2\ ^{cl} \longrightarrow_{\cap CC} blame_{G_2}\ l_2$ and $blame_{G_2}\ l_2 =_c blame_{G_2}\ l_2$.
- Rule GROUND. If $v : T \Rightarrow^l Dyn =_c e$ and $v : T \Rightarrow^l Dyn \longrightarrow_{CC} v : T \Rightarrow^l G : G \Rightarrow^l Dyn$ then by definition of $=_c$ and by applying rule E-RemoveEmpty zero or more times, $e \longrightarrow^*_{\cap CC} v' : (cv : T \Rightarrow^l Dyn\ ^{cl})$ such that $v =_c v' : cv$. By rule E-EvaluateCasts and by rule E-GroundCI, $v' : (cv : T \Rightarrow^l Dyn\ ^{cl}) \longrightarrow_{\cap CC} v' : (cv : T \Rightarrow^l G\ ^{cl} : G \Rightarrow^l Dyn\ ^{cl})$. As $v =_c v' : cv$, then by definition of $=_c$, $v : T \Rightarrow^l G : G \Rightarrow^l Dyn =_c v' : (cv : T \Rightarrow^l G\ ^{cl} : G \Rightarrow^l Dyn\ ^{cl})$.
- Rule EXPAND. If $v : Dyn \Rightarrow^l T =_c e$ and $v : Dyn \Rightarrow^l T \longrightarrow_{CC} v : Dyn \Rightarrow^l G : G \Rightarrow^l T$ then by definition of $=_c$ and by applying rule E-RemoveEmpty zero or more times, $e \longrightarrow^*_{\cap CC} v' : (cv : Dyn \Rightarrow^l T\ ^{cl})$ such that $v =_c v' : cv$. By rule E-EvaluateCasts and by rule E-ExpandCI, $v' : (cv : Dyn \Rightarrow^l T\ ^{cl}) \longrightarrow_{\cap CC} v' : (cv : Dyn \Rightarrow^l G\ ^{cl} : G \Rightarrow^l T\ ^{cl})$. As $v =_c v' : cv$, then by definition of $=_c$, $v : Dyn \Rightarrow^l G : G \Rightarrow^l T =_c v' : (cv : Dyn \Rightarrow^l G\ ^{cl} : G \Rightarrow^l T\ ^{cl})$.

We will now prove the left direction of the implication, that if $e_2 \longrightarrow_{\cap CC} e_2'$ then $e_1 \longrightarrow_{CC} e_1'$ and $e_1 =_c e_2$. We proceed by induction on the length of the derivation tree of $e_1 =_c e_2$.

Base cases:

- $x =_c x^T$. As x^T doesn't reduce by $\longrightarrow_{\cap CC}$, this case is not considered.
- $n =_c n$. As n doesn't reduce by $\longrightarrow_{\cap CC}$, this case is not considered.
- $true =_c true$. As $true$ doesn't reduce by $\longrightarrow_{\cap CC}$, this case is not considered.
- $false =_c false$. As $false$ doesn't reduce by $\longrightarrow_{\cap CC}$, this case is not considered.
- $blame_T\ l =_c blame_T\ l$. As $blame_T\ l$ doesn't reduce by $\longrightarrow_{\cap CC}$, this case is not considered.
- $blame_T\ l =_c e : (blame\ T'\ T\ l\ ^{cl})$. There are two possibilities:
  - Rule E-Evaluate. If $e : (blame\ T'\ T\ l\ ^{cl}) \longrightarrow_{\cap CC} e' : (blame\ T'\ T\ l\ ^{cl})$ and as $blame_T\ l$ is already a value, then $blame_T\ l =_c' e : (blame\ T'\ T\ l\ ^{cl})$.
  - Rule E-PropagateBlame. If $v : (blame\ T'\ T\ l\ ^{cl}) \longrightarrow_{\cap CC} blame_T\ l$ and as $blame_T\ l$ is already a value, then $blame_T\ l =_c blame_T\ l$.

Induction step:

- $\lambda x : T\ .\ e =_c \lambda x : T\ .\ e'$. As $\lambda x : T\ .\ e'$ doesn't reduce by $\longrightarrow_{\cap CC}$, this case is not considered.
- $e_1\ e_2 =_c e_3\ e_4$. There are 6 possibilities:
  - Rule E-PushBlame1. If $blame_{T' \to T}\ l\ e_2 = blame_{T' \to T}\ l\ e_4$ and $blame_{T' \to T}\ l\ e_4 \longrightarrow_{\cap CC} blame_T\ l$ then by rule E-PushBlame1, $blame_{T' \to T}\ l\ e_2 \longrightarrow_{CC} blame_T\ l$ and $blame_T\ l =_c blame_T\ l$.
  - Rule E-PushBlame2. If $e_1\ blame_{T'}\ l = e_3\ blame_{T'}\ l$ and $e_3\ blame_{T'}\ l \longrightarrow_{\cap CC} blame_T\ l$ then by rule E-PushBlame2, $e_1\ blame_{T'}\ l \longrightarrow_{CC} blame_T\ l$ and $blame_T\ l =_c blame_T\ l$.
  - Rule E-App1. If $e_1\ e_2 =_c e_3\ e_4$ and $e_3\ e_4 \longrightarrow_{\cap CC} e_3'\ e_4$ then by the definition of $=_c$, $e_1 =_c e_3$ and $e_2 =_c e_4$, and by rule E-App1, $e_3 \longrightarrow_{\cap CC} e_3'$. By the induction hypothesis, $e_1 \longrightarrow_{CC} e_1'$ and $e_1' =_c e_3'$. Then by rule E-App1, $e_1\ e_2 \longrightarrow_{CC} e_1'\ e_2$. By definition of $=_c$, $e_1'\ e_2 =_c e_3'\ e_4$.
  - Rule E-App2. If $v_1\ e_2 =_c v_3\ e_4$ and $v_3\ e_4 \longrightarrow_{\cap CC} v_3\ e_4'$ then by the definition of $=_c$, $v_1 =_c v_3$ and $e_2 =_c e_4$, and by rule E-App2, $e_4 \longrightarrow_{\cap CC} e_4'$. By the induction hypothesis, $e_2 \longrightarrow_{CC} e_2'$ and $e_2' =_c e_4'$. Then by rule E-App2, $v_1\ e_2 \longrightarrow_{CC} v_1\ e_2'$. By definition of $=_c$, $v_1\ e_2' =_c v_3\ e_4'$.
  - Rule E-AppAbs. If $(\lambda x : T'\ .\ e)\ v_2 =_c (\lambda x : T'\ .\ e')\ v_4$ and $(\lambda x : T'\ .\ e')\ v_4 \longrightarrow_{\cap CC} [x \mapsto v_4]e'$ then by the definition of $=_c$, $(\lambda x : T'\ .\ e) =_c (\lambda x : T'\ .\ e')$ and $v_2 =_c v_4$ and $e =_c e'$. By rule E-AppAbs, $(\lambda x : T'\ .\ e)\ v_2 \longrightarrow_{CC} [x \mapsto v_2]e$. As $v_2 =_c v_4$ and $e =_c e'$, then by definition of $=_c$, $[x \mapsto v_2]e =_c [x \mapsto v_4]e'$.
  - Rule E-SimulateArrow. There are two possibilities:
    * If $v_1\ v_2 =_c (v_3 : \varnothing\ T' \to T\ ^{cl})\ v_4$ and $(v_3 : \varnothing\ T' \to T\ ^{cl})\ v_4 \longrightarrow_{\cap CC} ((v_3 : \varnothing\ T' \to T\ ^{cl})\ (v_4 : \varnothing\ T'\ ^{cl})) : \varnothing\ T\ ^{cl}$ then by definition of $=_c$, $v_1 =_c (v_3 : \varnothing\ T' \to T\ ^{cl})$ and $v_2 =_c v_4$ and $v_1 =_c v_3$. By the definition of $=_c$, $v_2 =_c v_4 : \varnothing\ T'\ ^{cl}$. By the definition of $=_c$, $v_1\ v_2 =_c ((v_3 : \varnothing\ T' \to T\ ^{cl})\ (v_4 : \varnothing\ T'\ ^{cl}))$. By the definition of $=_c$, $v_1\ v_2 =_c ((v_3 : \varnothing\ T' \to T\ ^{cl})\ (v_4 : \varnothing\ T'\ ^{cl})) : \varnothing\ T\ ^{cl}$.
    * If $(v_1 : T_1 \to T_2 \Rightarrow^l T_3 \to T_4)\ v_2 =_c (v_3 : (cv : T_1 \to T_2 \Rightarrow^l T_3 \to T_4\ ^{cl}))\ v_4$ and $(v_3 : (cv : T_1 \to T_2 \Rightarrow^l T_3 \to T_4\ ^{cl}))\ v_4 \longrightarrow_{\cap CC} ((v_3 : cv)\ (v_4 : (\varnothing\ T_3\ ^{cl} : T_3 \Rightarrow^l T_1\ ^{cl}))) : (\varnothing\ T\ ^{cl} : T_2 \Rightarrow^l T_4\ ^{cl})$ then by definition of $=_c$, $v_1 =_c v_3 : cv$ and $v_2 =_c v_4$. By rule C-BETA, $(v_1 : T_1 \to T_2 \Rightarrow^l T_3 \to T_4)\ v_2 \longrightarrow_{CC} (v_1\ (v_2 : T_3 \Rightarrow^l T_1)) : T_2 \Rightarrow^l T_4$. As $v_2 =_c v_4$, then by definition of $=_c$, $v_2 : T_3 \Rightarrow^l T_1 =_c v_4 : (\varnothing\ T_3\ ^{cl} : T_3 \Rightarrow^l T_1\ ^{cl})$. As $v_1 =_c v_3 : cv$ and $v_2 : T_3 \Rightarrow^l T_1 =_c v_4 : (\varnothing\ T_3\ ^{cl} : T_3 \Rightarrow^l T_1\ ^{cl})$, then by the definition of $=_c$, $(v_1\ (v_2 : T_3 \Rightarrow^l T_1)) =_c ((v_3 : cv)\ (v_4 : (\varnothing\ T_3\ ^{cl} : T_3 \Rightarrow^l T_1\ ^{cl})))$. As $(v_1\ (v_2 : T_3 \Rightarrow^l T_1)) =_c ((v_3 : cv)\ (v_4 : (\varnothing\ T_3\ ^{cl} : T_3 \Rightarrow^l T_1\ ^{cl})))$, then by the definition of $=_c$, $(v_1\ (v_2 : T_3 \Rightarrow^l T_1)) : T_2 \Rightarrow^l T_4 =_c ((v_3 : cv)\ (v_4 : (\varnothing\ T_3\ ^{cl} : T_3 \Rightarrow^l T_1\ ^{cl}))) : (\varnothing\ T\ ^{cl} : T_2 \Rightarrow^l T_4\ ^{cl})$.
- $e_1 =_c e_2 : (\varnothing\ T\ ^{cl})$. There are two possibilities:
  - Rule E-Evaluate. If $e_1 =_c e_2 : (\varnothing\ T\ ^{cl})$ and $e_2 : (\varnothing\ T\ ^{cl}) \longrightarrow_{\cap CC} e_2' : (\varnothing\ T\ ^{cl})$ then by the definition of $=_c$, $e_1 =_c e_2$, and by rule E-Evaluate, $e_2 \longrightarrow_{\cap CC} e_2'$. By the induction hypothesis, $e_1 \longrightarrow_{CC} e_1'$ and $e_1' =_c e_2'$. As $e_1' =_c e_2'$ then by definition of $=_c$, $e_1' =_c e_2' : (\varnothing\ T\ ^{cl})$.
  - Rule E-RemoveEmpty. If $v_1 =_c v_2 : (\varnothing\ T\ ^{cl})$ and $v_2 : (\varnothing\ T\ ^{cl}) \longrightarrow_{\cap CC} v_2$ then by the definition of $=_c$, $v_1 =_c v_2$.
- $e : T_1 \Rightarrow^l T_2 =_c e' : (c : T_1 \Rightarrow^l T_2\ ^{cl})$. There are four possibilities:
  - Rule E-PushBlameCast. If $blame_{T_1}\ l : T_1 \Rightarrow^l T_2 =_c blame_{T_1}\ l : (c : T_1 \Rightarrow^l T_2\ ^{cl})$ and $blame_{T_1}\ l : (c : T_1 \Rightarrow^l T_2\ ^{cl}) \longrightarrow_{\cap CC} blame_{T_2}\ l$ then by rule CTX-BLAME, $blame_{T_1}\ l : T_1 \Rightarrow^l T_2 \longrightarrow_{CC} blame_{T_2}\ l$ and $blame_{T_2}\ l =_c blame_{T_2}\ l$.

- Rule E-Evaluate. If $e_1 : T_1 \Rightarrow^l T_2 =_c e_2 : (c : T_1 \Rightarrow^l T_2\ ^{cl})$ and $e_2 : (c : T_1 \Rightarrow^l T_2\ ^{cl}) \longrightarrow_{\cap CC} e_2' : (c : T_1 \Rightarrow^l T_2\ ^{cl})$ then by definition of $=_c$, $e_1 =_c e_2 : c$, and by rule E-Evaluate, $e_2 \longrightarrow_{\cap CC} e_2'$. By rule E-Evaluate, $e_2 : c \longrightarrow_{\cap CC} e_2' : c$. By the induction hypothesis, $e_1 \longrightarrow_{CC} e_1'$ and $e_1' =_c e_2' : c$. By rule E-Evaluate, $e_1 : T_1 \Rightarrow^l T_2 \longrightarrow_{CC} e_1' : T_1 \Rightarrow^l T_2$. As $e_1' =_c e_2' : c$, then by the definition of $=_c$, $e_1' : T_1 \Rightarrow^l T_2 =_c e_2' : (c : T_1 \Rightarrow^l T_2\ ^{cl})$.
- Rule E-MergeCasts. If $v : T_1 \Rightarrow^l T_2 =_c (v' : cv) : (\varnothing\ T_1\ ^{cl} : T_1 \Rightarrow^l T_2\ ^{cl})$ and $(v' : cv) : (\varnothing\ T_1\ ^{cl} : T_1 \Rightarrow^l T_2\ ^{cl}) \longrightarrow_{\cap CC} v' : (cv : T_1 \Rightarrow^l T_2\ ^{cl})$ then by the definition of $=_c$, $v =_c v' : cv$. As $v =_c v' : cv$, then by the definition of $=_c$, $v : T_1 \Rightarrow^l T_2 =_c v' : (cv : T_1 \Rightarrow^l T_2\ ^{cl})$.
- Rule E-EvaluateCasts. There are seven possibilities:
  * Rule E-PushBlameCI. If $blame_{T_1}\ l_1 : T_1 \Rightarrow^{l_2} T_2 =_c v : (blame\ T'\ T_1\ l_1\ ^{cl} : T_1 \Rightarrow^{l_2} T_2\ ^{cl})$ and $v : (blame\ T'\ T_1\ l_1\ ^{cl} : T_1 \Rightarrow^{l_2} T_2\ ^{cl}) \longrightarrow_{\cap CC} v : blame\ T'\ T_2\ l_1\ ^{cl}$ then by rule CTX-BLAME $blame_{T_1}\ l_1 : T_1 \Rightarrow^{l_2} T_2 \longrightarrow_{CC} blame_{T_2}\ l_1$ and $blame_{T_2}\ l_1 =_c v : blame\ T'\ T_2\ l_1\ ^{cl}$.
  * Rule E-EvaluateCI. If $v_1 : T_1 \Rightarrow^l T_2 =_c v_2 : (c : T_1 \Rightarrow^l T_2)$ and $v_2 : (c : T_1 \Rightarrow^l T_2) \longrightarrow_{\cap CC} v_2 : (c' : T_1 \Rightarrow^l T_2)$ then $v_1 =_c v_2 : c$ and by rule E-EvaluateCasts, $v_2 : c \longrightarrow_{\cap CC} v_2 : c'$. By the induction hypothesis, $v_1 \longrightarrow_{CC} v_1'$ and $v_1' =_c v_2 : c'$. By rule E-Evaluate, $v_1 : T_1 \Rightarrow^l T_2 \longrightarrow_{CC} v_1' : T_1 \Rightarrow^l T_2$. As $v_1' =_c v_2 : c'$, then by definition of $=_c$, $v_1' : T_1 \Rightarrow^l T_2 =_c v_2 : (c' : T_1 \Rightarrow^l T_2)$.
  * E-IdentityCI. If $v_1 : T \Rightarrow^l T =_c v_2 : (cv1 : T \Rightarrow^l T)$ and $v_2 : (cv1 : T \Rightarrow^l T) \longrightarrow_{\cap CC} v_2 : cv1$ then by the definition of $=_c$, $v_1 =_c v_2 : cv1$. By rule ID-BASE or ID-STAR, $v_1 : T \rightarrow^l T \longrightarrow_{CC} v_1$ and $v_1 =_c v_2 : cv1$.
  * E-SucceedCI. If $v_1 : G \Rightarrow^{l_1} Dyn : Dyn \Rightarrow^{l_2} G =_c v_2 : (cv1 : G \Rightarrow^{l_1} Dyn\ ^{cl_1} : Dyn \Rightarrow^{l_2} G\ ^{cl_2})$ and $v_2 : (cv1 : G \Rightarrow^{l_1} Dyn\ ^{cl_1} : Dyn \Rightarrow^{l_2} G\ ^{cl_2}) \longrightarrow_{\cap CC} v_2 : cv1$ then by the definition of $=_c$, $v_1 =_c v_2 : cv1$. By rule SUCCEED, $v_1 : G \Rightarrow^{l_1} Dyn : Dyn \Rightarrow^{l_2} G \longrightarrow_{CC} v_1$ and $v_1 =_c v_2 : cv1$.
  * E-FailCI. If $v_1 : G_1 \Rightarrow^{l_1} Dyn : Dyn \Rightarrow^{l_2} G_2 =_c v_2 : (cv1 : G_1 \Rightarrow^{l_1} Dyn\ ^{cl_1} : Dyn \Rightarrow^{l_2} G_2\ ^{cl_2})$ and $v_2 : (cv1 : G_1 \Rightarrow^{l_1} Dyn\ ^{cl_1} : Dyn \Rightarrow^{l_2} G_2\ ^{cl_2}) \longrightarrow_{\cap CC} v_2 : blame\ T'\ G_2\ l_2\ ^{cl_1}$ then by the definition of $=_c$, $v_1 =_c v_2 : cv1$. By rule FAIL, $v_1 : G_1 \Rightarrow^{l_1} Dyn : Dyn \Rightarrow^{l_2} G_2 \longrightarrow_{CC} blame_{G_2}\ l_2$ and by the definition of $=_c$, $blame_{G_2}\ l_2 =_c v_2 : blame\ T'\ G_2\ l_2\ ^{cl_1}$.
  * E-GroundCI. If $v_1 : T \Rightarrow^l Dyn =_c v_2 : (cv1 : T \Rightarrow^l Dyn\ ^{cl})$ and $v_2 : (cv1 : T \Rightarrow^l Dyn\ ^{cl}) \longrightarrow_{\cap CC} v_2 : (cv1 : T \Rightarrow^l G\ ^{cl} : G \Rightarrow^l Dyn\ ^{cl})$ then by the definition of $=_c$, $v_1 =_c v_2 : cv1$. By rule GROUND, $v_1 : T \Rightarrow^l Dyn \longrightarrow_{CC} v_1 : T \Rightarrow^l G : G \Rightarrow^l Dyn$. As $v_1 =_c v_2 : cv1$, then by the definition of $=_c$, $v_1 : T \Rightarrow^l G =_c v_2 : (cv1 : T \Rightarrow^l G\ ^{cl})$. As $v_1 : T \Rightarrow^l G =_c v_2 : (cv1 : T \Rightarrow^l G\ ^{cl})$, then by the definition of $=_c$, $v_1 : T \Rightarrow^l G : G \Rightarrow^l Dyn =_c v_2 : (cv1 : T \Rightarrow^l G\ ^{cl} : G \Rightarrow^l Dyn\ ^{cl})$.
  * E-ExpandCI. If $v_1 : Dyn \Rightarrow^l T =_c v_2 : (cv1 : Dyn \Rightarrow^l T\ ^{cl})$ and $v_2 : (cv1 : Dyn \Rightarrow^l T\ ^{cl}) \longrightarrow_{\cap CC} v_2 : (cv1 : Dyn \Rightarrow^l G\ ^{cl} : G \Rightarrow^l T\ ^{cl})$ then by the definition of $=_c$, $v_1 =_c v_2 : cv1$. By rule EXPAND, $v_1 : Dyn \Rightarrow^l T \longrightarrow_{CC} v_1 : Dyn \Rightarrow^l G : G \Rightarrow^l T$. As $v_1 =_c v_2 : cv1$, then by the definition of $=_c$, $v_1 : Dyn \Rightarrow^l G =_c v_2 : (cv1 : Dyn \Rightarrow^l G\ ^{cl})$. As $v_1 : Dyn \Rightarrow^l G =_c v_2 : (cv1 : Dyn \Rightarrow^l G\ ^{cl})$, then by the definition of $=_c$, $v_1 : Dyn \Rightarrow^l G : G \Rightarrow^l T =_c v_2 : (cv1 : Dyn \Rightarrow^l G\ ^{cl} : G \Rightarrow^l T\ ^{cl})$.

□